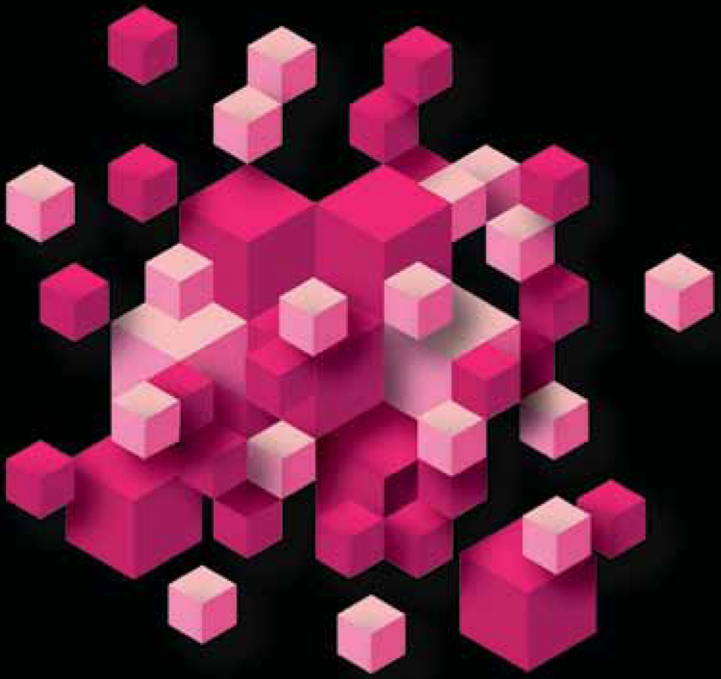
Create captivating games for all Windows 8 devices in C#



Windows 8 and Windows Phone 8 Game Development

Adam Dawes

Apress®

Contents at a Glance

About the Author xxi
About the Technical Reviewer xxiii
Acknowledgments xxv
Introduction xxvii
■Chapter 1: Getting Started 1
■Chapter 2: Getting Started with MonoGame 29
■Chapter 3: Creating a Game Framework 71
■Chapter 4: User Input 111
■Chapter 5: Sounding Out with Game Audio 167
■Chapter 6: Drawing with Vertices and Matrices 179
■Chapter 7: The World of 3-D Graphics 229
■Chapter 8: Further 3-D Features and Techniques 273
■Chapter 9: Enhancing Your Game 327
■Chapter 10: Application Management 351
■Chapter 11: Getting Started with XAML 363
■Chapter 12: XAML Controls and Pages 395
■Chapter 13: Mixing XAML and MonoGame 429
■Chapter 14: Distributing Your Game 439
Index 471

Introduction

The Goal of This Book

The effect of mobile computing in recent years has been impossible to miss. Nearly everyone carries a smartphone with them every second of the day. The last few years have also seen an explosion in the popularity of tablet devices, offering many of the uses and benefits of a phone but with much more screen space and processing power.

With the latest iterations of its desktop and mobile operating systems, Microsoft has started its push to conquer these areas of technology. Alongside the established desktop operating system market, Windows 8 introduces an all-new, touch-screen-orientated user interface that is fast and responsive on the desktop and on tablet devices alike. Windows Phone 8 continues to improve upon Microsoft's phone operating system, offering a smooth and powerful experience on both high- and low-specification devices.

Writing games for mobile devices offers huge potential. It is very easy for people to "pick up and play" a game on their mobile device because they always have it in their pocket or bag. Whether users are progressing through a sprawling role-playing game while riding on a train or they simply want a few minutes of casual diversion while waiting for an appointment, they can turn to mobile gaming. With Windows 8 on the desktop, too, even greater numbers of users are available.

One thing that didn't make the jump from Windows Phone 7 to Windows Phone 8, and was never available at all for Windows 8 app development, is the popular XNA gaming framework. Fortunately, an existing open source project named MonoGame was able to step in to fill this void, continuing to offer a powerful and easy-to-learn game development framework that can be used by .NET developers, with no need to get involved with C++ or DirectX. MonoGame is almost entirely compatible with XNA, so existing knowledge and code can be transferred from XNA to MonoGame with a little effort.

This book aims to bring you the knowledge and techniques that you will need to create your own games for computers and devices running the Windows 8 and Windows Phone 8 operating systems. Starting with the basics of the platforms and their development environment and progressing to advanced topics such as 3-D graphics, it will guide you step-by-step toward creating a simple and manageable environment into which you can write your own games and distribute them to the world for fun or profit. Example projects are provided to demonstrate all of the techniques discussed, and they are ideal as a basis for experimentation.

Who This Book Is For

This book is written for those users who are already familiar with programming one of the two main managed Visual Studio languages, C# or Visual Basic.NET. It is assumed that you already have a grasp of the fundamentals of programming and are familiar with using the environment for PC-based application development. This is not an introduction to programming or to Visual Studio itself.

You will, however, be given a complete guide to setting up the development environment for Windows 8 and Windows Phone 8 application programming, to getting your first programs to compile, and to interactively debugging your games as they run within the various development environments available—on a local desktop PC, a tablet device, or the Windows Simulator for Windows 8 app development—and for Windows Phone development on either the Windows Phone emulator included with the phone's free software development kit, or on a real device.

From there, you will be shown in detail how to set up and use MonoGame for your own creations, covering 2-D and 3-D graphics, user input, sound effects and music, and much more.

In order to develop your games, you will need to use the Visual Studio 2012 development environment. If you already have Visual Studio 2012, you can add the required development tools into your existing environment. If you do not have it, you can download the various Visual Studio 2012 Express editions for both Windows 8 and Windows Phone 8 development free of charge from Microsoft's web site.

While most of the projects in the book can be developed using just the provided software testing tools, it is recommended that you also have access to real devices to test your games if possible—a phone for Windows Phone development or a Windows 8 tablet of some description for Windows 8 development.

The examples in this book are all written using C#. Developers who are more familiar with VB.NET should find that the language code and concepts translate over to C# fairly easily, so this should not present too much of a barrier to entry.

Chapter Overview

The following overview provides a brief description of each chapter. The chapters tend to build on one another, so it is recommended that you read them in sequence to avoid knowledge gaps in later chapters.

Chapter 1 introduces Windows 8 and Windows Phone 8 development and explores how to use the Visual Studio 2012 development environment to create games and applications for each one. This chapter also explains how to set up simple .NET projects running against the various testing environments and real devices, explores debugging techniques, and discusses the various options for code development across the two operating systems.

Chapter 2 dives into MonoGame, exploring in detail the structure of MonoGame projects, the approach to displaying and updating graphics, how sprites can be used to create complex 2-D graphics output, and how to work with fonts and text.

Chapter 3 takes the concepts explored so far and builds them into a simple reusable game framework that simplifies many of the tedious elements of setting up a game project. This allows you to focus on the game itself rather than getting weighed down with object management. This chapter also introduces an example game project named *Cosmic Rocks*, which will bring together many of the techniques covered.

Chapter 4 covers the subject of user input. All sorts of input devices are available on Windows 8 and Windows Phone 8 devices, from touch screens, keyboards, and mice to gamepads and accelerometers. All of these are explored in detail to show how they can be used to allow your games to be controlled.

Chapter 5 turns up the volume and reveals the options for game audio. Covering simple sound effects to MP3 music playback, this chapter gives you everything you need to know about sound for your games.

Chapter 6 begins to explore rendering with vertices and matrices instead of using sprites. Matrix transformations are uncovered and explained so that graphics can be rotated, scaled, and translated, and concepts such as texture mapping, blending, and alpha effects in this environment are explored.

Chapter 7 lifts the MonoGame feature set up into the third dimension, explaining how to create 3-D game worlds. Subjects covered include perspective and orthographic projections, the depth buffer, and lighting so that your scenes really come to life.

Chapter 8 continues the exploration of MonoGame in the third dimension and introduces a number of useful new rendering features. These include importing 3-D objects from third-party modeling packages, moving and manipulating the camera within a game world, creating particle effects, creating background imagery with sky boxes, applying fog to a 3-D scene, and using MonoGame's Effect objects to add new features and capabilities to your game.

Chapter 9 provides some useful reusable components that may be used in any game. A framework for managing multiple game modes to simplify switching between menu screens and game play sections, a simple mechanism for loading and saving user settings, and a high score table implementation are provided to allow you to focus on writing your game rather than having to reinvent these features yourself.

Chapter 10 exposes the application life cycle and provides techniques for handling the window size changing. These are essential topics that you will need to come to grips with so that your game can live side-by-side with other applications that the user chooses to open.

Chapter 11 takes a brief diversion away from MonoGame and begins to explore the XAML user interface features available to Windows 8 and Windows Phone 8. While not specifically geared around games, XAML has a lot of functionality that can be used in your game to simplify the task of creating user interfaces. This chapter introduces the environment and explores how it is used.

Chapter 12 takes a more detailed look at the controls that are available for use in XAML pages. It also explores topics such as XAML page layout options, page navigation, and device orientation.

Chapter 13 brings MonoGame and XAML together at last, demonstrating how both of these frameworks can be used in the same project, separately or both on screen at once.

Chapter 14 sets up shop inside the Windows Store and the Windows Phone Store. These are the outlets that you need to use to distribute your games to the rest of the world and perhaps make some money from them, too. The chapter contains a guide to the Store submission requirements as well as tips on testing your game, managing application versions, creating trial versions, and more.

CHAPTER 1

Getting Started

Developing games for Windows 8 and Windows Phone 8 can be a hugely enjoyable and rewarding way to spend your time. With a little effort and determination, you can create wonderful, enjoyable games that you can put in your pocket and spread to audiences around the world at the same time.

Microsoft's latest versions of its operating systems provide some rather different environments from the versions that came before them. In Windows 8, the design of the operating system has been adapted to suit both desktop and touch-screen operation. Windows Phone is radically different to Microsoft's earlier Windows Mobile platform. While these two systems look and function differently in a number of ways, they have begun to converge into a more consistent set of programming commands and libraries to make developing applications across the two a much easier task than in the past.

There is one key element of Windows 8 and Windows Phone 8 that has stayed essentially the same as the platforms that preceded them: the use of the .NET programming environment to create games and applications. This brings with it some exceedingly powerful and flexible programming languages and one of the best available development environments.

