Produced by Microsoft UK

Building Windows Phone Apps: A Developer's Guide

Contributors: Colin Eberhardt, Pete Vickers, Andy Gore, Mike Hole, Gergely Orosz, Sasha Kotlyar, Dominic Betts, Will Johnson, Ben Cameron, James Bearne, Samidip Basu, Paul Marsh, Stuart Lodge.

Table of contents

Foreword

It was with a sense of trepidation that I blogged a request for people who might like to help create a Windows Phone eBook back in February 2011. Despite my initial skepticism, there was an overwhelming response; both from people interested in the idea and those keen to volunteer their time and knowledge to help make it a reality.

I cannot believe that was less that six months ago. I must have aged at least six years in the meantime.

The idea was simple. Combine useful, technical information about the platform, written by those who'd been there and done it, with some stories about the people and the apps they'd written. The book would act as a handy reference that gave you a "puddy up" into developing on the platform and, importantly, helped you avoid making the same mistakes others had made.

The plan was always to publish an initial set of chapters and then iterate, adding new chapters and updating / refining existing chapters as we went. The initial iteration has taken much longer than I expected for many reasons. I'm just hoping we can do some performance tuning to make things happen a bit quicker next time around.

My role has mainly been one of coordination and the reviewing / editing of content. Without the help of all the contributors (see below) and Rachel Collier, Ginger Howard, John Donnelly and Martin Beeby at Microsoft, this thing would exist only on my whiteboard.

Mike Ormond
Developer Evangelist, Windows Phone
Microsoft Ltd
8th August 2011

Contributors

This book was created in the spirit of community. All the authors volunteered their time and knowledge to contribute chapters. None of them was paid and nobody will receive any form of royalties from the book. Authors are listed below in no particular order.

Colin Eberhardt

Technical Architect

Scott Logic (http://www.scottlogic.co.uk/)

Colin is a prolific technical author who writes on Microsoft .NET and Silverlight related topics. Colin also provides technical direction for many of Scott Logic's financial clients covering many technologies.

http://www.scottlogic.co.uk/blog/colin/
http://twitter.com/ColinEberhardt

Andy Gore

Director / Lead Consultant

Team G Software Limited

I've been working with .NET since version 1.0, back when I was a mere pup. My day to day job typically involves designing and developing enterprise level applications with the Microsoft platform. Particularly areas of expertise are ASP.NET MVC, WCF, SOA/ESB and BizTalk.

I've been working with Windows Phone 7 since the first beta, and have produced two main applications, Tube Companion and Top Cashback, both of which have huge updates coming for the Mango release.

http://twitter.com/Andy_Gore

Pete Vickers

Director – Technical and Consulting

APPAMundi (http://appamundi.com/)

I have been developing software for more years than I care to remember, and developing mobile software for about 10 years.

A founding partner in APPAMundi, mobile development specialists, we have several apps in the Marketplace including APPA Tasks, APPA Traffic, APPA FlightInfo, Where is my Car, Pop Quiz and FlipNDrink. We also publish apps for CEE.

We carried out several Windows Phone 7 Workshop days for Microsoft, working with developers to get apps published.

http://mobileworld.appamundi.com/blogs/petevickers
http://twitter.com/petevick

Mike Hole

Senior Solutions Developer

Sequence (http://sequence.co.uk)

Mike has primarily been working on web based projects working with SharePoint and Dynamics CRM. At Sequence Mike was the developer behind the Snow+Rock Pathfinder application. Mike is also behind the development of the TaxiRoute Windows Phone application.

Mike runs the Cardiff arm of the UK Windows Phone User Group (WPUG).

http://mikehole.com
http://twitter.com/mikehole

Will Johnson

Developer

Gas Street Works (http://www.gasstreetworks.com/)

I'm a developer who loves to build stuff. At work I build things for the web, at home I'm currently focused on Windows Phone, building apps like Pingdom Pulse and My Lovefilm Mobile.

http://www.madebywill.net/
http://twitter.com/willj

Ben Cameron

Freelance Developer

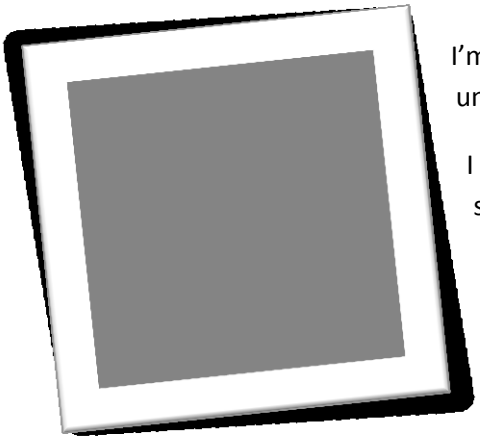
I am a passionate .NET dev who loves to keep up to speed with Microsoft technologies. Recently I've been spending my time mastering Silverlight, especially in the WP7 space as I expect this to be a growth area in the future. I've worked on a number of real world WP7 apps to date.

http://bencameron.blogspot.com/
http://twitter.com/ben_cameron00

James Bearne

Visual Designer

Sequence (http://sequence.co.uk)



I'm a 30 year old dad, husband and visual designer who has an unhealthy passion for sniffing freshly printed business cards.

I have a love for photography, design and all things that make me say 'I wish I'd done that'. I am currently working at Sequence as a visual designer on digital projects and It was here that I had the chance to design the Snow+Rock application for WP7 which gave me the opportunity to explore a new mobile platform.

http://twitter.com/Barnatronic

Dominic Betts

Principal Technologist

Content Master (a division of CM Group)
(http://www.contentmaster.com)

Dominic currently works at Content Master as a technical author and instructional design specialist, with a particular interest in Microsoft developer technologies. He has recently worked with Microsoft's patterns and practices group as the lead author on their "Windows Phone 7 Developer Guide" and, over the last two years, has worked with the same group at Microsoft on a number of Windows Azure guides. Before joining Content

Master, he worked as a freelance IT trainer, and in 2003 was the Institute of IT Training's Trainer of the Year.

http://cm-bloggers.blogspot.com/

Samidip Basu

Manager & MSFT Mobility Solutions Lead
Sogeti USA LLC (http://www.us.sogeti.com/)

Husband, work as Manager with Sogeti USA @ Columbus OH, Windows Phone 7, Mobile & Azure enthusiast, Total technology & gadget lover. Yes, a geek! I have had my hands in Windows Phone apps like Sogeti Hub, StirTrek, ShopTillYouDrop & soon-to-be-released Sogeti Battlecard. Spare times call for travel & culinary adventures with wife.

http://samidipbasu.com/
http://twitter.com/samidip

Paul Marsh

Lead Developer / Architect
Trace One (http://www.traceone.com)

Developer using mostly MS products, taking any excuse to use; Silverlight, WP7, MonoTouch and now MonoDroid too. In order to help learn WP7 I developed and published 'Steam Achievements' with design by Hannah Watkins. I'm keen to promote Windows Phone as a platform for quality applications.

http://pauliom.wordpress.com/
http://twitter.com/pauliom

Sasha Kotlyar

Sr Software Engineer
Auri Group (http://aurigroup.com)

I'm a developer who has recently been focusing more and more on mobile app development. Apps I've written for WP7 include the popular ArkWords - a dictionary, thesaurus, and translator; and Network Suite - an app that performs ping, TCP connection testing, WHOIS, and DNS lookups.

http://arktronic.com/
http://twitter.com/Arktronic

Gergely Orosz

WPF Developer

J P Morgan (http://jpmorgan.com)

Gergely has been developing for WP7 since its announcement. He co-developed Cocktail Flow, one of the highest rated applications on the platform and AppFlow, a marketplace discovery utility, both together with Distinction Ltd. Other apps developed include Flashlight 7 and Bus Tracker Edinburgh.

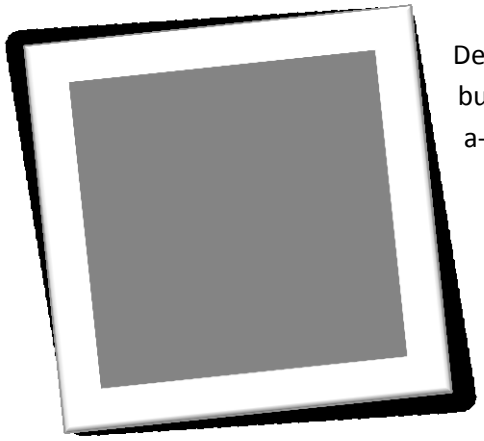
http://gergelyorosz.com/

http://twitter.com/GergelyOrosz

Stuart Lodge

Project Manager, Developer and Dogsbody

Cirrious Ltd



Developer since I first got my hands on a Sinclair ZX80. In WP7, I've built lots of apps including RunSat, Overflow7, Iron7 and Translate-a-Bull.

http://twitter.com/slodge

Introduction

By Colin Eberhardt

The science fiction writers of the last century dreamed up a vision of ubiquitous computing; intrepid intergalactic explorers with powerful computers and communication devices in the palm of their hand. Whilst we haven't quite managed the intergalactic part yet[1], as you walk around towns and cities, travel on trains and subways, virtually everyone has a powerful computer in the palm of their hands. The smartphone market, which Windows Phone 7 is a part of, feels almost ubiquitous to many of us.

The ubiquity of computing has arrived so quickly that it can almost feel like it happened overnight. The way in which we interact with computers is changing fast – gone are the days where your only interface with a computer was via a keyboard, monitor and mouse. Today, computer interaction is more tactile and visual, with user experiences enhanced by powerful graphics and location-aware applications. While the desktop has been evolving slowly, it is smartphones that are on the cutting edge of this evolution in how we interact with computers, and how computers interact with the world around us.

Over the past couple of years, Microsoft's Windows Mobile smartphone OS has been losing market share to its competitors, primarily iPhone and Android, which have brought a whole new cool dimension to smartphones, appealing to the younger generation of users. With Windows Phone 7, Microsoft started afresh, a reboot; without being constrained by the old enterprise-focused Windows Mobile, they were able to bring something entirely new to the market.

Launched at the back end of 2010, Windows Phone 7 brought with it a totally new Windows smartphone experience, with powerful hardware specifications, camera, GPS and the Metro design language – a highly recognisable style. Windows Phone 7 breathes new life into the Windows smartphone product line. The Metro style very much defines the Windows Phone 7 experience, with stark colours, clear text and fluid transitions combining to make a style that is instantly recognisable, even at the smallest size!

For developers, Windows Phone 7 is equally exciting, supporting the popular Silverlight and XNA frameworks. Applications are developed with Visual Studio and Expression Blend, giving a familiar environment for developers working with other .NET technologies. The emulator and on-device debugging are also fully integrated. This gives an excellent opportunity for existing XNA or Silverlight developers to transfer their skills to mobile devices. What about people without XNA or Silverlight

[1] Windows Phone 7 may not have gone intergalactic, but it has almost gone into outer space, with a team of meteorologists from Southampton (UK) tethering one of the devices to a weather balloon, using it as a cheap GPS-enabled computer!

experience? Perhaps now is the time to learn! There is something quite exciting about developing for a pocket-sized gadget, especially one that is location aware, sensitive to touch and tactile.

Having both the XNA and Silverlight frameworks at your disposal is something of a treat. Silverlight is typically used for business or utility applications. The user interface is defined in XAML (an XML-based markup language). You have suites of common controls, lists, buttons, images at your disposal, with the interaction logic written in 'code behind' or packaged in behaviours responding to events raised by these controls, or changes in data which is 'bound' to the UI. However, where Silverlight really comes into its own is in its powerful styling and templating features that give you full control over how your UI is rendered, with storyboards, animations and render transformation allowing you to create smooth effects and transitions for a truly engaging user experience.

The XNA framework is geared towards games development, bringing with it 2D sprite-based graphics and a 3D rendering engine. Just as Silverlight is familiar to web developers (and desktop developers of WPF applications), XNA is familiar to a whole host of Xbox and PC game developers. Existing XNA developers can now enjoy the challenge of creating games for the small screen, with added extras of multi-touch and an accelerometer, allowing the creation of games that response to tilt and shake.

Whilst both XNA and Silverlight give developers any easy path for mobile development, it is worth taking a little time to think about the mobile form factor before simply porting an existing game or application directly. There are simple differences, such as the very high screen DPI, which means that larger font sizes should be used. But other significant differences, such as the lack of a mouse, the overall small screen size, and possibilities such as UIs which are controlled via multi-touch or tilt/shake, are worth exploring. The Windows Phone 7 Silverlight framework includes two mobile-specific controls, the Pivot and Panorama, which are designed specifically to make the most of the mobile form factor, whilst adhering to the Metro theme.

Microsoft does not manufacture the Windows Phone 7 hardware itself; they have instead issued a minimum hardware specification. As a developer, this means you can guarantee that any phone running your code will have a camera, a 4-point multi-touch screen, GPS, accelerometer and three buttons (back, start and search). However, there will be differences: manufacturers can choose to include a hardware keyboard, for example. You need to be aware of these potential differences.

Windows Phone 7 applications are all distributed via the Microsoft Marketplace although, during development, you can upload the code to your phone directly from Visual Studio. Applications submitted to the marketplace must adhere to certain content restrictions and minimum quality standards in order to be published. Whilst Metro style is encouraged, it is not enforced, giving you a lot of freedom in terms of styling and branding your application.