The development platform for Microsoft's mobile devices has advanced substantially over the last decade. During the early years of the original Windows Mobile/Pocket PC operating system, programming involved using the suite of eMbedded Visual tools. They came supporting two different languages: eMbedded Visual Basic and eMbedded Visual C++.

eMbedded Visual Basic was based on the same technologies as Visual Basic for Applications (VBA). It was similar in a number of ways to Visual Basic 6 (VB6), the desktop version of VB that was current at the time, but had many shortcomings such as the lack of strongly typed variables and poor object orientation features. Programs were written using a stand-alone integrated development environment (IDE), which had its own peculiarities and different ways of working from VB6.

eMbedded Visual C++ presented more of a challenge because of differences not only in the IDE but also in the code. Although established C++ programmers would no doubt have managed to pick up this language without too many problems, those who were less versed in the intricacies of C++ would have found that the amount of new information they needed to learn might be a significant barrier to entry.

All this changed with the release of Visual Studio .NET, and then later the versions of the .NET Framework that are built to target Windows 8 and Windows Phone 8 application development. These .NET versions provide class libraries that are parallel to the desktop .NET Framework. While not completely identical to their desktop equivalents, a substantial set of identical functionality does exist, and any programmer who is comfortable developing C# or VB .NET applications for Windows will be instantly at home developing for Windows 8 and Windows Phone.

Windows 8 and Windows Phone 8 development uses the very latest Visual Studio 2012. The IDE has made advances in a number of ways since that of the earlier versions of Visual Studio, but best of all, Microsoft has chosen to release "Express" versions of Visual Studio that support development completely free of charge. Although there are charges and fees involved in some areas of development and in distribution of finished applications (as we will see later in this book when we discuss this subject in more detail), these are generally fairly modest and do not create the barriers to entry that having to purchase the full versions of Visual Studio presented in the past.

The development environment also integrates into the full (Professional, Premium, or Ultimate) versions of Visual Studio seamlessly if you have such a version already installed.

On Windows Phone 8, all applications are delivered through the Windows Phone Store (previously known as the Marketplace). The Store is operated by Microsoft, and it provides a lot of very useful features. These features include an extremely simple installation mechanism with no need to worry about the installation of runtime frameworks, support for trial versions of applications, automatic update notifications, and protection against piracy.

Windows 8 also installs applications through its own Windows Store, which operates in an extremely similar way. The desktop is also available on Windows 8, however, and for computers running the “full” versions of the operating system (Windows 8, Windows 8 Pro, or Windows 8 Enterprise, but not Windows RT, the cut-down version of the operating system intended for tablet devices such as Microsoft’s Surface), it is possible to download and install applications without using the Windows Store. For the purposes of this book, we will only be considering Windows Store applications. These applications will work on all versions of Windows 8, including Windows RT.

A major advantage of developing using Visual Studio is that the exact same IDE is used as for any other Windows development you may undertake. There is no need to learn the details or keyboard shortcuts of a new IDE; instead, you will be working within the environment you are already used to, which includes all your user interface tweaks and preferences changes. Developing an application for Windows 8 or Windows Phone 8 is simply a question of creating a different project type.

Programming within Visual Studio also means that the Windows Phone developer can take advantage of the maturity of its development environment. Microsoft has spent many years improving the user interfaces and functionality of Visual Studio, and countless versions and releases have cumulated in an extremely powerful and user-friendly studio for application design, development, and debugging. All this is at your disposal when developing your games and applications.

The Framework also retains much of the power of its desktop cousin, including extensive object orientation features, strong variable typing, generics, flexible collections, and powerful XML processing functions.

In this chapter, we will take a closer look at the .NET Framework, at setting up and using Visual Studio, and at creating your first simple Windows Phone application. We will also examine some of the options that are available for game development.

A Closer Look at Visual Studio Development for Windows 8 and Windows Phone 8

Let’s start by taking a look at the versions of Visual Studio that we can use for developing software for these operating systems.

The free versions of Visual Studio are available as a number of “Express” installations. The two versions we are interested in are Visual Studio Express 2012 for Windows 8 and Visual Studio Express 2012 for Windows Phone. These can be installed individually or together, depending on your requirements.

Alternatively the “full” paid versions of Visual Studio 2012 can be used, in which case the Windows 8 and Windows Phone 8 development options will be integrated into your existing IDE.

Development does have system requirements that necessitate a reasonably modern PC. For Windows 8 development, your computer will need to be running Windows 8 Pro; for Windows Phone 8 development, you will need a 64-bit installation of Windows 8 Pro. Windows Phone development generally uses the Windows Phone Emulator a lot, too. In order for this to work, you will need to have a processor that supports hardware-assisted virtualization (this will need to be enabled in your PC’s BIOS settings), Second Level Address Translation (SLAT), and hardware-based Data Execution Prevention (DEP). Without these, you will only be able to test your applications by running them on a physical device. You can find full details about these hardware requirements and how to configure them on Microsoft’s web site at `http://tinyurl.com/wp8emulator`.

There is no support for developing for Windows 8 or Windows Phone 8 in earlier versions of Visual Studio. The good news is that Visual Studio 2012 will install side by side with earlier versions of Visual Studio without causing any problems.

Language Choices

.NET development offers the choice of a number of different languages. Three "managed" languages are supported: C#, VB, and C++. Both platforms also now support development with unmanaged C++ and also with JavaScript/HTML. These latter options provide a lot of new options in terms of cross-platform application development.

In this book, we will focus on C# for all of our samples and example code. If you are familiar with any of these other languages, you should find that both reading and writing C# will come naturally with a little practice.

IDE Features

As would be expected from Visual Studio, a number of very useful features are available to help develop and debug Windows Phone applications.

Simulators and Emulators

Visual Studio offers a number of options to help test and debug your programs on a variety of devices. Although it is strongly advised to use a real device regularly during your application development process to ensure that everything works properly on actual hardware, being able to use these test environments for general coding and testing is extremely useful.

For Windows 8 development, there are several options available. By default, your applications will simply run on your PC, just as they would if they were finished applications downloaded from the store. Most of the time, this is the way that you will execute them.

In order to know how your app will work on other devices, however, the simulator can be used as an alternative target for execution. The simulator runs in a window and pretends to be a separate device. This allows you to try out your application in a number of different resolutions and also to simulate both mouse and touch input. A screenshot of the Windows 8 simulator is shown in Figure 1-1.

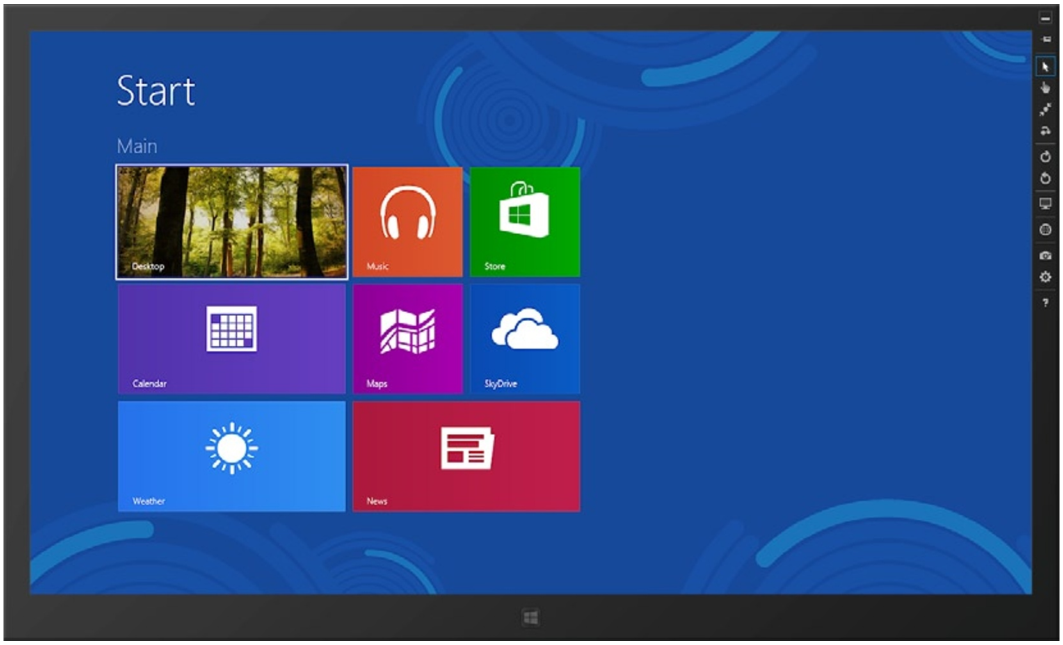


Figure 1-1. *Windows 8 simulator*

It should be noted that the simulator is actually running a virtual view of your own machine: any changes that you make inside the simulator will be made to your own PC as well, so be careful what you do!

For Windows Phone development, an emulator is provided inside which your applications can be executed. This emulator provides a full working implementation of a device, allowing you to test all aspects of integration between your app and other parts of the operating system. (This is a change from the emulator that was used for Windows Phone 7 development, which provided only a small subset of the functionality available on a real phone.)

The emulator offers access to a number of features of the device, including the ability to simulate networking, screen rotation, and touch-screen input using the mouse cursor. A screenshot of the emulator is shown in Figure 1-2.

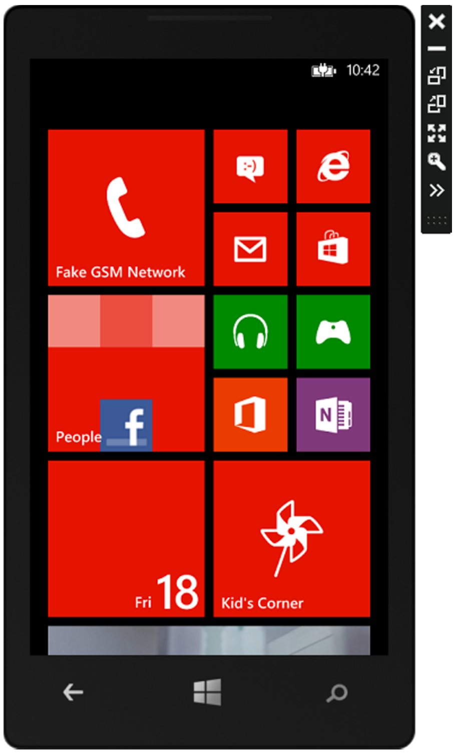


Figure 1-2. *Windows Phone 8 emulator*

Whichever platform you are developing for, switching between these different deployment options is as simple as can be. All you need to do is select to the target environment in the Visual Studio toolbar and start your application—it will launch in the appropriate environment, ready for you to use.

When we fire up the simulator and emulator shortly, you'll see that it takes a few seconds for each to initialize itself. This delay can be frustrating when you are in a repeat modify/compile/test cycle, but both the simulator and emulator can be left running in the background when you stop your code from executing. They will then resume much more quickly the next time you begin a debug session.

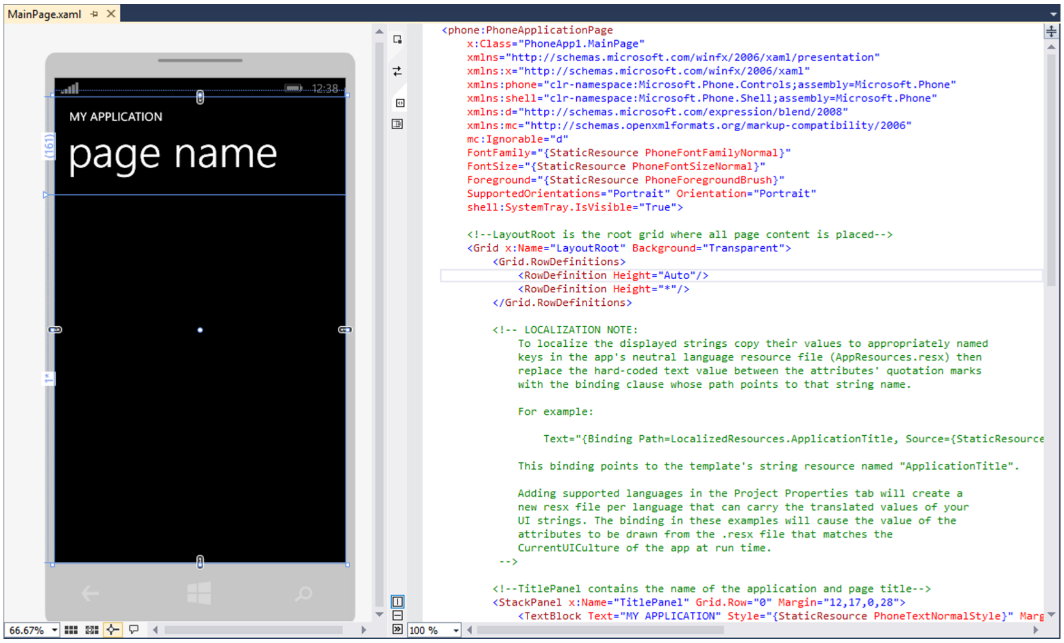
Also note that the Windows Phone emulator does not retain its state when it is closed. Each time you restart it, all previous data will have been cleared and the emulated device will be reset to its default settings. In order to retain stored data from one execution to the next, it is important not to close the emulator window.

XAML Page Designer

A fully featured page designer is available to lay out windows and controls for use within your applications. The designer goes as far as to display an image of the device around the edge of your page to help visualize its appearance.

These pages are created using an XML-based markup language called XAML (pronounced "zammal"), which is an abbreviation for eXtensible Application Markup Language. Pages may be created and modified by using the visual designer or by editing the underlying XAML directly.

Visual Studio will display both the designer and the XAML editor as side-by-side panels within the IDE, as shown in Figure 1-3, and any change made to either will be immediately reflected in the corresponding panel. This provides a very flexible mechanism for page design, allowing each panel to work together to perform its actions more efficiently.



```
<phone:PhoneApplicationPage
    x:Class="PhoneApp1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

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

        <!-- LOCALIZATION NOTE:
            To localize the displayed strings copy their values to appropriately named
            keys in the app's neutral language resource file (AppResources.resx) then
            replace the hard-coded text value between the attributes' quotation marks
            with the binding clause whose path points to that string name.

            For example:

                Text="{Binding Path=LocalizedResources.ApplicationTitle, Source={StaticResource

            This binding points to the template's string resource named "ApplicationTitle".

            Adding supported languages in the Project Properties tab will create a
            new resx file per language that can carry the translated values of your
            UI strings. The binding in these examples will cause the value of the
            attributes to be drawn from the .resx file that matches the
            CurrentUICulture of the app at run time.
         -->

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock Text="MY APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Marg
```

Figure 1-3. *The XAML page editor showing the designer and page source together for a Windows Phone app*

We won't be using XAML for actual game development in this book, but later on we will look in much more detail at how it can be used alongside other parts of our games in order to provide easy and functional user interfaces.

Breakpoints

Another extremely useful tool is Visual Studio's breakpoint feature. No doubt familiar to any desktop developer, breakpoints are fully supported for Windows 8 and Windows Phone development, and they can be used when running against the simulator, emulator, and a physical device. It can be extremely useful to break into your code, examine your variables, and step through the instructions while watching the results on a real device on the desk next to you.

Debug Output

Access to the Visual Studio Output window is available from applications running inside the IDE. Text can be written to the Output window at any time, allowing you to easily keep track of what your program is doing. Pairing this with the ability to have two independent screens (your PC screen and your device screen) makes this tool particularly powerful.

Windows 8 Devices

Windows 8 itself runs on a very diverse set of devices: desktop PCs, laptops with or without touch screens, and tablet devices. Some devices may be running the cut-down Windows RT, others the full versions of Windows 8. In order to maximize your potential audience, your applications will need to tolerate all of these environments and work seamlessly regardless of what conditions they find themselves running in.

Let's take a look at what we can expect.

Screen Hardware

PCs come with screens of all shapes, sizes, and resolutions. At the time of this writing, according to the "Steam Hardware and Software Survey," which is performed by the game company Valve, the most common desktop PC display resolution is 1920-by-1080 pixels (which is standard "full HD" resolution), used by 28 percent of users. While this may be a good default resolution to target, it does mean that 72 percent of users are not going to be using that resolution. (If you'd like to see up-to-date statistics on this and other areas of hardware, `http://store.steampowered.com/hwsurvey` contains the full survey results.)

When it comes to tablet devices, things are different once again. While the Surface Pro does use a 1920-by-1080 screen, the cut-down Surface RT's resolution is only 1366-by-768 pixels. Other tablets from other manufacturers could fall anywhere on this scale, and let's not forget that tablet devices can easily be switched between landscape and portrait orientations.

Clearly, knowing the resolution to use for your game is going to be a challenge!

Fortunately, there are some features of the game environment that we will be using that will come to our rescue here. We'll take a look at these later on. The important thing to bear in mind is that you have no idea what screen resolution you will be operating in, so you will need to test in a variety of resolutions inside the Windows 8 simulator.

Hardware Buttons

Once again, the identity of the physical buttons your users will have connected is going to be a complete unknown. At one end of the scale is the desktop PC, replete with keyboards and multibutton mice. At the other end is the tablet device, which has just one single button—the Windows button.

In order for your game to work across all these devices, you will have to cater for the simplest set of hardware buttons available, which is essentially none at all. (The Windows button cannot be intercepted by your application.) There are standard ways to deal with this that we will look at later on.

It should be noted, however, that if you can provide additional functionality for users who are using a desktop PC, they will more than likely appreciate it! Little touches such as supporting the Escape key to go backward can make all the difference to your users' experience.

Graphics Hardware

The range of graphics cards and their capabilities is, of course, huge. In terms of desktop PCs, all modern graphics cards are astoundingly powerful and will more than likely cope with whatever challenges you throw at them. Laptops and, in particular, tablet devices are likely to be less capable in this area.

In order to avoid any nasty surprises, try to test your games on lower-powered devices such as tablets as frequently as possible. If you find that you are pushing past the capabilities of the hardware, it's important to realize this as soon as possible to prevent significant reworking later on.

The simulator will not provide a realistic representation of performance on other devices as its performance will be based on that of your development PC, so there is no substitute for real devices in this instance.

Location and Orientation

Some devices will offer sensors to tell you which way up the device is being held and the location in the world where it is being used (using GPS or even via detection of local WiFi connections if GPS is not available, albeit with less accuracy). In certain environments, these are great for games. Orientation sensors can be used to implement steering wheels for driving games, for example, where tilting the device steers the car. Location sensors offer possibilities for “accentuated reality” games, where the real world and the game mix together to provide an additional layer of realism.

These sensors are unlikely to be available in anything other than tablet devices, however (tilting your desktop PC monitor is probably not going to catch on!), so if you decide to take advantage of one of these sensors, consider your alternatives for when they are not available.

Cooperation with the Device

Unlike desktop applications, Windows Store apps have a stricter set of requirements when it comes to their interactions with other parts of the device. They have limited access to the file system, can only use the managed .NET APIs, and have highly constrained multitasking capabilities.

Generally, when your app loses focus, Windows will allow it to keep running in the background for a short time. After a while, however, the app will be put to sleep and will no longer execute any code. It can be resumed at any time the user wishes, but it cannot perform updates while in this state.

For games, this does not often create too many problems, but do be aware that you have less power and control over the device than you may be used to on other operating systems.

Windows Phone Devices

One of the major changes that Microsoft made when first introducing Windows Phone, as compared with the previous Windows Mobile platform, concerns hardware requirements.

The huge diversity of hardware could provide quite a barrier for entry for Windows Mobile. Devices all had different screen resolutions, different hardware buttons, and a fairly substantial range of other internal hardware. Writing games and applications that worked across the entire set of devices could result in a considerable amount of additional effort, whereas saving time by not addressing all these platforms could exclude a significant proportion of the potential customer base from being able to use the application.

This was tackled head on in Windows Phone 7 by requiring a very rigid set of hardware requirements. As the operating system and technologies evolved, this started to become overly restrictive. As a result, with Windows Phone 8, these restrictions have been relaxed slightly. This does create some overheads that you will need to be aware of, but it still provides a solid and predictable platform to develop against.

Let’s take a look at what we can expect.

Screen Hardware

To begin with, all Windows Phone 7 devices had exactly the same screen resolution: Wide VGA (WVGA) resolution (480 pixels across-by-800 pixels tall). This greatly simplified the task of ensuring that games and applications properly fit on the screen without having to stretch, shrink, or leave large areas of the screen unused.