The world of Windows Phone 7 development offers a new and exciting environment, giving the developer a rich landscape to explore. This book is not a beginner's guide to Silverlight, XNA or Windows Phone 7 development; rather, it is a collection of insights, tips and tricks, advice and stories from developers who have already started on their own Windows Phone 7 journey. It is the intention that this book will inspire you to join them, so read on (and why not go ahead and start downloading the tools whilst you are reading) and enjoy the ride.

Interlude: Football Crazy

INTERLUDE
Football Crazy

MyFC (www.manageyourfootballclub.com) lets you run one of 170 football clubs from the English or other European leagues. With a beautiful, fluid Windows Phone 7 Metro user interface, the game is a must-play for football fans. We spoke to Roger Womack (producer, pictured) and Arden Aspinall (developer) at My Interactive Limited (www.myinteractive.co.uk) to find out how they got on, bringing the game to Windows Phone 7.

Tell us a little about the history of MyFC

We've been working on football manager games almost exclusively for 18 years and the engine has been evolving constantly since then. Originally it was in C++ and then we ported it to Java for Android but that version was a bit too slow so we don't use it any more. But it made the conversion to C# for Windows Phone 7 (WP7) less time-consuming.

How much experience did you have with the platform when you started?

None of us had worked on Silverlight before but we found it had a quick learning curve. We started in June 2010 with the Technology Advance Preview and, apart from converting the engine, we rewrote the whole user interface to fit in with the Microsoft Metro style. For us, that was the most challenging thing.

How did that go?

There are many screens in a football manager game – at least 100 – that had to be coded and scripted separately. But without a shadow of a doubt, WP7 is far ahead of the crowd for speed of development. We got the product ready for launch day on time and on budget. The strong tools and support made that possible. On Android, it would have been a push, to put it mildly.

How big was the team and how long did it take?

At any one time, we had up to three developers and an artist working on the project from July to October 2010. Our other WP7 projects were smaller. For example, Snooker was just two people and it took a matter of weeks. But MyFC is a big project with nearly 300 classes and a lot of screens.

What did you learn?

Once we had done the first WP7 project, the others went much faster because we had already got the hang of C# and Silverlight. Also, we found a really fast way of refactoring C++ to C# quickly. Copy constructors and pointers to pointers took a while to figure out but we have good ways of using them now. Also, we had to work hard to get the startup time down.

What would you tell other developers if they were starting today?

Just do it. Give it a try and check out all the resources and answers online. You can save yourself a LOT of time that way.

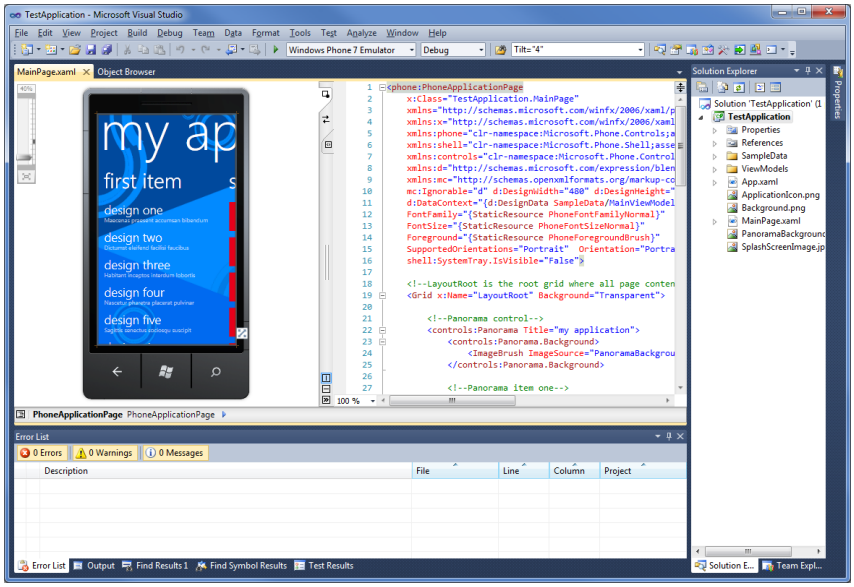

17 Aug 2010
Transfer List
Filter On
Club Balance
£8,335,388
Clb
Player
Skill
Value
J.Shelvey
Midfielder (C)
63
£1,107,091
M.Johnson
Midfielder (R)
63
£1,255,495
J.Thomas
Midfielder (R)
62
£1,087,389
J.Morrison
Midfielder (R)
61
£1,088,310
S.Schaors
Midfielder (C)
71
£2,886,860
N.Zimleng
Midfielder (L)
62
£1,124,543
G.Douming
Midfielder (R)
63
£1,383,170
A.Otti
70
£2,245,118
1
24 Jul 2010
Match
Speed x1
Chelsea
Man Utd
2
:124
2
GOAL!
Timeline
F.Lampard
M.Carrick
F.Malouda
J.Park
Y.Benayoun
P.Scholes
4

The Windows Phone Developer Tools

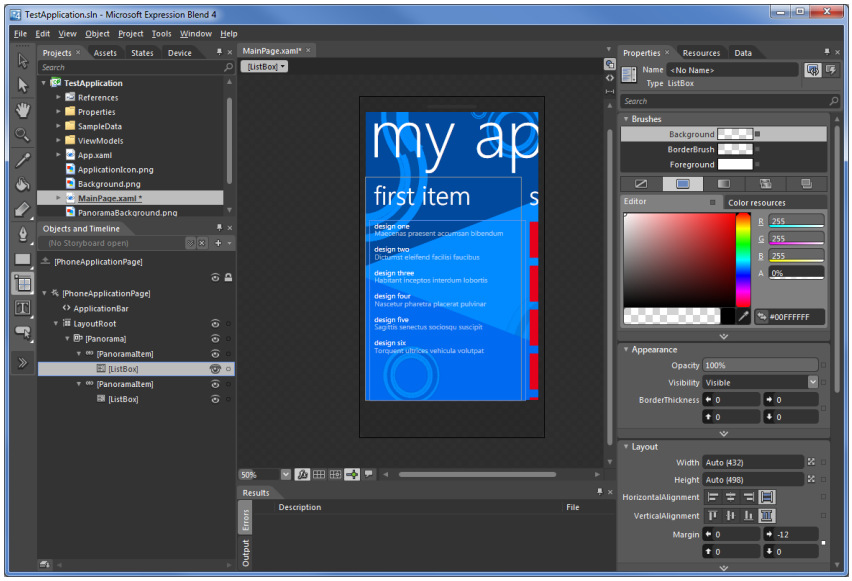
By Colin Eberhardt

I think most people are familiar with the adage that “a good craftsman never blames his tools”, which to paraphrase, means that it is the skill of the craftsman, not the quality of the tools, that dictate the quality of the finished product. Whilst there is every truth to this saying, give any skilled craftsman the choice between a poor set of tools, or those of the finest quality, and they will take the quality ones every time! This is because quality tools allow skilled craftsmen to get their job done much more quickly. The same is true of quality software development tools.

Microsoft is widely recognised as an industry-leading provider of developer tools, and as a Windows Phone 7 developer, you will benefit from this. Phone applications that are written using the Silverlight or XNA frameworks are both written within Visual Studio. If you are already an experienced .NET developer, you will enjoy the familiarity of this environment.



More recently, Microsoft added Expression Blend to their suite of development tools. Whereas Visual Studio is a developer-focused environment, Blend offers a complementary environment for the designer. Blend relies on the clean separation of the View from the logic behind it that is made possible by XAML (the XML markup used to define your user interface). This allows both the developers and designers to work together on the same codebase without interfering with each other’s work. Again, phone developers using the Silverlight framework get the full benefit of the developer-designer workflow that these tools support.



Visual Studio and Blend are a powerful combination; however, probably the most important tool for Windows Phone 7 developers is the emulator, which allows you to run phone applications directly on your desktop.

In this chapter, we will look at how to obtain these tools and how they support the phone application development process. It is assumed that you have previous Silverlight or XNA development experience and have also used Visual Studio before. This chapter will therefore focus on the tools that are specific to phone development.

Getting Started

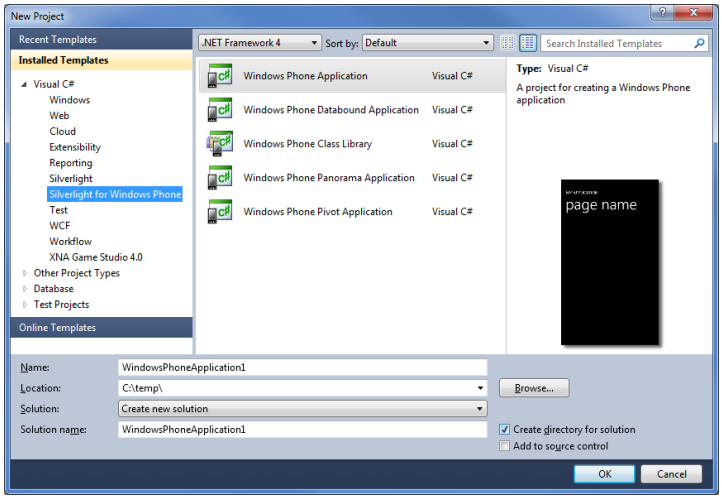
The first step towards Windows Phone 7 development is to download the development tools. These are freely available as one of the 'express' editions of Visual Studio. You can find the latest details of which tools to download and any updates via the App Hub Getting Started pages.

Microsoft Visual Studio 2010 Express for Windows Phone

The key tools included within this express edition are: Visual Studio, Expression Blend, the Silverlight and XNA frameworks and the Windows Phone Emulator.

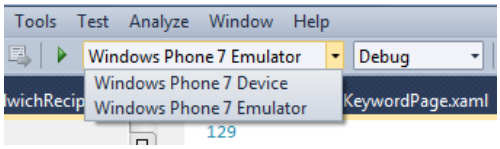
These tools give you everything you need to develop applications for your phone.

The version of Visual Studio included in this express edition contains a number of project templates for Silverlight or XNA development; each will give you the basic project setup and a few hints regarding where you should place your code. If you create a project based on one of these templates, compiling the generated code and starting the application should launch the emulator.



The Emulator

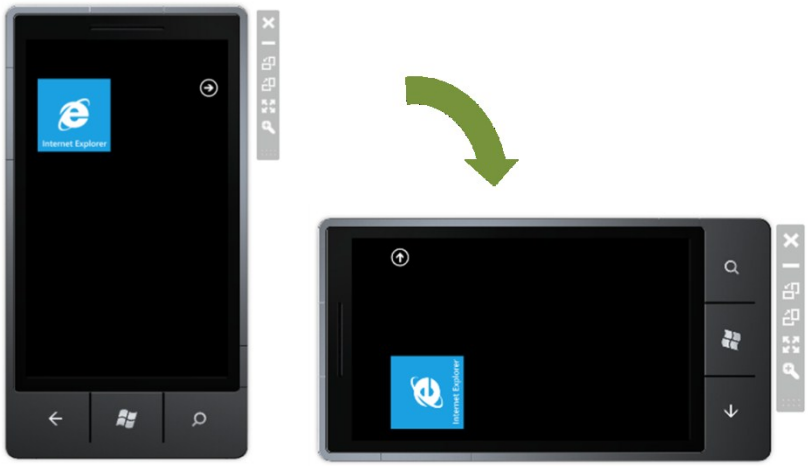
During application development, you can select to run your code on a physical device or the emulator:



The emulator not only gives you the opportunity to run your code more quickly, reducing development cycles, but also gives developers who do not own a physical device the opportunity to try their hand at phone development.

Typically, you will leave the emulator running while you are developing. Each time you execute your code, the latest version will be deployed to the emulator.

The emulator has its own miniature toolbar that allows you to change its orientation and size. On a typical monitor, a zoom of approximately 40% makes the emulator roughly the same size as a physical device. However, the resolution of the phone screen is much higher than that of a typical monitor, so you will probably find yourself playing with the emulator zoom quite often!



The emulator has Internet Explorer installed and the Settings application; other than that, it is a clean slate. Interestingly, if you do not close down the emulator, you can deploy multiple applications, and also pin your apps to the start screen.

Simulation of Phone Features

The emulator is a fantastic tool for most purposes, from experience; it really is an excellent representation of how your code will run on a real device. However, there are some limitations to what you can emulate. We'll take a brief look at some of these limitations, and the novel solutions that the developer community have come up with, in this section.

The phone accelerometer allows you to detect phone movement in all three axes. This feature opens up great opportunities for game developers and also enables the phone to support shake gestures. Changes in accelerometer readings can be detected via an event in both Silverlight and XNA applications. Unfortunately, the emulator does not provide a mechanism for simulating accelerometer input. Fortunately, it is a relatively straightforward process to replace this event with some other mock source for the purposes of emulation. For example, Mateus Velloso describes using keyboard input as source of data, whereas for the more ambitious, it is even possible to use a Nintendo Wii remote for accelerometer readings!