As technology and competing devices marched forward, Windows Phone had to ensure that it kept up pace, so this is now just the minimum resolution. A number of devices currently use higher resolution screens (for example, the HTC 8x has a 720-by-1280 pixel screen, and the Nokia Lumia 920 uses a resolution of 768 by 1280). This results in some superb-looking devices, but it also means that you, the developer, will have to cater for all the possible displays that your game will need to run on. Fortunately, the emulator is able to run at all these resolutions, so you can see how your app responds without needing a fleet of different phones.

All Windows Phone devices have multitouch capacitive touch screens, with a minimum of four distinct points able to be tracked at once. This opens up some interesting possibilities for gaming, and we’ll be looking at how to use multitouch in games later on in this book.

Just as with tablet devices running Windows 8, an important consideration when designing a game for the phone is that the screen orientation can be rotated into portrait or landscape orientation. Fortunately, Windows Phone has extremely good support for rotating between these orientations, so you can take advantage of whichever screen layout best suits your game.

Hardware Buttons

One of the details that Microsoft has been very strict about for devices running its new operating system is hardware buttons. All devices must have exactly three buttons on the front of the device: a Back button, a Windows button, and a Search button. Of these, only the Back button can actually be used by your application, and you must ensure that it is used for predictable navigation and nothing else. This is, of course, one button more than we can rely on when developing for Windows 8, so we can try to view this as a positive!

Having consistency over the available buttons is good for developers as it means that we don't have to worry about lots of combinations of control mechanisms. However, this limited set of buttons means that there will be no directional pad available, which is a pity because they are very useful as a game input device.

Instead, we can use the touch screen for input, and there are lots of clever and creative ways that this can be done from designing games that the user interacts with by touching objects on the screen, through virtual input pads that spring up wherever the user touches the display, to displaying movement buttons at the bottom of the screen for the user to press.

These rigid requirements don't rule out the possibility of a device manufacturer including a hardware keyboard with the device. A number of Windows Phone 7 devices featured such keyboards, though they don't seem to be available in the Windows Phone 8 world yet. The presence of a keyboard opens up the opportunities for the player to control your game, but clearly we need to avoid making this a necessity for your game so that it works on those devices that do not have one available.

Processors

The Windows Phone platform may not be at the absolute cutting edge when it comes to processors, but it is now commonly using dual-core processors in order to improve processing capability. The operating system has been tuned very well to run in the mobile environment, however, so it often outperforms more powerful competing devices in general use. For serious number crunching though, it may not be able to compete.

If your application is processor-intensive and you are testing it on a high-end device, it would be a good idea to find someone with a less powerful phone and get that person to verify that the performance is acceptable.

Graphics Hardware

All Windows Phone 8 devices contain dedicated graphics hardware and take advantage of this to provide hardware-accelerated 3-D graphics. This is extremely important for game development, of course, and it is one of the factors that make the platform viable for modern games.

The performance of the hardware is impressive considering its size, but there is no real comparison between it and the current top-end desktop graphics hardware. Be realistic about what you are going to ask it to do—though don't be afraid to experiment and see how far you can push it!

Location and Orientation

Also standard on all devices will be an accelerometer and a Global Positioning System (GPS) receiver.

The accelerometer can be very useful for game developers. It allows the device to detect which way up it is being held and can sense in detail any movement that results in the device being rotated. This provides an excellent input control mechanism, allowing players to influence what is happening on the screen by physically moving their phones.

Probably of less interest for gaming is the GPS functionality. When appropriate line-of-sight reception has been established with the GPS satellites, the device can detect where in the world it is located. This opens opportunities for making games that revolve around the player's whereabouts, but the scope for this in gaming may be limited.

Cooperation with the Device

Let's not forget an extremely important fact: your game is running on other people's phones and personal organizers. They will place more importance on tasks such as answering a phone call or responding to a calendar reminder alert than in continuing to play your game.

Running applications have limited control over what happens when other features of the device become active. An application that loses focus will be immediately placed into a suspended state, and no further processing is possible until it is reopened by the user.

In the original release of Windows Phone 7, games were terminated entirely when they lost focus and could only be restarted from scratch. If they wanted to maintain state so that the user could resume the game, they had to take care of handling the game state themselves. This could be a time-consuming and frustrating programming task.

Fortunately, this was hugely improved with a subsequent operating system update. This updated behavior is present in Windows Phone 8, too. Windows Phone is now able to automatically restore any recent app to exactly the state it was left in. Your app is notified when this happens; however, it can be useful to take advantage of this by, for example, automatically pausing when the game resumes to allow the user time to adjust to what is going on. People will appreciate small details such as these.

Using Visual Studio for Windows 8 and Windows Phone Development

Let's take a look now at the steps required to begin development of games and applications.

Installing Visual Studio

Installing Visual Studio for Windows Phone development is very easy. If you do not already have a full version of Visual Studio installed, you can visit `www.microsoft.com/visualstudio/eng/products` to download the Visual Studio 2012 Express editions. These are free, complete, and fully functional development environments that will provide all the tools needed for you to develop your games.

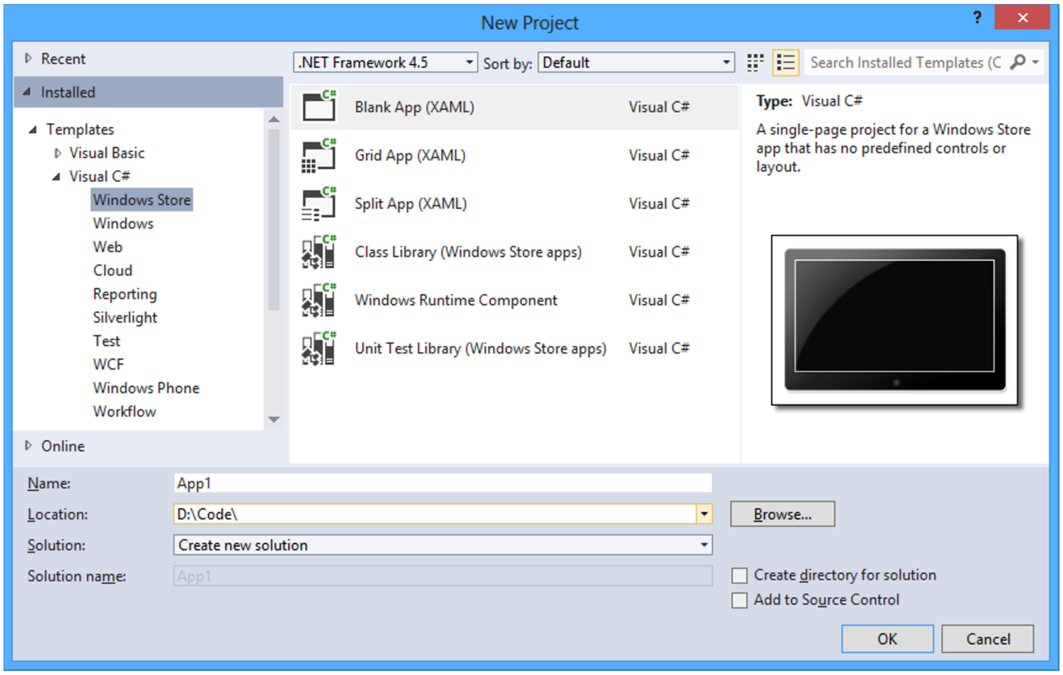
You will find editions for both Windows 8 and Windows Phone, so download whichever of these you need. For reasons we will cover later, however, it is important that the Windows Phone development environment is available, even if you are planning to develop only for Windows 8. Make sure that you do install that as well. The Express editions install as two separate applications, each of which enables you to target just the appropriate types of project.

When the Express versions of the product are first launched, they will prompt you to enter a registration product key. It is possible to continue without such a key for a period of 30 days, but after this, Visual Studio will stop working until such a key is entered. The keys can be obtained free of charge by clicking the Register online button and following the instructions that appear. You will need a Microsoft Live ID in order to complete this, and the registration process will guide you through creating one if necessary. Your Live ID will later be used for publishing your finished apps, so make sure you look after your account details carefully.

Following this, you may be prompted to obtain a developer license. This is also free and the process takes just a few seconds to complete. The developer license is valid for one year, after which time it will need to be renewed—Visual Studio will prompt you automatically to repeat these steps when needed.

If you already have a full version of Visual Studio 2012 installed (one of the Professional, Premium, or Ultimate editions), support for building Windows 8 apps (known as "Windows Store" applications) will already be available. In order to add support for Windows Phone development, use the same installer for Visual Studio Express for Windows Phone as detailed at the start of this section. The setup application will detect the presence of Visual Studio when it is launched, and it will download just the components that are required, based on your current system configuration.

Once the installation is complete, Visual Studio can be launched and the project types for Windows Store and Windows Phone projects will appear. Figure 1-4 shows the New Project window ready to create a new, empty Windows Store application for Windows 8. Figure 1-5 shows the same window preparing to create a new Windows Phone project.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Store
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (C
Blank App (XAML) Visual C#
Grid App (XAML) Visual C#
Split App (XAML) Visual C#
Class Library (Windows Store apps) Visual C#
Windows Runtime Component Visual C#
Unit Test Library (Windows Store apps) Visual C#
Create directory for solution
Add to Source Control
Browse...
OK
Cancel

Type: Visual C#

A single-page project for a Windows Store app that has no predefined controls or layout.