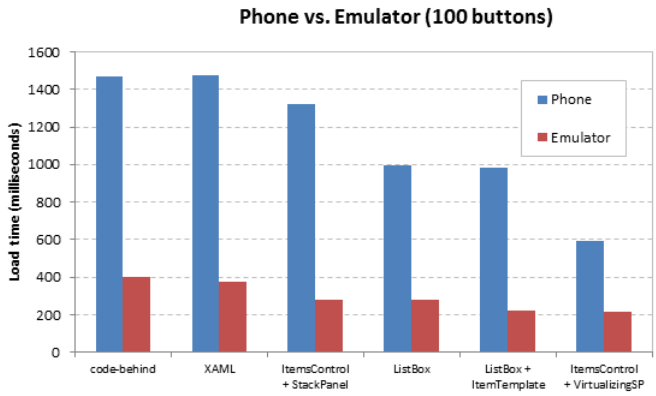
Multi-touch is an important part of the Windows Phone 7 experience, providing users with a tactile and exciting interface to use. If you are fortunate enough to have a multi-touch monitor or tablet, the phone emulator will accommodate this. However, most desktop computers still rely on the standard, monitor/keyboard/mouse configuration. If your phone application relies on multi-touch gestures, you will need to find a way of simulating this with the emulator. A popular solution to this problem is the Multi-TouchVista CodePlex project (http://multitouchvista.codeplex.com/), which includes a multi-touch driver for Windows 7 that can accept input from a number of sources. Mihail Mateev has published a tutorial which details how the Multi-TouchVista drivers can be used in conjunction with a second USB mouse to supply multi-touch input to the emulator. Again, the more ambitious can build their own multi-touch input device. Szymon Kobalczyk has described a novel approach where he uses a web-cam positioned under a 'light box' and image processing to detect multiple finger locations!

With GPS, your phone is able to pinpoint its location (in terms of latitude and longitude) with a remarkable level of accuracy. The use of GPS allows you to develop location-aware applications; a popular usage of this feature in games is to provide regional league tables. The Geolocation API is quite similar to the accelerometer API, with an event being raised each time the phone location changes. Again, the emulator does not provide a mechanism for simulating this data. One interesting method of emulating GPS involves using a Bing Maps client, with the selected location being fed to the phone via a WCF interface.

Performance Considerations

With the various tricks described above, it is quite possible to develop applications or games without having to purchase a phone yourself. However, there is one final aspect that you should consider when developing code using the emulator ... and that is performance. The emulator is not designed to provide a faithful emulation of the performance of a real device; it simply executes your code as fast as it can. As a result, the application performance will depend on the hardware you are running your emulator on. Most developers will have pretty powerful desktop machines which can execute their application code on the emulator far faster than a physical device can.

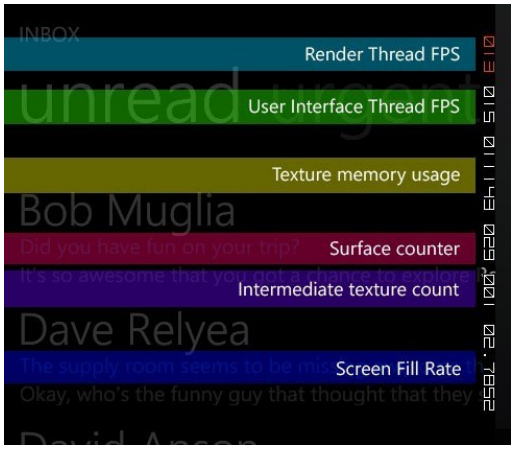
Page load times and user interface responsiveness have a big impact on the usability of phone applications. The chart below shows the recorded page load times on an emulator versus the same page load times on a physical device:



The difference between hardware and an emulator is significant. For this reason, it is important to test your application on a real device as often as you can. It is also worth doing some experimentation in order to learn the performance profile of the hardware and which operations are more expensive than others. As an example, moving from floating point to integer maths will make your code run faster in the emulator, but surprisingly, this same optimisation has been found to have the opposite effect on a physical device!

Performance Counters

When you execute your code in debug mode on the emulator, or a physical device, a number of frame rate counters are displayed. These are illustrated below:



You can also turn these counters on and off programmatically as follows:

```
private void FrameRateCounters_Checked(object sender, RoutedEventArgs e)
{
   var checkbox = (CheckBox)sender;
   var newValue = checkbox.IsChecked.GetValueOrDefault();

   Application.Current.Host.Settings.EnableFrameRateCounter = newValue;
   SystemTray.IsVisible = !newValue; // Hides frame rate counter otherwise
}
```

The counters are as follows:

User Interface Thread FPS – The number of frames per second that the main user interface thread is experiencing. This thread handles user input and performs primary code execution; it includes data binding and parsing XAML to construct the visual tree. It is recommended that you aim for a rate of above 15fps. If the frame rate drops below 15fps, user experience will suffer and this counter will turn red.

Render (or Composition) Thread FPS – The number of frames per second that the render/composition thread is experiencing. This thread handles some of the work that the UI thread would typically handle in web-based Silverlight applications, such as storyboard-driven animations. It is recommended that this counter is kept around 60fps. If the frame rate drops below 30fps, user experience will suffer and this counter will turn red.

Texture memory usage – This value represents the video memory and system memory copies of textures being used in the application.

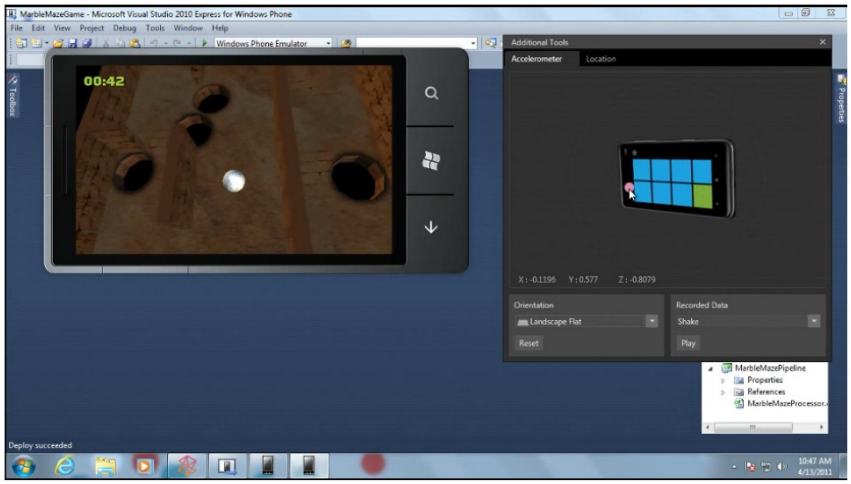
Surface counter – A count of the number of surfaces that are passed to the graphics chip.

Intermediate texture count – The number of intermediate textures created for compositing.

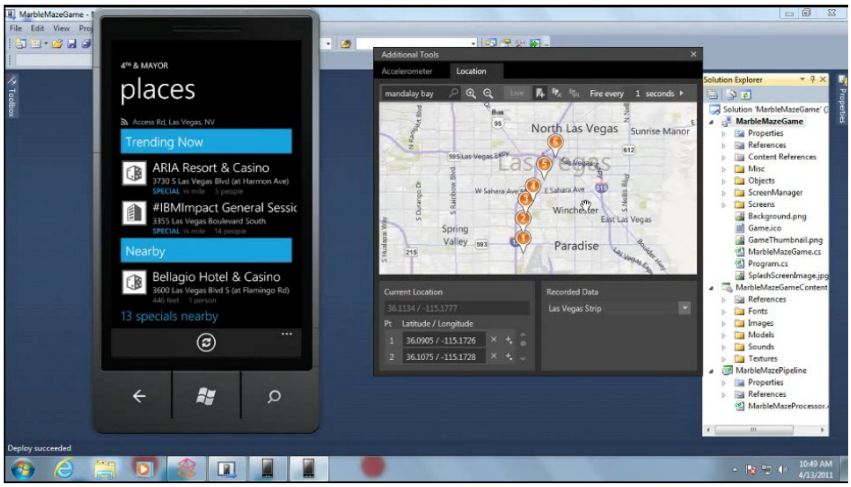
Screen fill rate – This counter indicates the number of pixels being painted per frame. A value of 1 represents 480 x 800 pixels. It is recommended that this value does not exceed 2.5, otherwise the overall frame rate will begin to drop.

Coming in Mango

At the MIX11 conference which was held in April 2011, a number of new and exciting Windows Phone 7 features were announced which will be coming with the Mango update to the OS. A number of improvements to the phone developer tools were also announced. The next updates will include a much more feature-rich emulator where it will be possible to simulate the accelerometer by manipulating a 3D model of the phone:

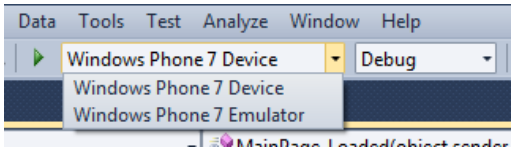


It will also be possible to simulate GPS data via a Bing Map which is embedded within the emulator tools. You can also record and replay routes:

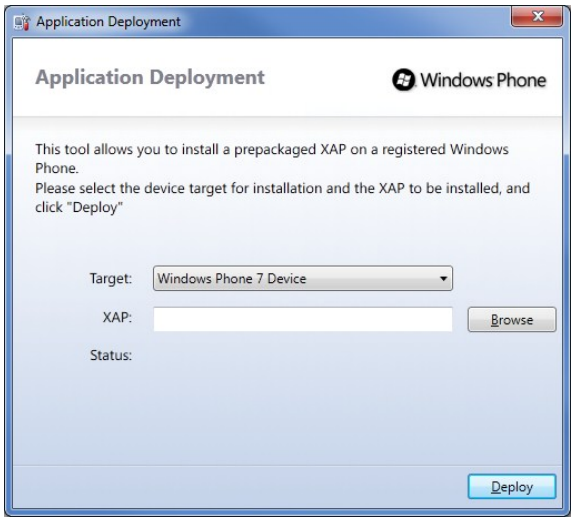


Developing With a Physical Device

The deployment and execution of code on a physical device (i.e. a real phone!) is just as easy as using the emulator. Simply select Windows Phone 7 Device as the target and execute your code as you would on the emulator. Your code will be deployed to your developer-unlocked phone and the application will start. The development experience on a real phone is exactly the same as with the emulator; you can view debug output and even set breakpoints in the code running on the phone.



You can also deploy Silverlight XAP files to your phone without Visual Studio by using the Application Deployment tool. This tool is useful if a fellow phone developer wants you to test their application and sends you a copy of their compiled XAP file:



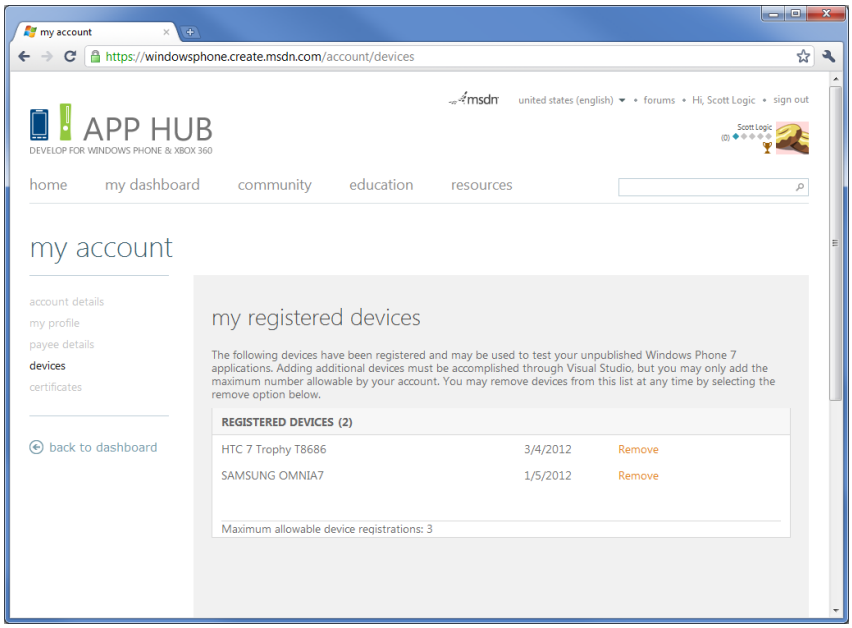
Registering a Phone

There are a few steps you need to follow in order to unlock a phone for the deployment of development applications. Firstly, you will need the following:

- A Windows Live ID
- The Zune software, which is used for transferring data to and from your phone
- A subscription to the App Hub, which costs $99 per annum

Once you have the above, you need to run the Developer Phone Registration tool:

When your phone is successfully registered, it will appear under the list of registered devices on your App Hub account summary page:



Once you have successfully completed this process, it is simply a matter of connecting your phone to your computer; the Zune software will automatically be launched, then you can deploy and run your code.

Summary

Developing for Windows Phone 7 is a fantastic experience, with the maturity and familiarity of Visual Studio and Expression Blend making it easy to transition from developing with other .NET technologies to the phone. The tools are free to download and the emulator makes development without a device possible, allowing anyone to try their hand at Windows Phone 7 development.

Interlude: Going underground

UP INTERLUDE
Going Underground

Andy Gore (@Andy_Gore) built the popular Tube Companion for Windows Phone 7. It was a weekend and evening project for this busy C# developer – a "bit of a geek hobby" in his words. How easy is it for one person to build their own app? We talked to Andy to find out.