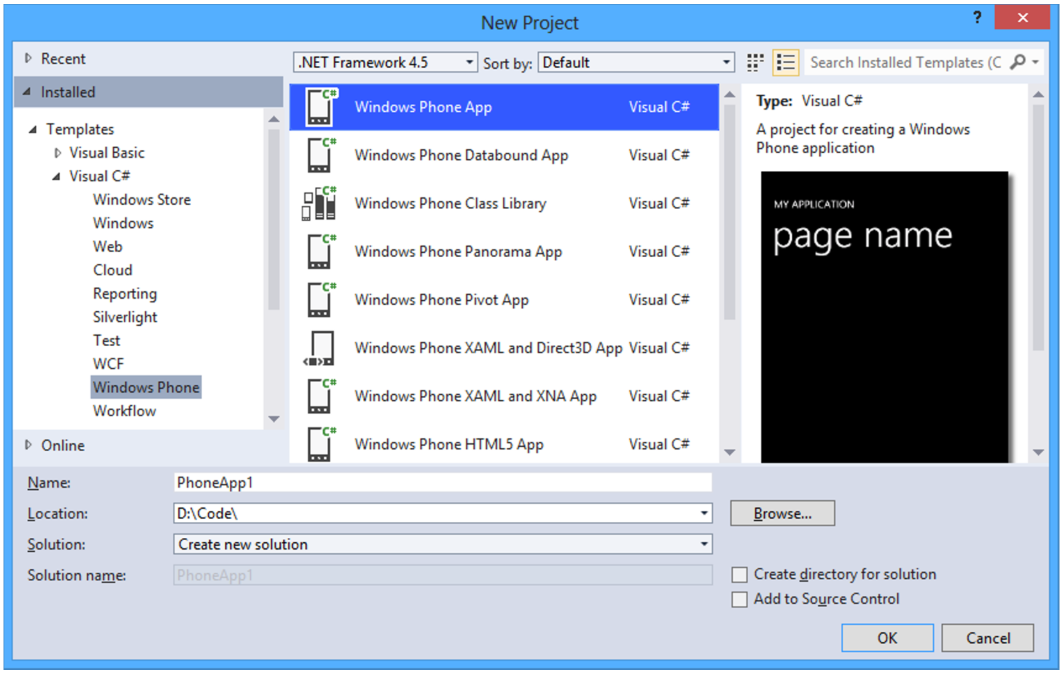
Name: App1

Location: D:\Code\

Solution: Create new solution

Solution name: App1

Figure 1-4. *Creating a new Windows Store project in Visual Studio 2012*


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Store
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (C
Windows Phone App Visual C#
Windows Phone Databound App Visual C#
Windows Phone Class Library Visual C#
Windows Phone Panorama App Visual C#
Windows Phone Pivot App Visual C#
Windows Phone XAML and Direct3D App Visual C#
Windows Phone XAML and XNA App Visual C#
Windows Phone HTML5 App Visual C#
MY APPLICATION
page name
Create directory for solution
Add to Source Control
Browse...
OK
Cancel

Type: Visual C#

A project for creating a Windows Phone application

Name: PhoneApp1

Location: D:\Code\

Solution: Create new solution

Solution name: PhoneApp1

Figure 1-5. *Creating a new Windows Phone project in Visual Studio 2012*

■ **Tip** Visual Studio 2012 ships with two visual “themes” named Light and Dark, and it defaults to using the Dark theme. If you wish to switch between these two themes, you can do so via the Tools ➤ Options menu. The Color theme is the first item in the Environment ➤ General section of the Options window.

Once everything is installed as required, we are ready to begin. The following sections will guide you through creating your first project for Windows 8 and for Windows Phone. They will both follow essentially the same steps and end up with the same test project. Feel free to work through either or both of them, as needed. Much of the content is repeated for these sections so that they can each be read in isolation.

Creating a Windows Store Project

With tools all in place, it is time to create a simple Windows Store application for Windows 8 and take a look at how we can execute it. We will create a simple blank XAML application for this purpose.

To begin, select File ➤ New ➤ Project within Visual Studio, and choose the `Visual C#` item within the list of Installed Templates, followed by the `Windows Store` item. The panel to the right will display the available Windows Store templates that can be used to create a new project. Select the `Blank App (XAML)` item.

If you are using one of the full versions of Visual Studio, you will see that above the templates is a drop-down list that allows the .NET Framework version to be selected. For Windows Store development, we will always leave this set to .NET Framework 4.5. The Express editions of Visual Studio automatically select .NET 4.5.

At the bottom of the window, select a project directory and enter a project name (or accept the default name of `App1` if you wish). Unless you have a particular preference for using separate directories for the solution and project files, uncheck the `Create directory for solution` check box to keep the directory structure a little tidier.

Once everything is ready, click the OK button to create the project. After a few seconds, the new project will open within the Visual Studio IDE.

Project Templates

A number of different templates are provided by Visual Studio for your new project, one of which we selected in the previous section. Each of these will result in a different initial project for you to start working on. The templates that you will most likely find useful are as follows:

- *Blank App (XAML)*: This is the default empty project template for creating XAML applications. If you are creating apps other than games, this is often the place you will begin. We will actually use a custom template for our games, which we will see in Chapter 2.
- *Grid App (XAML)*: This template creates another near-empty XAML project, but with some predefined user interface pages revolving around the XAML Grid control.
- *Split App (XAML)*: This is another near-empty XAML project template, this time with a different preset user interface design.
- *Class Library (Windows Store apps)*: If you need to create any separate Class Library projects (to build DLL files) for your Windows Store apps, use this project template to create them. The resulting project is essentially completely empty, but it is compatible with Windows Store projects and so ready for you to add your classes to.

It is not possible to change the project type once the project has been created. If you find that you need to change the project once you have started developing it, you will need to create a new project of the required type and copy in all of the existing code files.

Designing a Page

Now, we are ready to make some minor changes to your test application's default page. The page is named `MainPage.xaml` inside Solution Explorer, and you can double-click it to open it in the XAML page editor.

In the top section of the page editor window is the page designer and preview area. In the bottom half is the XAML code for the page. If you want, you can change the orientation of the divide by clicking the Horizontal Split or Vertical Split buttons at the bottom or right of the dividing bar between the two. The two parts can also be quickly swapped over by clicking the Swap Panes button, also located inside the divider. The designer panel also contains a useful zoom slider that allows you to focus more closely on the page design for detailed layout work, if required.

Initially, the page is a simple, empty, black page. For the purposes of this simple application, we will simply place a Button control onto the page and get it to display a message when clicked. The Button is added from the Toolbox exactly as it would be for a desktop application: click the Button icon in the Toolbox panel (expand the `Common XAML Controls` item if needed) and then draw it into the empty region in the center area of the page. The result can be seen in Figure 1-6.

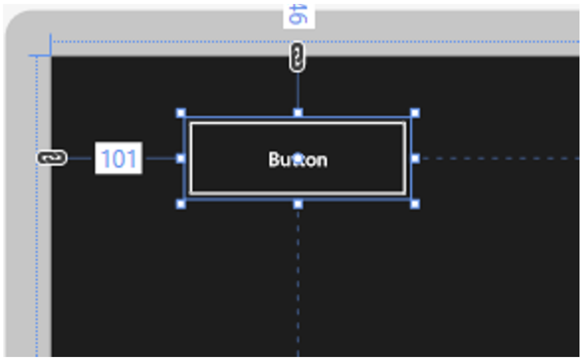
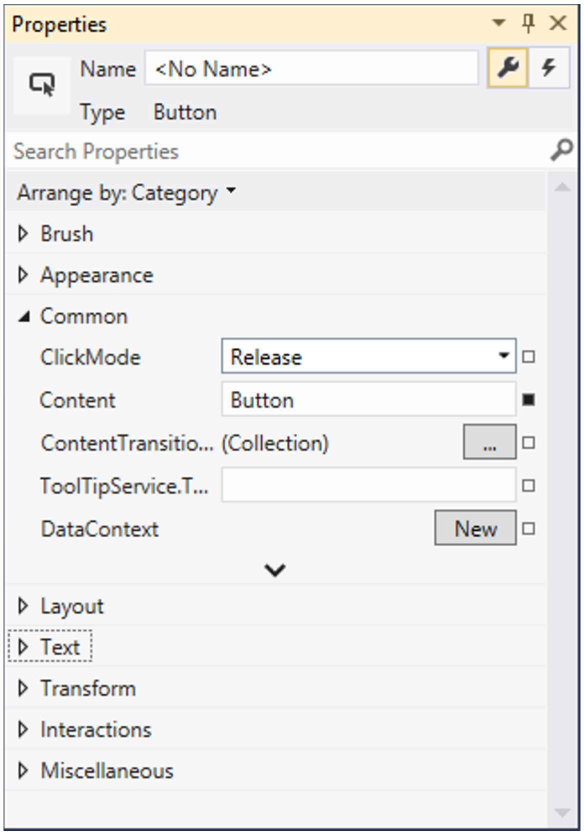


Figure 1-6. *A button placed within the Windows Store app XAML page designer*

Once you have added your Button, take a look at its properties in the Properties window (see Figure 1-7). If you have created Silverlight applications in earlier versions of Visual Studio, you will probably recognize many of the available properties. If you have worked only with Windows Forms in the past, many of these properties might be unfamiliar, but there should also be some whose purpose is obvious. We'll look into more of these properties in greater detail in the XAML chapters later in this book.



Name <No Name>
Type Button

ClickMode Release
Content Button
ContentTransitio... (Collection)
ToolTipService.T...
DataContext New

Figure 1-7. *The Button's properties*

■ **Tip** If the Properties window is not open, it can be opened by selecting the `View/Properties Window` item from Visual Studio's main menus. Under the default key mappings, it can also be opened by pressing F4 on the keyboard.

Double-click the button to open the code designer and create the button's `Click` event handler. This will display the "code-behind" file for the page, which is where the C# code is developed. In Solution Explorer, you will see that the `MainPage.xaml` item has expanded to reveal `MainPage.xaml.cs` contained within. These two files are analogous to the form design and form code files that are present when working with Windows Forms.

The code within `MainPage.xaml.cs` should look just as you would expect when developing a desktop application. At this stage, it should be clear how much similarity there is within the Visual Studio IDE between desktop and Windows Store application development.

Complete the implementation of the `Button_Click_1` procedure, as shown in Listing 1-1.