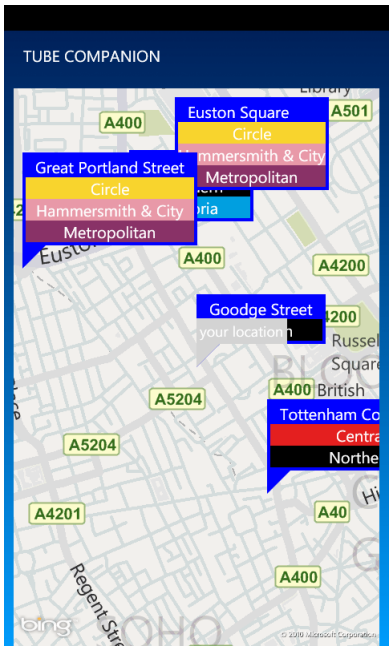
Andy started developing Tube Companion as a learning exercise to understand Windows Phone 7 development. He was (and is) a keen contributor to the blogging community around the platform and this inspired him to try to build a fully-fledged commercial product. "I was working on this by myself and I wanted to make a quality product," he says.

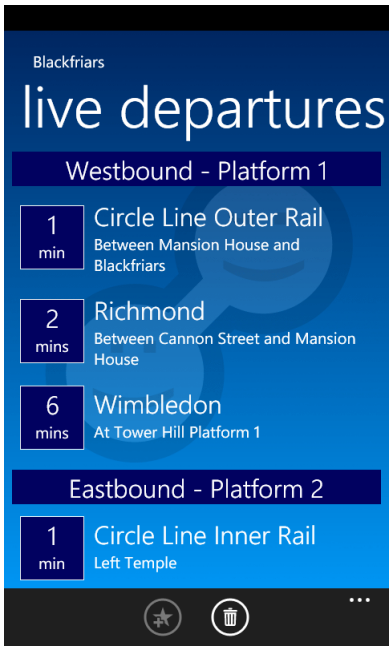
He worked on the project in his own time at home. He runs his own company designing and developing .NET enterprise projects primarily using ASP.NET, MVC and WCF so, as an experienced C# developer; he found it easy to get started.

However, he was not familiar with Silverlight or the phone-specific APIs in WP7, such as push notifications. This is where the App Hub website, forums and blogs came in handy. "It wasn't a big challenge to learn and the WP7 toolkit and APIs are really, really easy to use."

His Tube Companion is a free app, so it has had a lot of downloads. This has raised Andy's profile and he plans to split his time between his existing enterprise work and paid-for WP7 development. He says, "Sure, it's exciting to do something new, and this definitely started off as a geek love affair, but it's developed into a real commercial opportunity for me."

Perhaps the biggest pay-off, at least on a personal level, is when he actually sees people using his app on the Underground. This has happened a couple of times and "It's always a big thrill."





The Application Lifecycle

By Dominic Betts

A Windows Phone 7 device must manage a conflicting set of demands from a user:

1. Users expect their mobile device to be multi-tasking, just like a desktop machine.
2. Users expect applications on the device to be highly responsive, with minimal delays due to processing requirements or task switching. Again, just like on a desktop machine.
3. Users want their battery to last for as long as possible.

Although battery technology is improving all the time, it's set to remain a limited resource on mobile devices for the foreseeable future, so the operating system and your applications should do anything they can to minimize their power requirements without compromising the functionality available to the user.

The remainder of this chapter describes the application lifecycle on the Windows Phone 7 platform, from when the user first launches an application, to when the application is finally shutdown. As you'll see, much of the rationale behind the design of the lifecycle on Windows Phone 7 is derived from the need to conserve battery life. You'll also see how you can help Windows Phone 7 deliver on the three demands above by designing your applications appropriately.

Only One Application at a Time

Although users have come to expect multi-tasking behaviour on mobile devices, having many applications running concurrently is a big drain on the battery. The solution adopted on the Windows Phone 7 platform is to only allow a single application to run: the current foreground application. To give the illusion of multi-tasking, the platform helps you to save and restore state data for your application as the user switches to and from other tasks. It's down to you to make sure that you save enough state data when the user switches away from your application, so that if and when they switch back, you have enough information to restore the application to its original state. However, you must save and restore the application state within a fixed time period, otherwise your application won't meet the Windows Marketplace application certification requirements.

As you can see, all three of the user demands introduced above are involved in this process: multi-tasking, responsive UI, and long battery life.

Application Lifecycle

Before looking at the details of how to save and restore your application state, it's useful to get an overview of the process. The following diagram is adapted from the Windows Phone 7 documentation on MSDN and captures all of the key aspects of the lifecycle that you should be aware of when you are writing your Windows Phone 7 application.

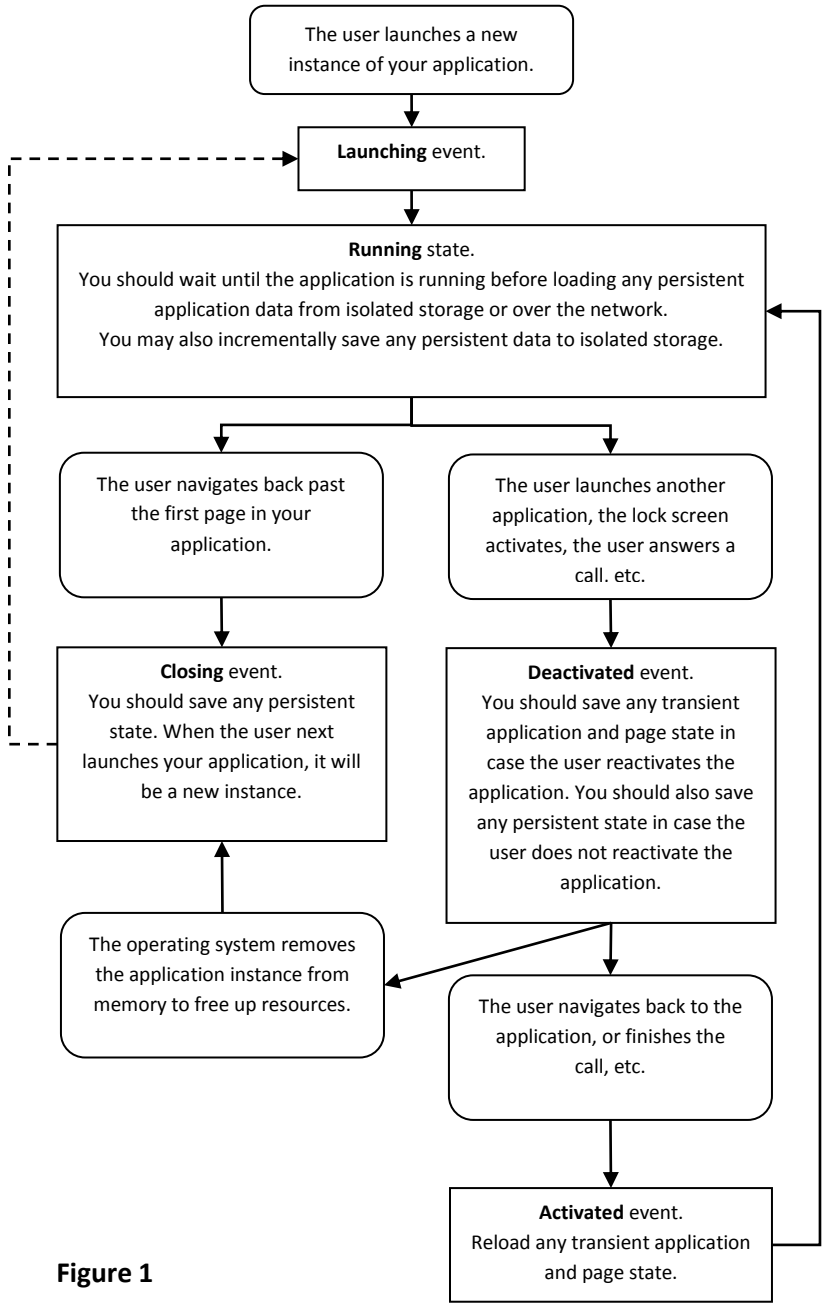


Figure 1

Transient, Page and Persistent State

In Figure 1, there are three types of state data: persistent state, transient application state, and transient page state. The first, persistent state is standard application data that your user does not want to lose: you store this type of data in isolated storage or, perhaps, in a Web service. In Windows Phone 7, your application should save this data automatically because users don't expect to have to click on a Save button (and may not be able to if the application has been deactivated). Transient application data is the data that you need to reconstruct the application state if the application is reactivated. Transient page state is the data you need to reconstruct the UI state of a page if a user navigates back to the page after the application has been reactivated. You should not care if the transient state data is lost.

The Deactivated and Closing Events

The **Closing** event fires when the application is being completely shutdown, so this is your last chance to save any persistent data. In a standard Windows Phone 7 project, you'll find a stub event handler in the App.xaml.cs file.

The **Deactivated** event fires when the application is being temporarily suspended (although you mustn't assume it will always get reactivated so you do want to save any persistent data here). This is your opportunity to save any transient application state that you can use to reconstruct the application's state later. This process of removing the running application from memory, but saving any transient state data is often referred to as 'tombstoning.'

In the **Deactivated** event, you don't need to record which page the user is currently viewing. The Windows Phone 7 platform automatically stores the navigation stack. You use the **PhoneApplicationService** object to save your data, which must be serializable. You could invoke the following method from the **Application_Deactivated** handler in the App.xaml.cs file to save your transient state data. This method is part of a simple utility class named **Tombstoning.**

```
public static void Save(string key, object value)
{
 if (PhoneApplicationService.Current.State.ContainsKey(key))
 {
  PhoneApplicationService.Current.State.Remove(key);
 }

 PhoneApplicationService.Current.State.Add(key, value);
}
```

Finally, to meet the Windows Marketplace application certification requirements, you must ensure that the **Deactivated** event completes within 10 seconds. Applications that fail to do so are terminated by the OS and will fail Markeplace submission.

The Activated and Launching Events

The **Launching** event fires whenever the application is starting up from scratch, so there's no transient state data involved. Note that you should keep the work done in the Launching handler to a minimum as this delays application startup. Specifically you shouldn't load persistent state data (eg from Isolated Storage or over the network). Instead you should wait until your first page displays, and then load your persistent data. To improve the user experience, you should consider loading the persistent data asynchronously.

The **Activated** event is your opportunity to load transient application state data that you saved in the **Deactivated** event. Remember that your application will automatically display the page that the user was last viewing, and that the navigation stack will be reconstructed for you. You may also want to reload your user's persistent data during this event.

You could invoke the following method from the **Application_Activated** handler in the App.xaml.cs file to load your transient state data. This method is part of a simple utility class named **Tombstoning.**

```
public static T Load<T>(string key)
{
  object result;

  if (!PhoneApplicationService.Current.State.TryGetValue(
    key, out result))
  {
    result = default(T);
  }
  else
  {
    PhoneApplicationService.Current.State.Remove(key);
  }

  return (T)result;
}
```

Like the **Deactivated** event, the **Activated** event must complete within 10 seconds.

Summary

To help maximise battery life, Windows Phone 7 only runs a single application at a time. However, by saving and restoring transient application state, the illusion of a multi-tasking environment is maintained and as long as the saving and restoring of transient application state happens quickly enough the user will be happy with the responsiveness of the UI.

Interlude: Snow+Rock

Mike Hole is a senior solutions developer for Sequence, a Cardiff-based digital agency. He created a Windows Phone application for Snow+Rock called Pathfinder. (He also wrote the Windows Phone app for TaxiRoute.co.uk.)

Outdoorsy types can use it to discover hiking trails and track their progress along them, or they can create the own trail and share it with the Snow+Rock community. There's also a Snow+Rock shop on the app featuring appropriate clothing and accessories.

In his 'day job', Mike works on SharePoint applications and Microsoft Dynamics customisation. However, he was interested in Windows Phone development personally and began to teach himself how to do it. When the opportunity came up at work to develop the Snow+Rock app, he was the logical choice.

Mike had done some Silverlight programming before but the challenge for Snow+Rock was to create an app that looked as good as the mock-ups created by the agency's designers. (It succeeds!)

“Then the workings of the phone posed some additional puzzles,” he explains. For example, you have to deal with incoming calls and interruptions by storing the app’s state. Then there was the challenge of learning how to use the geolocation system so that the app could actually track the user’s movement along a trail.

Luckily, during testing, Mike was on a “bit of a health kick.” He was able to test the application on his daily run and this helped to iron out several problems.

The development ran from July to September 2010 working full time – a quick project, thanks to the “easy development environment and powerful emulator for testing.”

When he’s not building apps, working for Sequence or running, Mike is also a leading light in the Cardiff Windows Phone user group.

Accessing Phone Features (launchers, choosers and input features)

By Pete Vickers

Windows Phone 7 contains some neat features for carrying out tasks which were always difficult in early versions of Windows phones.

Windows Phone 7 applications are isolated in their own sandbox, for both execution and file storage. Because of this, applications are not able to access information stores such as contact lists, nor are they able to invoke other applications directly such as the phone, or messaging applications. To enable applications to use these common tasks, Windows Phone exposes a set of APIs known as Launchers and Choosers.