Listing 1-1. The Button_Click_1 procedure for the Windows Store app

```
private async void Button_Click_1(object sender, RoutedEventArgs e)
{
    MessageDialog msg = new MessageDialog("Welcome to your app!","Message");
    await msg.ShowAsync();
}
```

You may receive some compilation warnings after entering this. If you do, try checking the following:

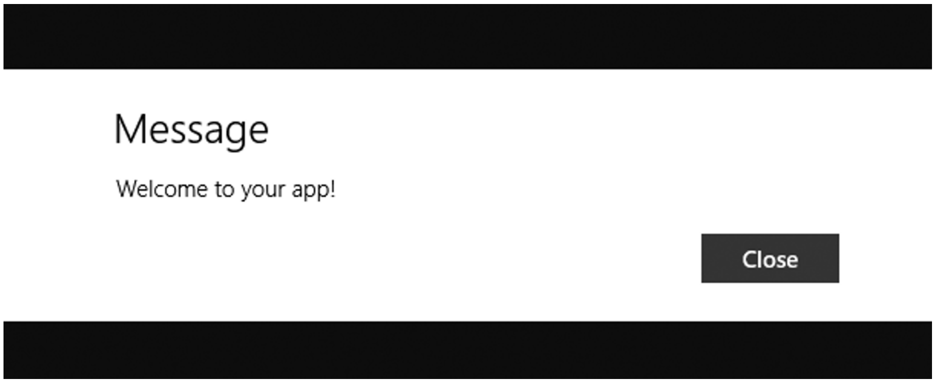
- Note the `async` keyword that was added to the procedure declaration line. You will need to add this manually. This should resolve one of the errors.
- The `MessageDialog` class resides in a .NET namespace that we have not instructed the compiler to automatically use. To fix this, right-click the `MessageDialog` class name within your source code and then choose Resolve ➤ using Windows.UI.Popups; from the menu that appears. This will add the required `using` statement to the top of the source file and correct the other errors.

■ **Note** You may be unfamiliar with two of the keywords used here: `async` and `await`. These are both new to .NET 4.5 and are designed to simplify your code when operating time-consuming tasks that might otherwise cause the user interface to become unresponsive. We will look at these keywords in more detail later in the book.

Running the Application

We are now ready to compile and run the project. Press F5 or click the Run button in the Visual Studio toolbar to begin the process. After compilation (and assuming that there are no errors!), Visual Studio launches the application directly on your PC. As with all Windows Store applications, this will entirely take over the screen—this may be something of a surprise at first, but you will quickly get used to it!

Unsurprisingly, the app displays a black screen containing a button. Clicking the button will display the message dialog box shown in Figure 1-8.



Message

Welcome to your app!

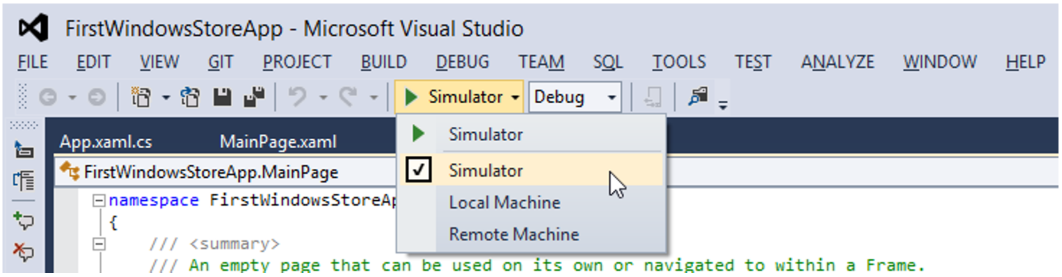
Figure 1-8. *The test application in action*

To stop the program, Alt-Tab back to the Desktop (or click the Desktop in the running apps panel by moving the mouse cursor to the very top-left of the screen) and click the Stop Debugging button in Visual Studio. The IDE will return to edit mode, and your program will close.

Running in the Simulator

Most of the time, the quickest and most efficient way to test and develop your app will be running it directly on your PC, but when you want to be able to see how it behaves on different devices (specifically, devices with different screen resolutions), the simulator is an excellent and straightforward way to achieve this.

To switch to using the simulator, click the drop-arrow on the Debug Target button on the toolbar and select Simulator, as shown in Figure 1-9. Then begin running your project as normal.


FirstWindowsStoreApp - Microsoft Visual Studio
FILE EDIT VIEW GIT PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Simulator
Debug
App.xaml.cs
MainPage.xaml
FirstWindowsStoreApp.MainPage
Simulator
Simulator
Local Machine
Remote Machine

```
namespace FirstWindowsStoreAp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
```

Figure 1-9. *Selecting to run the project against the simulator*

This time, instead of launching on your PC and taking over the whole display, the simulator application will launch on your desktop and the project will run inside it. You can then switch back and forth between the simulator and Visual Studio as much as you need to in order to debug and improve your project.

■ **Caution** The simulator is actually a window into your own PC configuration (it is internally implemented as a specialized Remote Desktop session). Any changes that you make to the setup of the computer inside the simulator will also be made to your own PC, so be sure you know what you are doing before changing things!

On the right edge of the simulator window, a number of tools are displayed to help control the way the simulator works. These include a number of "input mode" buttons to control whether the simulator is simulating mouse or touch input. Among these are "pinch" and "rotate" modes, which simulate using two fingers at once. These can be operated by holding the left mouse button and rolling the mouse wheel. Clearly, this is not as intuitive and simple to use as an actual touch screen, but it is much better than nothing!

Another button allows the device resolution to be controlled, as shown in Figure 1-10. These options will immediately switch the simulator to the appropriate screen resolution without needing any kind of restart. This can sometimes prove quite confusing to your app, which may not expect such a change to be possible. If unusual behavior occurs following a resolution change, it is usually advisable to restart any running app

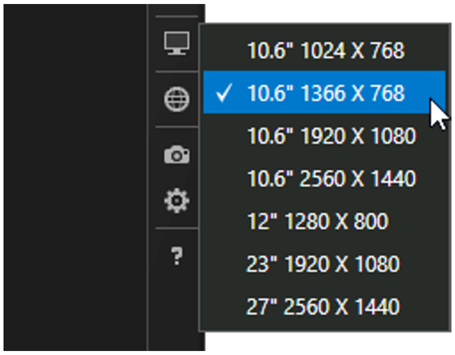


Figure 1-10. *Available simulator screen sizes and resolutions*

Some of these resolutions will be inconvenient or impossible to display on your PC monitor, so the simulated screen is automatically scaled to fit the size of the simulator window. On the larger resolutions, this will probably result in a loss of detail as the image will be shrunk to fit, so be aware that you may not be seeing the full display detail in these situations.

Running on an External Device

You will no doubt be pleased to hear that running your application on a separate device is no more difficult than running it on your PC or within the simulator, and it provides an extremely similar experience. This procedure works equally well for portable devices (tablets) and other desktop PCs. There are a few steps that you will need to go through before you can begin working with the device, however. Let's take a look at the details now.

Connecting the Device

First of all, you will need to ensure that the device you are going to work with is connected to the same network as your development PC. All remote-device interaction is carried out across the network rather than by connecting devices physically (with USB cables, for example).

If your target device is a portable device, ensure that its WiFi is enabled and connected to the appropriate network. If the device is another PC, ensure that it is part of the same Local Area Network as your development PC.

Installing the Remote Debugger

In order for your project to be able to launch on another device, that device must be running the Visual Studio Remote Debugger. This is a small, free application released alongside Visual Studio that you can install on the target device.

The installer for the Remote Debugger can be found on the Visual Studio download page: `www.microsoft.com/visualstudio/downloads—search` the page for "Remote Tools for Visual Studio 2012." There are three different versions of the installer available, one for each supported processor architecture (x86, x64, or ARM). If you wish to install this on a Windows RT device such as the Surface, you will need to launch the Desktop and then download to the device's disk in exactly the same way as you would on a normal desktop PC, then double-tap the downloaded file to begin the installation.

Once the debugger has been installed, you will find a new Remote Debugger icon on your Start menu. Select this to launch the debugger. The first time it launches, the debugger will display an Options window, as shown in Figure 1-11. These options can be left at their default settings. After clicking OK, the Remote Debugging Monitor will appear, showing content similar to that in Figure 1-12. When this appears, the device is ready to communicate with Visual Studio.

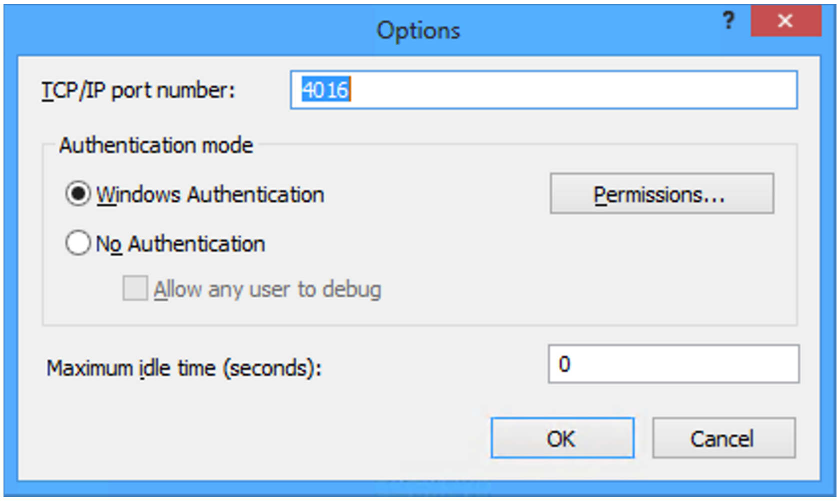
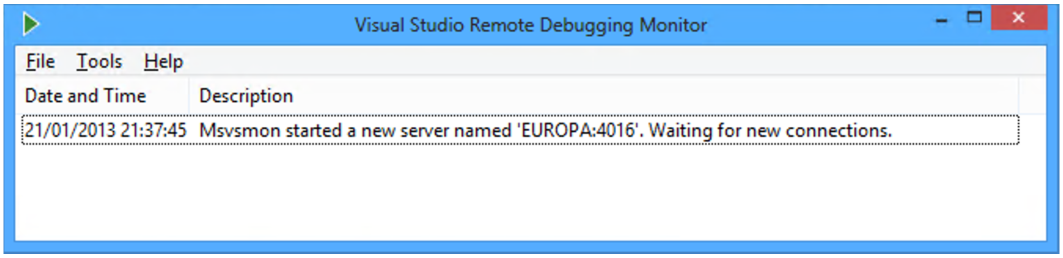


Figure 1-11. *The Remote Debugger Options window*



Date and Time	Description
21/01/2013 21:37:45	Msvsmon started a new server named 'EUROPA:4016'. Waiting for new connections.

Figure 1-12. *The Remote Debugger Monitor window waiting for a connection from Visual Studio*

Deploying to the Device from Visual Studio

Now we are ready to deploy our project to the device from inside Visual Studio.

First, click the drop-arrow on the Debug Target button on the toolbar and select Remote Matching from the available targets. Visual Studio will display the Remote Debugger Connections window, as shown in Figure 1-13. Your configured remote device should appear in the device list (as “Europa” does in this illustration). Click the required device and then the Select button that appears next to it in order to specify the device against which to debug.

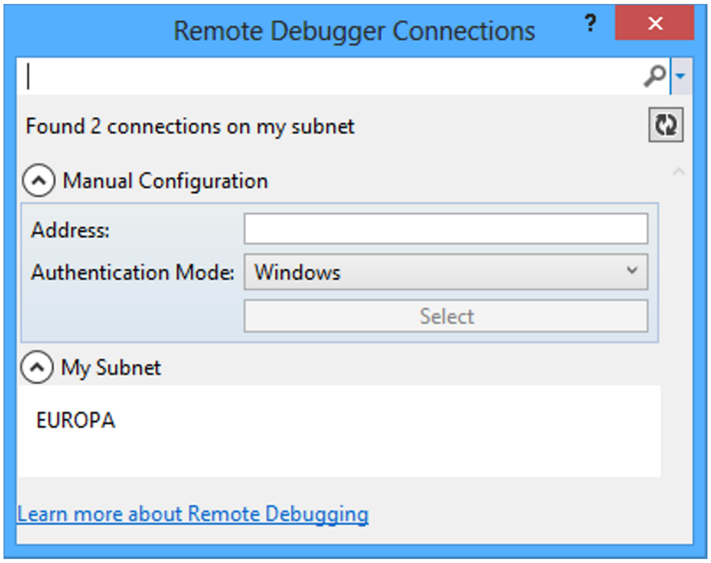

Remote Debugger Connections
?
×
Found 2 connections on my subnet
Manual Configuration
Address:
Authentication Mode: Windows
Select
My Subnet
EUROPA
Learn more about Remote Debugging

Figure 1-13. *Selecting a remote device to use for debugging*

■ **Tip** Once you have selected a device, Visual Studio will save this selection as part of your Project configuration. If you wish to change to a different device later on, open the Project Properties window and select the Debug tab. Within the Start Options section, you will be able to select the appropriate target device within the Remote machine field.

At last, everything is configured and ready to run—start your project running and it will connect to and deploy your app to the target device. Well, almost. The very first time, you will need to obtain a developer license for the target machine. Just as with the license for your development machine, this is free and takes just a few seconds to obtain by following the prompts that appear to complete this step. Once this is done, your app should finally launch.

■ **Note** For deployment to succeed, the device must be logged in, the screen must be on, and the lock screen must not be displayed. If these conditions are not met, a deployment error will occur, although Visual Studio usually provides good feedback about what the problem is to help you figure out how to proceed.

This sounds like quite a lot of work, but once the initial configuration has been completed, it's very easy to repeat later on: simply launch the Remote Debugger on the target machine, select Remote Machine as your debug target, and launch the project.

For tablet devices in particular, this is a delightful way to work: the tablet need not be tied down to your development machine with physical cables—it can simply sit on the desk next to you.

Apps that are deployed to target devices in this way will continue to be available even once the development machine and the Remote Debugger have been closed, so you can continue to use them just like any other app that you have installed.

Congratulations—you have written and deployed your first Windows Store application!

Creating a Windows Phone Project

Now it is time to create a simple Windows Phone application and take a look at how we interact with both the emulators and real devices. We will create a simple XAML project for this purpose.

To begin, select File ➤ New ➤ Project within Visual Studio, and choose the `Visual C#` item within the list of Installed Templates, followed by the `Windows Phone` item. The panel to the right will display the available Windows Phone templates that can be used to create a new project. Select the `Windows Phone App` item.

If you are using one of the full versions of Visual Studio, you will see that above the templates is a drop-down list that allows the .NET Framework version to be selected. For Windows Phone development, we will always leave this set to .NET Framework 4.5. The Express editions of Visual Studio automatically select .NET 4.5.

At the bottom of the window, select a project directory and enter a project name (or accept the default name of `PhoneApp1` if you wish). Unless you have a particular preference for using separate directories for the solution and project files, uncheck the Create directory for solution check box to keep the directory structure a little tidier.

Once everything is ready, click the OK button to create the project. Another dialog box will appear prompting you to choose the target Windows Phone OS version: select Windows Phone OS 8.0 from the available options and click OK.

After a few seconds, the new project will open within the Visual Studio IDE.

Project Templates

Visual Studio provides a number of different templates for your new project. Each of these templates will result in a different initial project for you to start working on. The templates that you may find the most useful might include the following:

- *Windows Phone App*: This is the main project template for creating empty XAML applications. If you are creating apps other than games, this is often the place you will begin. We will actually use a custom template for our games, which we will see in Chapter 2.
- *Windows Phone DataBound/Panorama/Pivot App*: Each of these creates another near-empty Silverlight project, but a particular predefined user interface example is in place.
- *Windows Phone Class Library*: In order to create a class library that can be used by other Windows Phone projects, use this template. Such class libraries will be accessible from the games that we will be creating. The resulting project is essentially empty, ready for you to add your classes to.

It is not possible to change the project type once the project has been created. If you find that you need to change the project once you have started developing it, you will need to create a new project of the required type and copy in all of the existing code files.

Designing a Page

Now, we are ready to make some minor changes to your test application's default page. The page is named `MainPage.xaml` inside Solution Explorer and should open automatically when the project is created.

On the left half of the page editor window is the page designer and preview area. On the right half is the XAML code for the page. If you want, you can change the orientation of the divide by clicking the Horizontal Split or Vertical Split buttons at the bottom or right of the dividing bar between the two. The two parts can also be quickly swapped over by clicking the Swap Panes button, also located inside the divider. The designer panel also contains a useful zoom slider that allows you to focus more closely on the page design for detailed layout work, if required.

Initially, the page is a simple, empty, black page. For the purposes of this simple application, we will simply place a Button control onto the page and get it to display a message when clicked. The Button is added from the toolbox exactly as it would be for a desktop application: click the Button icon in the Toolbox panel (expand the `Common Windows Phone Controls` item if needed) and then draw it into the empty region in the center area of the page. The result can be seen in Figure 1-14.

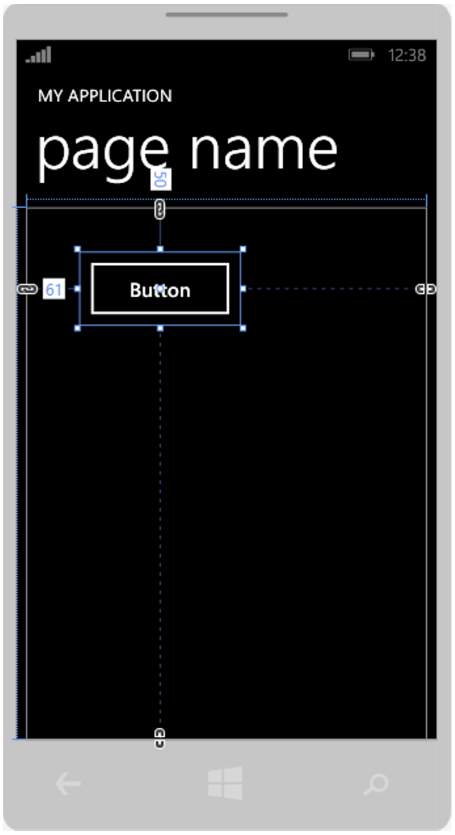


Figure 1-14. *The Windows Phone XAML page designer*

Once you have added your Button, take a look at its properties in the Properties window (this is pretty much the same as for the Windows Store project's Button properties, as shown back in Figure 1-7). If you are used to creating Silverlight applications in earlier versions of Visual Studio, you will probably recognize many of the available properties. If you have worked only with Windows Forms in the past, many of these properties might be unfamiliar, but there should also be some whose purpose is obvious. We'll look into many of these properties in greater detail in the XAML chapters later in this book.

Double-click the button to open the code designer and create the button's `Click` event handler. This will display the code-behind file for the page, which is where the C# code is developed. In Solution Explorer, you will see that the `MainPage.xaml` item has expanded to reveal `MainPage.xaml.cs` contained within. These two files are analogous to the form design and form code files that are present when working with Windows Forms.

The code within `MainPage.xaml.cs` should look just as you would expect when developing a desktop application. At this stage, it should be clear how much similarity there is within the Visual Studio IDE between desktop and Windows Phone (and Windows Store) application development.

Complete the implementation of the `Button_Click_1` procedure, as shown in Listing 1-2.