Launchers

A launcher is an API that launches one of the built-in applications where no data is returned to the calling program. Using PhoneCallTask as an example, your application supplies a name and phone number and invokes the task. The user can then decide whether to place or cancel the call. When the user ends the phone application, control is passed back to your application, but your application does not get a return value. The launchers you can use are:

- Make a phone call
- Compose an SMS
- Compose an email
- Launch the web browser
- Launch a search
- Launch the media player
- Marketplace details
- Marketplace hub
- Marketplace review
- Marketplace search

Choosers

A chooser is an API that launches one of the built-in applications and returns some kind of data to the calling application. Using the PhotoChooserTask as an example, the application can use this chooser to show the photo chooser application. The user can then either select a photograph, or cancel out of the operation. When this happens, the calling application is reactivated and the result of the operation is returned. From this, the application knows if the user has chosen a photograph, or cancelled the operation. Bear in mind, when the user is in the chooser application, they can press the Start button, and control may never be returned to your application.

The choosers you can use are:

- Choose a photograph
- Take a photograph
- Choose a phone number
- Save a phone number
- Choose an email address
- Save an email address

Launchers, Choosers And Tombstoning

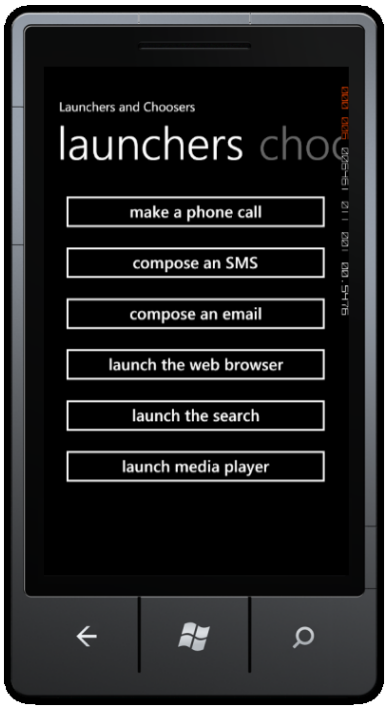
It is important to remember that when a launcher or a chooser is activated, your application is terminated, and you must use tombstoning to ensure the data you need in your application is stored by the system.

In order to use launchers and choosers, you will need to include the Microsoft.Phone.Tasks namespace in your application. The following sections will illustrate how to use launchers and choosers, and how they can be combined to perform particular actions.

Launchers

Using a launcher is straightforward. Create an instance of the relevant task object, assign values to its properties, and then call the Show() method to action the task.

The following examples are available as downloadable source code from http://wpebook.codeplex.com/ which contains a full application demonstrating all the examples.



Making a Phone call

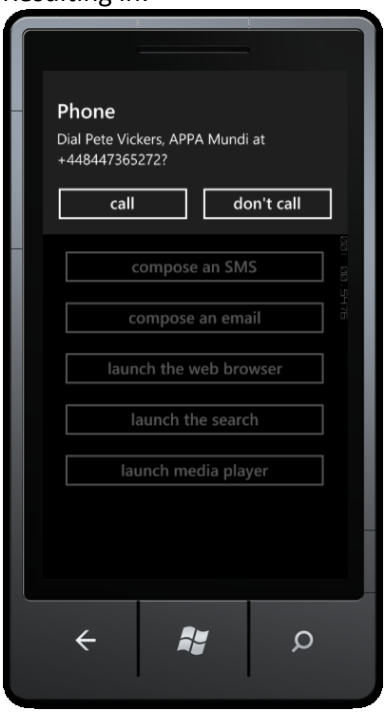
There can be many reasons you would want to make a phone call from within your application. For example, in 'About' page in your application, you can include a support number, and then when the

user taps on the number, it places a support call automatically. We have developed a 'Traffic' application, which includes an option to report traffic incidents to the relevant hotline. This is done by simply clicking on the number.

Declare the phone call as a task, set the properties, and then call Show().

```
private void btnMakePhoneCall_Click(object sender, RoutedEventArgs e)
    {
      PhoneCallTask task = new PhoneCallTask();
      task.PhoneNumber = "+448447365272";
      task.DisplayName = "Pete Vickers, APPA Mundi";
      task.Show();
    }
```

Resulting in:



At this point, you can obviously place the call, or cancel it.

Compose an SMS

Being able to send an SMS from your application can be very useful. In our 'Traffic' application, traffic incidents are received from an RSS feed and displayed in a list box to the user. The user can tap on a particular incident to get more details. At this point, the user has the option of sending details of the traffic incident by SMS to friends or colleagues who may be in the area.

Declare the SMS as a task, set the properties, and then call Show().

```
private void btnComposeSMS_Click(object sender, RoutedEventArgs e)
    {
      SmsComposeTask task = new SmsComposeTask();
      task.To = "+448447365272";
      task.Body = "This is a sample SMS message.";
      task.Show();
```

```
}
```

This time, the result shows the number the SMS is being sent to, and the SMS text, so you can amend it before you send.

At this point, you can amend and send the SMS, click on the paper clip to attach a picture, or simply press the 'back' button to cancel.

Compose an Email

As noted earlier, our traffic application can send an SMS to notify your friends and colleagues about a particular traffic incident. From the same screen, there is an option to email details of the traffic incident. The application uses the title of the incident as the subject, and the details of the traffic incident as the body of the email. The email task is then launched, and the user can select contacts to send the email to. Details like this can make all the difference to the usability of your application, and can differentiate it from other applications in the Marketplace.

Declare the email as a task, set the properties, and then call Show(). The minimum properties you should set are the 'to', 'subject' and 'body'.

```
private void btnComposeEmail_Click(object sender, RoutedEventArgs e)
    {
       EmailComposeTask task = new EmailComposeTask();
       task.To = "contact@appamundi.com";
       task.Cc = "pete@appamundi.com";
       task.Subject = "Testing email launcher";
       task.Body = "This is a sample e-mail message created by our test application";
       task.Show();
    }
```

This code will need to run on a device, as mail is not set up on the emulator. If you have more than one mail account set up, you will be prompted to choose the account, and then you will see the email message exactly as if you had created it using Outlook. At this point, you can add to the message, add an attachment, or simply send the email.

Launch The Web Browser

Providing links to web pages from your application, either directly in the code, or from a clickable link, can be useful for giving your users more information, or providing more help and guidance. The RSS Traffic feed in our application includes links to the Highways Agency website. The incident details include a clickable URL which links directly to a map of the incident.

Declare the browser as a task, set the properties, and then call Show().

```
private void btnLaunchBrowser_Click(object sender, RoutedEventArgs e)
    {
      WebBrowserTask task = new WebBrowserTask();
      task.URL = "http://www.appamundi.com";
      task.Show();
    }
```

Launch The Search

The 'Search' uses the Bing web searcher, and can be really useful in guiding your users. In an upcoming application, the user needs to enter certain cost details, which will vary from country to country. The addition of a Bing search icon containing the search critera gives the user useful information on the costs they are looking for, automatically.

Declare the search as a task, set the properties, and then call Show().

```
private void btnLaunchSearch_Click(object sender, RoutedEventArgs e)
    {
      SearchTask task = new SearchTask();
      task.SearchQuery = "APPA Mundi";
      task.Show();
    }
```

The results are displayed exactly as if you had entered a query into Bing, and the results are 'clickable' to drill through to the website.

Launching The Media Player

MediaPlayerLauncher launches the Media Player application and plays the specified media file. You can play media files that are stored in isolated storage, in the application's installation directory or out on the web. As with the other launchers, declare the launcher as a task, set the properties, and then call Show(). In the first example, a video on Windows Phone Development from Channel 9 will be shown, and in the second example, an MP3 file deployed with the project will be played.

```
private void btnLaunchMediaplayer_Click(object sender, RoutedEventArgs e)
    {
      MediaPlayerLauncher task = new MediaPlayerLauncher();
      task.Media = new Uri("http://ecn.channel9.msdn.com/o9/ch9/6f27/2b943705-cc7c-
46e0-adc3-9de901876f27/WP7Session1_2MB_ch9.wmv");
      task.Show();
    }
```

The above code goes out to the web, and plays a WP7 video from Microsoft Channel 9. MediaPlayerLauncher also lets you play media located on your device, but they must be copied down as data with the install (as in our sample), or copied into Isolated Storage. The location is controlled by the ‘location’ property of the task. The following code plays ‘test.mp3’ which is installed with the XAP of the sample code.

```
private void btnLaunchMP3_Click(object sender, RoutedEventArgs e)
    {
      MediaPlayerLauncher task = new MediaPlayerLauncher();
      task.Location = MediaLocationType.Install;
      task.Media = new Uri("Music/test.mp3",UriKind.Relative);
      task.Show();
    }
```

Choosers

Choosers work very much like launchers, in as much as you declare the task, set properties, and do a Show(). The difference comes in where the chooser returns a result to the user, and the result needs to be handled. Choosing and taking photographs both return results, and it is these that the first examples will demonstrate.

You need to beware of a couple of 'gotchas' when using choosers. When accessing certain choosers, if the device is docked, a completed event will be raised, and a task result of 'Cancel' is returned, as if the user has pressed 'cancel'. To get around this, we can check if the phone is docked by checking the state of the network connection. If the network connection returns 'Ethernet', the phone is docked.

To check this, write a function to return if the phone is docked:

```
private Boolean  Ethernet_Connected()
    {
       if (!(Microsoft.Phone.Net.NetworkInformation.NetworkInterface.NetworkInterfaceTy
pe == Microsoft.Phone.Net.NetworkInformation.NetworkInterfaceType.Ethernet))
       {
          return false;
       }
       else
       {
          return true;
       }
    }
```

Before carrying out certain tasks, call this function, and if it returns 'true', inform the user that they can't carry out the task whilst the device is connected.

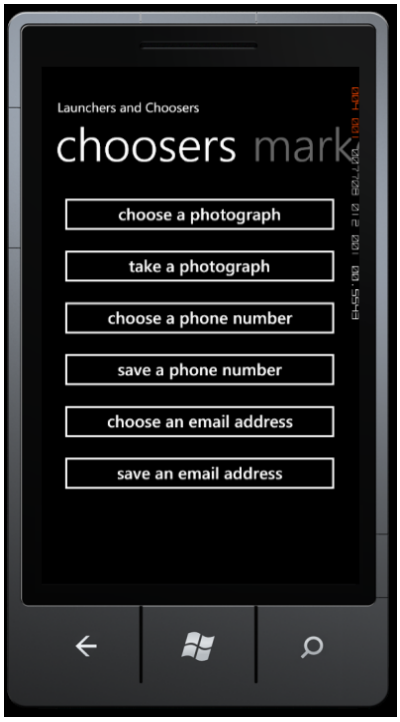
Beware of the 'back' button – it behaves as it should in most of the launchers and choosers, but in the sample, if you go to save a phone number or contact, and hit the 'back' button, you will notice it goes back to the first pivot item, not the pivot item it came from. So pay special attention to tombstoning for these two functions. Finally, declare the 'choosers' at a page level, for consistency of handling the 'back' button.

The table below (courtesy of Peter Foot) summarises the 'gotchas':

Task name	Emulator	Phone (docked)
CameraCaptureTask	Yes (simulated data)	No (completed returns TaskResult.Cancel)
EmailAddressChooserTask	Yes	Yes
EmailComposeTask	Shows error message (no account)	Yes
MarketplaceDetailTask	Exception 80070057	No
MarketplaceHubTask	Exception 80070057	No
MarketplaceReviewTask	Exception 80070057	No
MarketplaceSearchTask	Exception 80070057	No
PhoneCallTask	Yes	Yes
PhoneNumberChooserTask	Yes	Yes
PhotoChooserTask	Yes (sample data)	No (completed returns TaskResult.Cancel)
SaveEmailAddressTask	Yes	Yes
SavePhoneNumberTask	Yes	Yes
SearchTask	Yes	Yes
SmsComposeTask	Yes	Yes
WebBrowserTask	Yes	Yes

The Marketplace tasks perform no action and don't have a Completed event, so you don't get any feedback. You can provide similar logic to the above to warn the user that they can't use Marketplace functionality when docked; you could also hide any Marketplace links from your UI, based on the network state.

Only the PhotoChooserTask sample includes this test, but you should cater for it in your application, based on the table above.



Choosing a Photograph

In a forthcoming application, the user enters details about their children. The data collected is name, date of birth etc., but the user can also attach a photograph of each child, or take a photograph of their child. The application uses the PhotoChooser or CameraCapture to achieve this. If the user decides to take a photograph, it is saved in isolated storage.

Declare the photograph chooser as a task, and set the properties.

```
private void btnChoosePhoto_Click(object sender, RoutedEventArgs e)       {
        if (Ethernet_Connected())
        {
        PhotoChooserTask photoChooserTask;
        photoChooserTask = new PhotoChooserTask();
        photoChooserTask.Completed += new EventHandler<PhotoResult>(photoChooserTas
k_Completed);
        photoChooserTask.Show();
        }
        else
        {
           MessageBox.Show("To view photographs -
 please disconnect the camera","Camera Connected",MessageBoxButton.OK);
        }
     }
```

Additionally, create an event handler for the result, and then do a Show(). The ‘standard’ choose picture screen is then displayed, allowing a choice of pictures taken by the camera, and pictures on the device. Click on the picture you wish to select, then the ‘PhotoChooserTask_Completed’ event declared fires, and the picture is displayed in your program.


```
void photoChooserTask_Completed(object sender, PhotoResult e)
    {
      if (e.TaskResult == TaskResult.OK)
      {
        BitmapImage ChosenImage;
        ChosenImage = new System.Windows.Media.Imaging.BitmapImage();
        ChosenImage.SetSource(e.ChosenPhoto);
        imgPicture.Source = ChosenImage;
      }
    }
```

The code first checks that the user has indeed selected a picture. If they have, declare a BitmapImage and then set it to the chosen photograph returned, in e.ChosenPhoto. Finally, set the image source to the bitmap created from the photograph.

Taking a Photograph

Taking a photograph is very similar to choosing a photograph. Declare the task, set up an event handler, and then Show() the task.

```
private void btnTakePhoto_Click(object sender, RoutedEventArgs e)
    {
      CameraCaptureTask ctask;
      ctask = new CameraCaptureTask();
      ctask.Completed += new EventHandler<PhotoResult>(ctask_Completed);
      ctask.Show();
    }
```

This is almost identical to choosing a photograph, only in this case, the camera is started and you can take a picture. Click the button to take a picture, and the picture is returned to the code. (The emulator allows a 'picture' to be taken, but shows a white block to represent it.)

```
void ctask_Completed(object sender, PhotoResult e)
    {

      if (e.TaskResult == TaskResult.OK && e.ChosenPhoto != null)
      {

        CapturedImage = PictureDecoder.DecodeJpeg(e.ChosenPhoto);
        imgPicture.Source = CapturedImage;
        save_the_picture();
      }
      else
      {
        imgPicture.Source = null;
      }

    }
```

The event checks the result of the camera, and ensures the photograph returned is valid. If this is the case, in the sample above, the captured image is loaded into an image control on the form by first loading the photo taken into a WriteableBitmap, and then loading an image box on the form with the result.

It is likely that you will want to save your photo, and possibly upload it to SkyDrive or similar. The routine 'save_the_picture' does this, and sizes the image as it does so.

```
void save_the_picture()
    {
      if (CapturedImage != null)
      {
        //Create filename for JPEG in isolated storage
        String tempJPEG = "myphoto.jpg";

        //Create virtual store and file stream. Check for duplicate tempJPEG files.
        var myStore = IsolatedStorageFile.GetUserStoreForApplication();
        if (myStore.FileExists(tempJPEG))
        {
          myStore.DeleteFile(tempJPEG);
        }
        IsolatedStorageFileStream myFileStream = myStore.CreateFile(tempJPEG);

        //Encode the WriteableBitmap into JPEG stream and place into isolated storage.
        System.Windows.Media.Imaging.Extensions.SaveJpeg(CapturedImage, myFileStrea
m, 400, 240, 0, 100);
```

```
            myFileStream.Close();

        }
    }
```

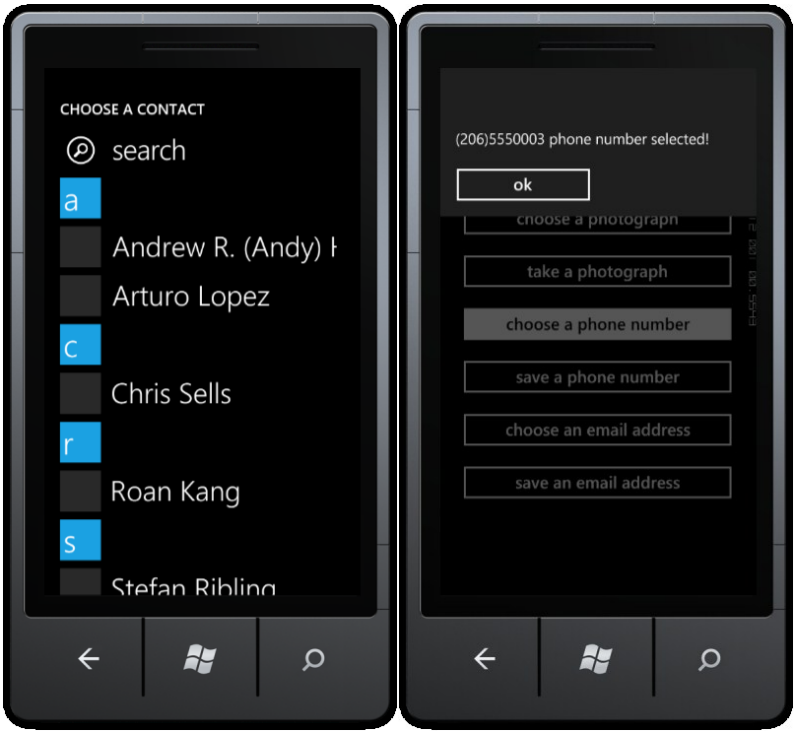
The above code saves the photograph taken in isolated storage.

Choosing a Phone Number

Choosing a phone number from an application can be useful. As noted earlier, our 'traffic' app can send SMS or email details of an incident. It would be easy to implement choosing a phone number and placing a call to a friend or colleague to let them know about the traffic incident, using the choosers and launchers. We chose not to implement this feature because we couldn't guarantee the user had a 'hands-free' system in their car. As this is a UK-only app, and making a call without a hands-free unit is illegal, we decided to err on the side of caution.

Choosing a phone number follows the consistent path already shown by the first two choosers: declare the task, set up an event handler, and then Show() the task. The only slight variance here is that the event is actually in the task.

```
private void btnChoosePhone_Click(object sender, RoutedEventArgs e)
    {
        PhoneNumberChooserTask task = new PhoneNumberChooserTask();
        task.Completed += (s, evt) =>
        {
            if (evt.Error == null && evt.TaskResult == TaskResult.OK)
            {
                MessageBox.Show(evt.PhoneNumber + " phone number selected!");
            }
        };
        task.Show();
    }
```



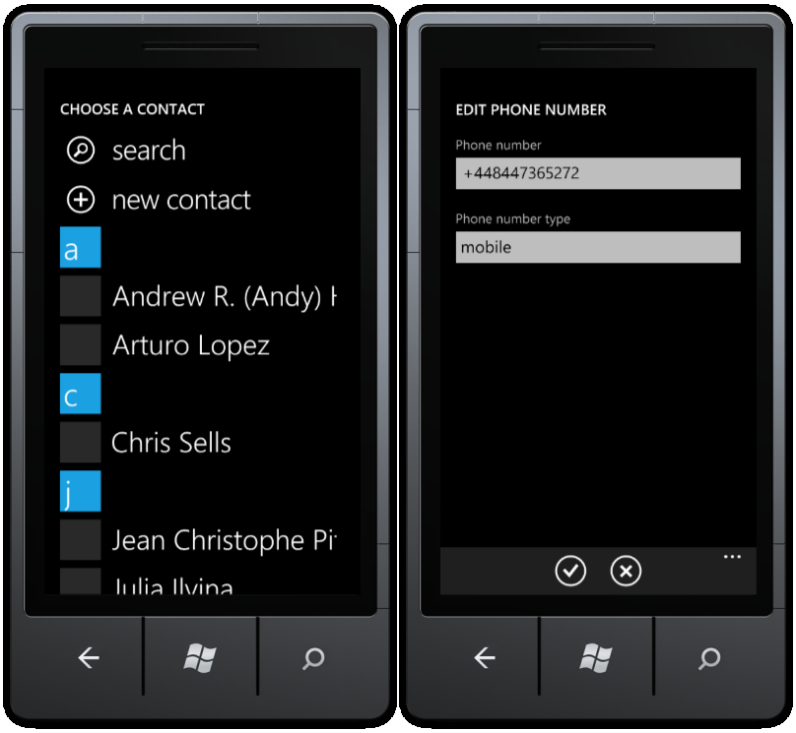
Choose ‘Chris Sells’ and his phone number is displayed.

Saving a Phone Number

Saving a phone number again follows the now familiar pattern: supply the phone number you wish to save, and call ‘Show()’.

```
private void btnSavePhone_Click(object sender, RoutedEventArgs e)
    {
      SavePhoneNumberTask task = new SavePhoneNumberTask();
      task.PhoneNumber = "+448447365272";
      task.Show();
    }
```

You are shown your contacts, and can choose to save the phone number as a new contact, or save to an existing contact, as shown below.



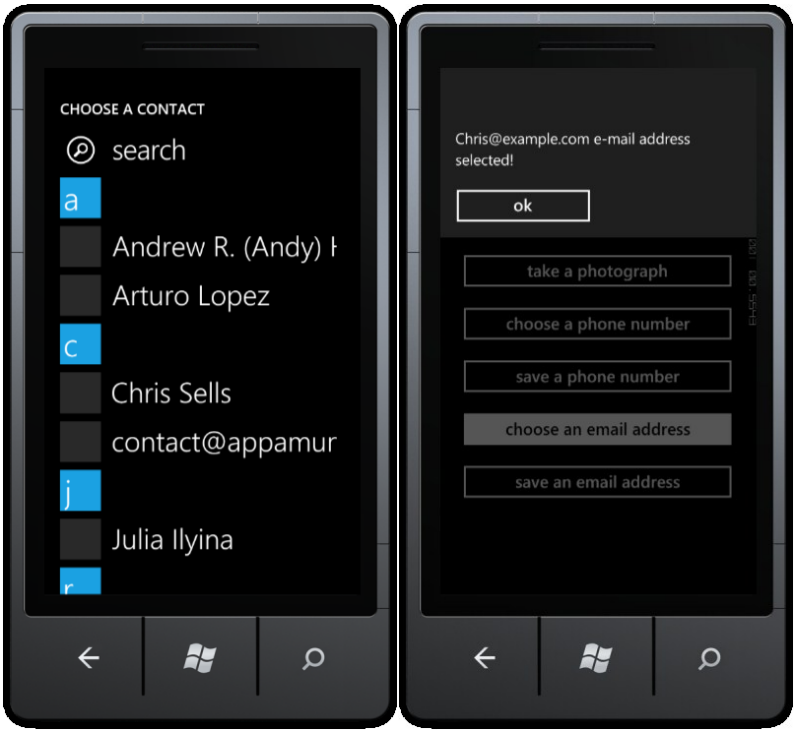
In this case, a new contact is chosen, and you are prompted to save the phone number. You can then continue entering other details for your new contact.

Choosing an Email Address

An example of using email in application was covered earlier in composing an email, where the traffic application can send an email of traffic incident details. In hindsight, a combination of a chooser and a launcher would have worked better, allowing the user first to choose who to send the email to, and then compose the email. An example of this is shown later, but now we will show how to choose an email address.

Choosing an email address is almost identical to choosing a phone number. Again, declare the task, set up an event handler, and then Show() the task.

```
private void btnChooseEmail_Click(object sender, RoutedEventArgs e)
    {
        EmailAddressChooserTask task = new EmailAddressChooserTask();
        task.Completed += (s, evt) =>
        {
            if (evt.Error == null && evt.TaskResult == TaskResult.OK)
            {
                MessageBox.Show(evt.Email + " e-mail address selected!");
            }
        };
        task.Show();
    }
```



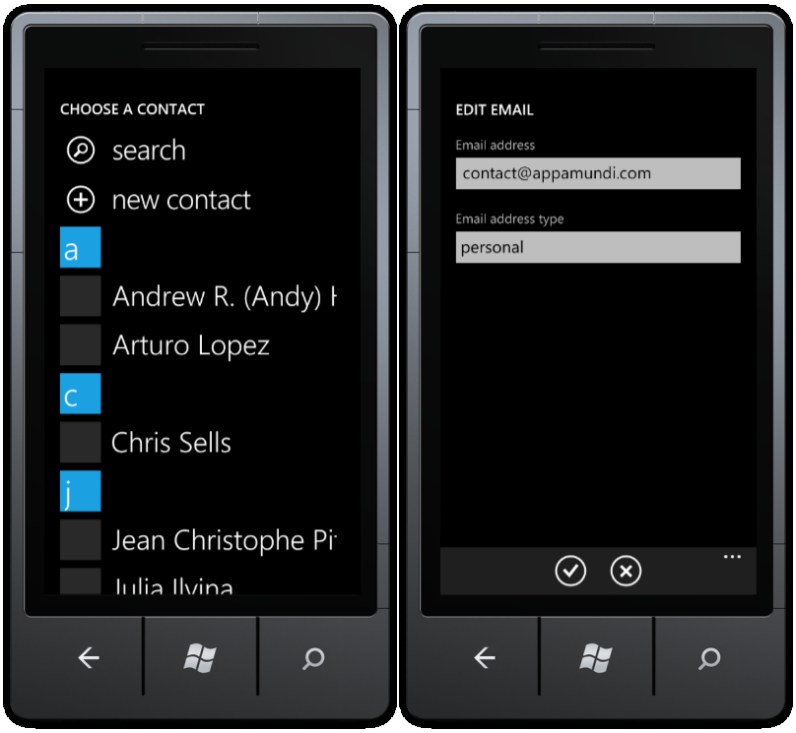
Again Chris Sells is chosen, and his email address is displayed.

Saving an Email Address

Saving an email address follows the now familiar pattern, and is almost identical to saving a phone number: supply the email address you wish to save, and call 'Show()'.

```
private void btnSaveEmail_Click(object sender, RoutedEventArgs e)
    {
      SaveEmailAddressTask task = new SaveEmailAddressTask();
      task.Email = "contact@appamundi.com";
      task.Show();
    }
```

You are shown your contacts, and can choose to save the email address as a new contact, or save to an existing contact, as shown below:



In this case, a new contact is chosen, and you are prompted to save the email address. You can then continue entering other details for your new contact.