Listing 1-2. The Button_Click_1 procedure for the Windows Phone app

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Welcome to your first app!", "Message", MessageBoxButton.OK);
}
```

Running in the Emulator

We are now ready to compile and run the project. Press F5 or click the Run button in the Visual Studio toolbar to begin the process. After compilation (and assuming that there are no errors!), Visual Studio launches the Windows Phone emulator. This can take a few seconds to open, so be patient while this task completes. Subsequent deployments to the emulator will go much more quickly if the emulator is already running, so try to get into the habit of leaving it open when you switch back to Visual Studio between debug runs.

Once this is all complete, your program will launch. Clicking the button will display the `MessageBox`, as you would expect (see Figure 1-15).

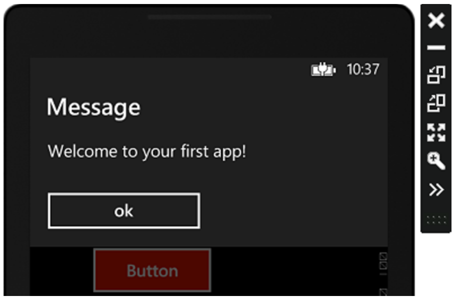


Figure 1-15. *The test application running in the Windows Phone emulator*

To stop the program, click the Stop Debugging button in Visual Studio. The IDE will return to edit mode, and your program will close on the emulator. The emulator will keep running, ready for any additional programs that you start.

Selecting an Emulator Image

Visual Studio actually ships with a number of different emulators, each representing a different device profile. Any one of these may be selected from the Debug Target drop-down in the Visual Studio toolbar. If you select an emulator image that is not already running, then it will launch. Otherwise, Visual Studio will reconnect to the existing running emulator image.

The emulator images that are available are as follows:

- *Emulator WVGA 512MB*: This emulates the lowest-spec device available for Windows Phone 8, both in terms of screen resolution and memory. The WVGA resolution is 480-by-800 pixels (the same resolution as used by all Windows Phone 7 devices). Most Windows Phone 8 devices have 1GB of RAM available, but some of the lower-cost devices have only 512MB available, as mirrored by this emulator image. This image therefore represents the “low-end” extreme of available devices, so you should ensure that your games run on this emulator without any display problems or crashes. For this reason, it can be useful to make this your standard development emulator so that problems can be detected as quickly as possible.
- *Emulator WVGA*: This image is the same as the WVGA 512MB image, except with the “standard” 1GB of RAM available.
- *Emulator WXGA*: This image keeps the standard 1GB of RAM but increases the screen resolution to 768-by-1280 pixels. This is the highest specification emulator image available. Consequently, it should represent the other extreme image against which to test. You will need to ensure that everything in your game scales correctly for display on this much higher-resolution display.
- *Emulator 720P*: This final image also has 1GB of RAM and uses a resolution of 720-by-1280 pixels—the standard 720p High Definition screen resolution.

Running on a Real Device

Switching your project across to run on a real phone is quick and painless and provides a very similar experience to running inside the emulator. One important difference between using a device and the emulator is that a device will give you a real indication of how fast your game is going to run. The emulator runs at the fastest speed it can, which is often much faster than on an actual phone.

There are a few steps that you will need to go through before you can begin working with the device, however. Let's take a look at the details now.

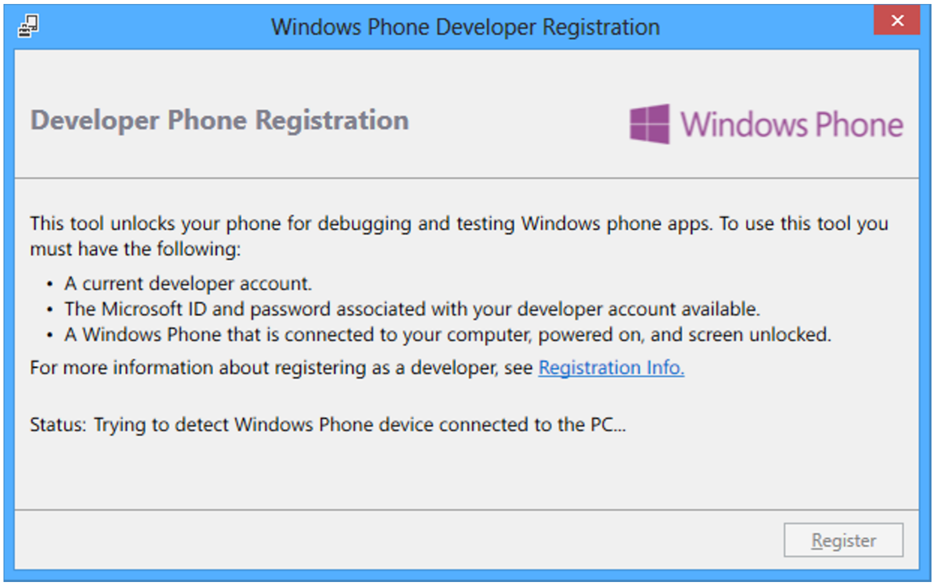
Registering the Device

Before you can deploy applications to a device, you must first have a Windows Phone developer account. These accounts are not free, though they are relatively inexpensive (currently priced at $99 per year, but this could change in the future). You will need this account before you can publish any of your finished games into the Windows Phone Marketplace anyway, so there is really no way to avoid this charge, even if you develop entirely on the emulator.

You can sign up for an account at the `developer.windowsphone.com/join` web page. You will be required to provide various pieces of information to Microsoft as part of this process and will receive notification messages telling you how to do this. The process might take a couple of days from start to finish. Besides providing your personal details, you will also be able to provide banking details so that you can be paid for applications that you create and sell in the Windows Phone Store.

Once your account is registered and active, the next step is to set the device up to connect to your PC. Unlike Windows 8 app development, Windows Phone apps are deployed from Visual Studio by connecting the phone to your PC via its USB connection. Plug the device in and allow Windows to set up its drivers. This should be a completely automatic process.

The next step is to register your device. This connects the device up to your developer account and allows Visual Studio to use it for deployment. To begin this process, launch the Windows Phone Developer Registration application, the main window of which can be seen in Figure 1-16. This application installed as part of the Windows Phone 8 SDK (or as part of Visual Studio Express for Windows Phone), and you should be able to find it in the Start menu. The application should connect to your phone and guide you through the registration process within a few seconds.


Windows Phone Developer Registration

Developer Phone Registration

Windows Phone

This tool unlocks your phone for debugging and testing Windows phone apps. To use this tool you must have the following:

- A current developer account.
- The Microsoft ID and password associated with your developer account available.
- A Windows Phone that is connected to your computer, powered on, and screen unlocked.

For more information about registering as a developer, see Registration Info.

Status: Trying to detect Windows Phone device connected to the PC...

Figure 1-16. *The Windows Phone Developer Registration application*

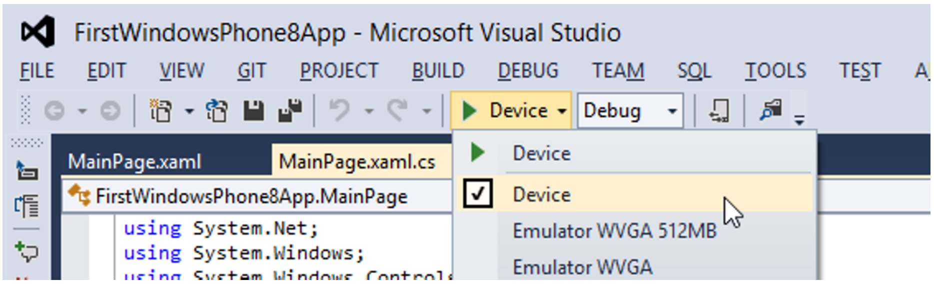
■ **Note** If you have previously developed for Windows Phone 7, you may notice that there is no mention of launching the Zune software in this description. Zune is no longer needed when developing for Windows Phone 8; Visual Studio can now connect directly to the device on its own.

At this point, you are finally ready to begin working with your phone in Visual Studio. These steps only need to be performed once for each device that you use.

Deploying to the Device from Visual Studio

Now we are ready to deploy our project to the device from inside Visual Studio.

Once everything is ready, choose to deploy your application to the device rather than the emulator. This is done by dropping down the Debug Target combo box in the toolbar and selecting Device, as shown in Figure 1-17. When you next launch your project, Visual Studio will connect to the device and then install and launch the application executable.


FirstWindowsPhone8App - Microsoft Visual Studio
FILE EDIT VIEW GIT PROJECT BUILD DEBUG TEAM SQL TOOLS TEST A
Device
Debug
MainPage.xaml
MainPage.xaml.cs
FirstWindowsPhone8App.MainPage
Device
Device
Emulator WVGA 512MB
Emulator WVGA

```
using System.Net;
using System.Windows;
using System.Windows.Control
```

Figure 1-17. Choosing to launch the application on a physical device

■ **Note** For deployment to succeed, the phone must be switched on and the lock screen must not be displayed. If these conditions are not met, a deployment error will occur, although Visual Studio usually provides good feedback about what the problem is to help you figure out how to proceed.

Unsurprisingly, the project running on the phone looks and behaves just as it did in the emulator. Congratulations—you have written and deployed your first Windows Phone application!

Debugging

Now that you have a simple application written and working, let's take a closer look at some of the debugging features that are available.

The powerful debugging tools that can be used within the Visual Studio IDE make development systems from the past look extremely primitive in comparison. We can use all these tools for Windows 8 and Windows Phone development, making tracking down problems simple.

Breakpoints

First, try setting a breakpoint on the line of code containing the `MessageDialog` or `MessageBox` function call. Launch the program (on a real device, the simulator or the emulator), and click the button within the page. As you would expect, the breakpoint triggers just as it would on a desktop application.

From here, you can explore all the usual attributes of your application: the call stack, object property windows, visualizers, and immediate window commands. Everything is present and working.

The one useful feature that is not available, however, is "edit and continue." Unfortunately, code changes cannot be applied at runtime and will either need to be held back until the IDE returns to edit mode, or you will need to restart your application in order for the changes to have any effect.

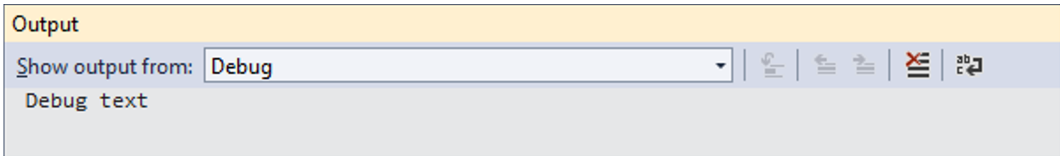
Debug Output

At any stage within your application, you can display text in Visual Studio's Output window. This is done in just the same way as for a desktop application by using the `System.Diagnostics.Debug` object. To test this, add the code shown in Listing 1-3 to the beginning of your `Button_Click_1` procedure.

Listing 1-3. Writing text to the Debug Output window

```
System.Diagnostics.Debug.WriteLine("Debug text");
```

Each time you click the button, you will now see your debug text appear within the IDE, as shown in Figure 1-18.



```
Debug text
```

Figure 1-18. *Debug text appearing in the Debug Output window*

■ **Tip** If the Output window is not displayed, it can be opened by selecting View/Output from Visual Studio's menu. If the Output window is open but no text is appearing, make sure that the "Show output from" combo box in the window toolbar is set to Debug, as shown in Figure 1-11.

Cross-Platform Development

This book's focus is on game development for both Windows 8 and Windows Phone 8. So far, the tiny example application that we have created has consisted of two separate and unrelated projects. If you intend to develop only for one of these platforms, there's probably little more to consider in this area. But, if you plan to develop for both, you will need to consider how to provide code that works across both of the platforms.

You may have already noticed that even with a trivial one-line application, the code is different between the two platforms. While it is true that the Windows 8 and Windows Phone product lines have become significantly closer together, they are sadly still some distance from one another.

This distance is reflected both in the way the devices operate (back button for Windows Phone, not for Windows 8; swiping gestures to control the environment for Windows 8, not for Windows Phone, and so on) and in their programming APIs.