Putting Launchers and Choosers together

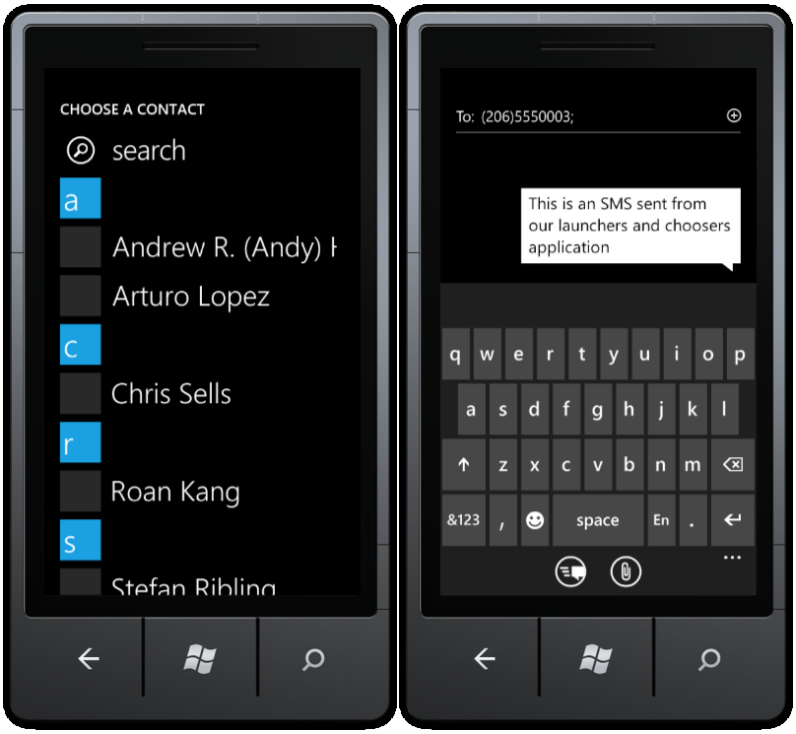
Launchers and choosers can be put together to complete quite complex tasks very simply. The next two examples will show how to choose a contact, and send them an SMS, and choose a contact and send an email.

Sending an SMS Message

To choose a contact and send an SMS is simply a case of combining the 'Choose a phone number', and 'Compose an SMS' from earlier in the chapter.

```
private void btnSendSMS_Click(object sender, RoutedEventArgs e)
    {
      PhoneNumberChooserTask contactsTask = new PhoneNumberChooserTask();
      contactsTask.Completed += (s, evt) =>
      {
        if (evt.Error == null && evt.TaskResult == TaskResult.OK)
        {
          SmsComposeTask smsTask = new SmsComposeTask();
          smsTask.Body = "This is an SMS sent from our launchers and choosers application
";
          smsTask.To = evt.PhoneNumber;
          smsTask.Show();
        }
      };
      contactsTask.Show();
    }
```

Firstly, the contacts screen is shown and, yet again, Chris Sells is chosen from the list of contacts. As soon as the contact has been chosen, the SMS screen is shown, pre-populated with the chosen telephone number and the text of the SMS message we are sending. The cursor is at the end of the text, allowing entry of any additional text.



At this point, you can send the SMS, or choose an attachment and then send it.

Sending an Email

To choose a contact and send an email is simply a case of combining the 'Choosing an email address, and 'Compose an email' from earlier in the chapter.

```
private void btnSendEmail_Click(object sender, RoutedEventArgs e)
    {
      EmailAddressChooserTask contactsTask = new EmailAddressChooserTask();
      contactsTask.Completed += (s, evt) =>
      {
        if (evt.Error == null && evt.TaskResult == TaskResult.OK)
        {
          EmailComposeTask emailTask = new EmailComposeTask();
          emailTask.Body = "This is a new email message from our launchers and choosers
application.";
emailTask.Subject = "Aren't choosers and launchers great!!";
          emailTask.To = evt.Email;
          emailTask.Show();
        }
      };
      contactsTask.Show();
    }
```

As soon as you choose your contact from the contacts screen, if you have more than one mail account set up, you will be prompted to choose the account, and then you will see the email message exactly as if you had created it using Outlook. At this point, you can add to the message, add an attachment, or simply send the email. Again, this code needs to run on a device, as Outlook is not set up on the emulator.

Marketplace Functions

There are four Marketplace launchers, which allow you to look at details of an application, go to the hub (either to Marketplace or to music), review an application or view an application.

Marketplace functions follow the now familiar path you have seen throughout the chapter: set a task, optionally set options and then Show() the task.

'MarketplaceDetails' expects an application id, and will show you details of the app as held in Marketplace.

```
private void btnDetails_Click(object sender, RoutedEventArgs e)
    {
      MarketplaceDetailTask marketplaceDetailTask = new MarketplaceDetailTask();
      marketplaceDetailTask.ContentIdentifier = "55fcf880-6b03-e011-9264-
00237de2db9e";
      marketplaceDetailTask.ContentType = MarketplaceContentType.Applications;
      marketplaceDetailTask.Show();
    }
```

'MarketplaceHub' takes you directly to the 'hub', where you can browse, search and generally work within the hub. The setting 'ContentType' determines which part of the hub you are directed to: 'Applications' or 'Music'.

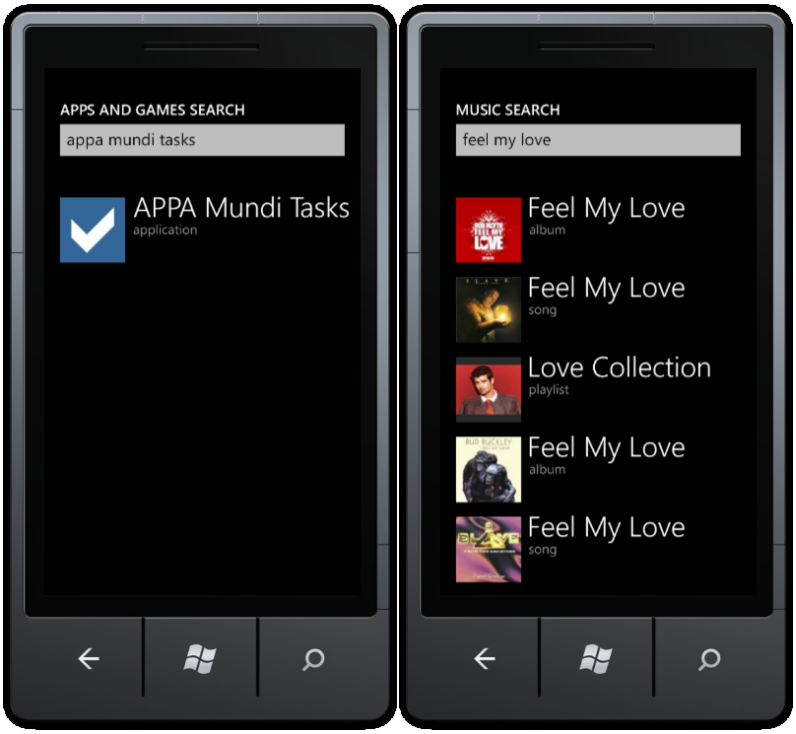
'MarketplaceReview' allows you to review an application.

```
private void btnReview_Click(object sender, RoutedEventArgs e)
    {
      MarketplaceReviewTask marketplaceReviewTask = new MarketplaceReviewTask();
      marketplaceReviewTask.Show();
    }
```

The final Marketplace function is 'MarketplaceSearch'. Simply supply your search string to the task, and do a Show(). Again, what is supplied as 'ContentType' determines what is searched. 'ContentType' can be set to 'Applications' or 'Music', and will take 'Applications' as the default. This differs from the standard Marketplace functions on the phone, which by default will search both music *and* applications.

```
private void btnSearch_Click(object sender, RoutedEventArgs e)
    {
      //Search for an application, using the default ContentType.
      MarketplaceSearchTask marketplaceSearchTask = new MarketplaceSearchTask();
      marketplaceSearchTask.SearchTerms = "appa mundi tasks";
      marketplaceSearchTask.Show();
    }

    private void btnMusicSearch_Click(object sender, RoutedEventArgs e)
    {
      //Search for a music item.
      MarketplaceSearchTask marketplaceSearchTask = new MarketplaceSearchTask();
      marketplaceSearchTask.ContentType = MarketplaceContentType.Music;
      marketplaceSearchTask.SearchTerms = "feel my love";
```

```
    marketplaceSearchTask.Show();
}
```



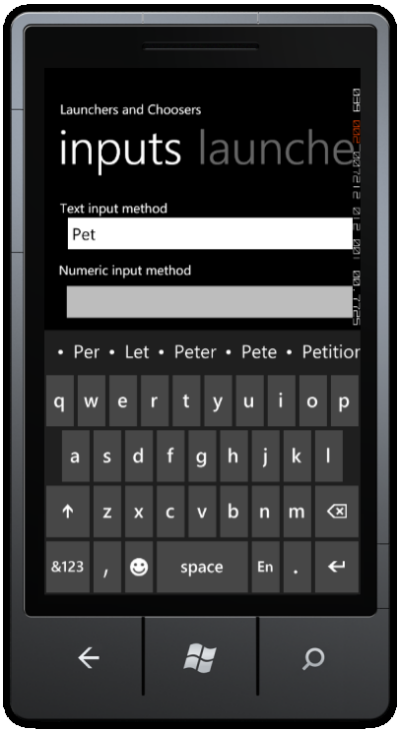
Input Features

Unlike its predecessor, keyboard input in Windows Phone 7 is a joy to use, and is infinitely more flexible. The key to using this new-found flexibility in your textboxes is the InputScope keyword. The following examples will show the most common input scopes. Every example follows the same layout. When you declare your textbox, amend the XAML, and add InputScope=”Text” for example, on the end of the declaration.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="12,23,0,0" Name="txtDefault" T
ext="" VerticalAlignment="Top" Width="460" InputScope="Text" />
```

Text

Declaring an inputscope of ‘Text’ will invoke ‘suggestion’, and after the first two letters, will start suggesting words, as below.



Number

Declaring an inputscope of 'Number' will display the full numeric keyboard.

URL

Declaring an inputscope of 'URL' will add the domain to the keyboard, and a 'go' arrow at the bottom right of the keyboard. You need to insert the code to detect if the arrow has been pressed.

First of all, in the XAML, you need to code a 'KeyDown' event.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,227,0,0" Name="txtURL" Tex
t="" VerticalAlignment="Top" Width="460" InputScope="Url" KeyDown="Url_KeyDown"/>
```

Next, you need to code the 'KeyDown' event in the xaml.cs, to capture the arrow and action it.

```
private void Url_KeyDown(object sender, System.Windows.Input.KeyEventArgs e)
    {
      if (e.Key == Key.Enter)
      {
        // Do search...
      }
    }
```

Tap and hold on '.com' to pop up more domains. The domains change according to the keyboard language (see below).

Search

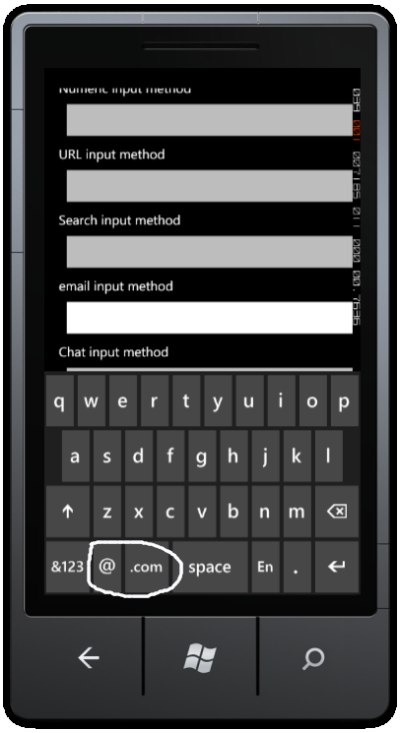
'Search' is similar to 'URL', and you need to code the 'KeyDown' event in the same way. The 'Search' keyboard differs from the 'URL' keyboard in as much as the domain key does not appear.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,327,0,0" Name="txtSearch" T
ext="" VerticalAlignment="Top" Width="460" InputScope="Search" KeyDown="Search_KeyD
own"/>
```

Email

The ‘email’ keyboard is similar to the text keyboard, with the addition of a domain key and an ‘@’ key. The ‘email’ keyboard does not suggest words.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,427,0,0" Name="txtEmail" Te
xt="" VerticalAlignment="Top" Width="460" InputScope="EmailNameOrAddress" />
```



Chat

The ‘Chat’ keyboard is suitable for composing SMS or Twitter messages. The keyboard contains a ‘smiley’ key, which allows many of the common smiley characters to be input.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,527,0,0" Name="txtChat" Te
xt="" VerticalAlignment="Top" Width="460" InputScope="Chat" />
```

Telephone Number

The ‘Telephone number’ keyboard brings up the same keyboard as the ‘phone’ icon from the front screen.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,627,0,0" Name="txtPassword
" Text="" VerticalAlignment="Top" Width="460" InputScope="TelephoneNumber" />
```

Currency and Symbol

This option shows a numeric keyboard and a currency symbol. Tap and hold on the currency symbol to get a list of alternate currencies. The XAML can be coded differently as we have seen earlier.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="10,727,0,0" Name="txtcurrency
" Text="" VerticalAlignment="Top" Width="460">
                <TextBox.InputScope>
                  <InputScope>
                    <InputScopeName NameValue="CurrencyAmountAndSymbol" />
                  </InputScope>
                </TextBox.InputScope>
              </TextBox>
```

Alternate Keyboards

Every keyboard (except the 'telephone number') shown in the previous examples has a 'language' key. Tap and hold this key to get a list of languages, and tap on the language to change to it.

Summary

This chapter has shown the ease with which it possible to create relatively complex tasks, such as choosing a contact, composing and sending an email. That operation can be achieved using just 13 lines of code!

Launchers and choosers can greatly enhance the usability and functionality of your application with minimal effort.

Using InputScope to match your input text boxes can also enhance the user experience when using your application.

Interlude: Interactive sci-fi movies

Alastair Maggs splits his time between programming and film editing. This combination led to Timedancer from Chaos Created. It's an episodic, interactive sci-fi television series available on Windows Phone. It combines compelling video with puzzle-solving and exploration.

"We're big fans of Lost and we wanted to explore a story like that," says Maggs. In Episode 1, you can follow two routes through the story, making decisions along the way. In Episode 2, there are even more choices and a big cliffhanger. The filming had a full cast and a crew with ten people – it's a big production!

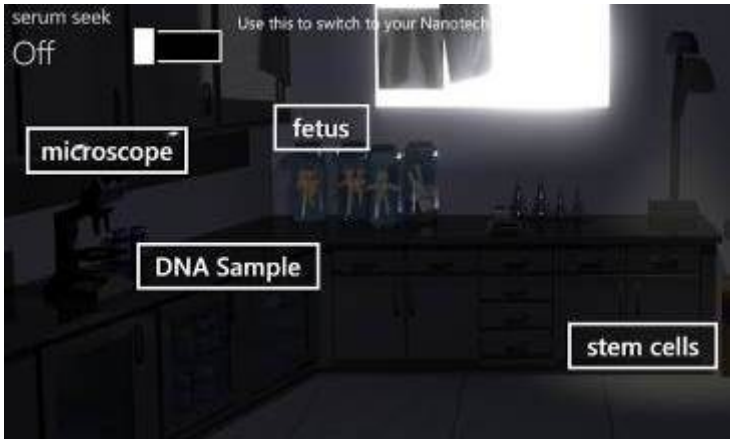
The team initially wanted to do an iPhone version but they struggled with Apple's restrictions, particularly around screen rotation. It's pretty hard to have a widescreen interactive cinematic experience in portrait mode. Android was even more problematic because of the wide variety of

screen sizes and device capabilities. So when Microsoft approached Chaos Created about a Windows Phone version, “it was perfect for what we were doing,” says Maggs.

The bulk of the programming took about a month but with tweaking and editing after that. The filming was much more demanding. Maggs had already worked in C# and a bit of web-based Silverlight so developing for Windows Phone was an easy switch. A big benefit is that “every single phone is going to have the same baseline functionality.” Another is the ability to develop a single codebase for Windows Phone, Xbox, Web and PC. “Anyone getting into Windows Phone development should aim to program once and run on many different platforms.”

One of the things that Maggs really appreciates is the human contact with the Microsoft team. “You can email people in the Windows Phone division in Microsoft and they’ll get back to you.”

Now the team is doing pre-production for Episode 3 and they’re looking at Microsoft’s ad platform to allow them to release free but ad-supported episodes in future. And the plot? Don’t ask. They won’t tell. “It’s going to be exciting but we’re not giving anything away!”



Location Aware Applications and Mapping

By Andy Gore, Mike Hole

The Location Service

Location services within Windows Phone 7 provide a rich API, which allows applications to identify the handsets location by responding to a series of specific events, i.e. change of device location. It is worth noting that location services is not just GPS-based tracking, the service can also determine a device's location from wireless networks, cell towers etc.

Getting Started

All the location-based services reside in the System.Device.Location namespace, which can be found within the System.Device assembly.

The main class that developers will utilise for location services is the GeoCoordinateWatcher class. It is this class that is responsible for determining current position, movement to a new location etc.

The following code shows you how to initialise the GeoCoordinateWatcher, as well as the key events that you would typically subscribe to.

```
private GeoCoordinateWatcher watcher;
public GeoTestClass()
{
  watcher = new GeoCoordinateWatcher(GeoPositionAccuracy.High);
  watcher.MovementThreshold = 20;
  watcher.StatusChanged += new
EventHandler<GeoPositionStatusChangedEventArgs>(watcher_StatusChanged);
  watcher.PositionChanged += new
EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher_PositionChanged)
;
  watcher.Start();
}
```

The first thing to notice is that the GeoCoordinateWatcher's constructor takes in a GeoPositionAccuracy enumeration. As the name suggests, this value dictates the accuracy of the location data provided. The level of accuracy selected also influences the mechanism by which the geo data is obtained. Lower accuracy will use cell towers or Wi-Fi to obtain the data, whereas high accuracy will use the device's on-board GPS, although this comes at a cost of increased battery consumption.

The MovementThreshold property dictates the distance in metres that should be reached before a PositionChanged event occurs. In most cases, you'd probably never want to set the threshold to be

too low, as the value in retrieving the data this frequently would only suit a minority of applications. In fact, the MSDN documentation recommends a minimum threshold of 20 metres.

The StatusChanged event fires when the status of the GeoCoordinateWatcher changes. The GeoCoordinateWatcher can be in one of four statuses at any one time:

- Initialising – Location services are being initialised i.e. trying to get a location fix.
- Ready – Location services now in a state to provide data.
- NoData – No data available. This could be that the GeoCoordinateWatcher has yet to be started.
- Disabled – Occurs when location services have been disabled on the device. In this instance, you could also query the GeoCoordinateWatcher's Permission property to see if the user has actually disabled the location services functionality.

The PositionChanged event, as the name suggests, is raised when the device registers a change of location that exceeds the predefined MovementThreshold. It is worth noting that, not only the latitude and longitude of the device is returned, but also altitude and speed amongst other values.

The following code fragment shows a simple usage of the PositionChanged event. This code simply displays the current latitude and longitude within a text box.

Note that the events raised by the GeoCoordinateWatcher don't return on the UI thread, so you'll need to ensure that any events are dispatched there first before modifying the UI.

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

private void GeoPositionChanged(GeoPositionChangedEventArgs<GeoCoordinate> e)
{
  if (watcher.Status == GeoPositionStatus.Ready)
  {
    txtPosition.Text = String.Format("Current latitude : {0}, current longitude: {1}",
e.Position.Location.Latitude, e.Position.Location.Longitude);
   }
}
```

Application Certification Requirements

There are a few things that you need to consider when writing applications that use location services with respect to the application certification process:

- The user must be able to turn the location aware features on and off.
- The location aware features should be off when the user first launches the application.

- The user should explicitly grant the application permission to use location services.
- The user should be presented with information detailing how the captured location information is going to be used by your application.

Using the Bing Maps Control

Windows Phone 7 comes complete with a Silverlight Bing Maps Control, which allows developers to easily incorporate rich mapping functionality into their applications. The Map Control provides a variety of ways to visualise map data, from aerial and street views with varying zoom levels, Pushpins to plot-specific points of interest/information, amongst a multitude of other functionalities.

Where do I start?

Before you start developing any applications that take advantage of Bing Maps, you'll need to sign up for a Bing Maps account at https://www.bingmapsportal.com/. Once this is done, you'll need to create a key for your application. This key becomes your authentication credential for the mapping functionality.

Now that you have your key available, you're ready to add a map to your application. To do this, you simply need to reference the Microsoft.Phone.Controls.Maps assembly and then reference the relevant namespace within your page's XAML declaration as follows:

```
xmlns:map="clr-
namespace:Microsoft.Phone.Controls.Maps;assembly=Microsoft.Phone.Controls.Maps"
```

Once the map namespace is referenced, you can now add a map to your XAML mark-up as follows:

```
<map:Map
Name="JoggingMap"
CredentialsProvider="Your Bing Maps Key"
    Mode="Aerial"
    ZoomLevel="16"
    Center="53.32058,-2.73706"
    >
</map:Map>
```

This produces the following map:

Note: in the above XAML, the Bing Map credentials are entered as plain text within the mark-up. In a production application, you should take care to protect your Bing Maps credentials, and never store them as plain text anywhere within your app.

Controlling the map

Controlling the map is straightforward. It can be easily manipulated via XAML mark-up or code behind. In fact, the previous XAML code snippet achieved quite a lot with a small amount of mark-up. It set the map to display in aerial mode, got it to centre on a location based upon its longitude and latitude, and finally zoomed the map. If we wanted to see a slightly more zoomed-in street view of the map, we'd only have to change the XAML to the following:

```
<map:Map
Name="JoggingMap"
CredentialsProvider="Your Bing Maps Key"
    Mode="Road"
    ZoomLevel="18"
    Center="53.32058,-2.73706"
    >
</map:Map>
```

This would then produce the map below:

Pushpins

Pushpins within Bing Maps provide us with a mechanism to mark locations on a map. A very simplistic example could be that you want to highlight your current location on a map.

This is achieved with the code below:

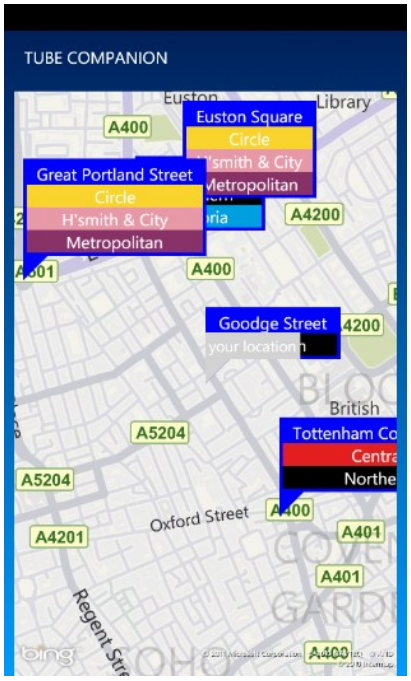
```
Pushpin myLocation = new Pushpin();
myLocation.Content = "I'm here!";
myLocation.Location = new GeoCoordinate(53.32058,-2.73706);
MyMap.Children.Add(myLocation);
```

Essentially, an instance of a Pushpin is created, its geo-coordinates are set, and the Pushpin is added to the map. One thing to notice from the above code is that the Pushpin class is a Content control, so as with any other Silverlight control, you can add pretty much anything you like to the Pushpin.

In one application I've been involved with, the application needed to show Pushpins of all of the nearby tube stations on the London Underground, with each pin detailing the lines that run from the station. In this instance, I simply created a user control which rendered a list of tube lines and added this to each Pushpin, for example:

```
foreach (TubeStation station in stations)
{
  Pushpin stationPin = new Pushpin();
  stationPin.Location = new GeoCoordinate()
{ Latitude = station.Latitude,
Longitude = station.Longitude };
  stationPin.Background = new SolidColorBrush(Colors.Blue);
  stationPin.Foreground = new SolidColorBrush(Colors.White);
  stationPin.Content = new TubeStationPushPin()
{ TubeStation = station };
  map.Children.Add(stationPin);
 }
```

Creating the following:



Polylines

Whereby Pushpins give us the ability to mark specific locations on a map, Polylines allow us to draw lines between geo-coordinates. This is particularly useful for any route planning/tracking-based applications.

The following code creates a series of lines between each of the geo-coordinates specified in the MapPolyline's Locations collection.

```
MapPolyline routePolyLine = new MapPolyline();
routePolyLine.Stroke = new SolidColorBrush(Colors.Yellow);
routePolyLine.StrokeThickness = 3;
routePolyLine.Opacity = 0.7;
routePolyLine.Locations = new LocationCollection();
MyMap.Children.Add(routePolyLine);
```

Simply adding another geo-coordinate item to the Locations collection will automatically plot a line from the last coordinate to the newly added coordinate, giving the following:

Using the Bing Web Services

There are three services to the Bing Maps REST service:

- Locations API – Find locations based upon address, location or query.
- Imagery API – Provide images based upon location, area etc.
- Routes API – Plot routes between locations.

All of the services stem from the one base URL:

http://dev.virtualearth.net/REST/v1/

The next part of the request URI lets the services know which service is required (Locations, Imagery or Routes), with each service then having a specific URI structure. In this book, we won't detail how to make all of the different permutations of the requests to the services; to find out more you can visit the Bing Maps REST Services section of the MSDN Library.

Making a call to get data

The simplest way to get data from a remote server is to make use of the WebClient class. This class will take care of making the request to the server and processing the response.

It is recommended, however, that within a production environment the WebClient class is not used as it runs on the UI thread. This could cause your application to be unresponsive while processing requests.

The alternative method is to use the HttpWebRequest class. This has more control over how the request is made and is capable of running on a background thread.

For more information regarding obtaining data via the HttpWebRequest class, see the 'Networking' section of this book.

Here is an example of using the WebClient class to get a route from the Eiffel Tower to the Louvre museum:

```
string req = "http://dev.virtualearth.net/REST/V1/Routes/Walking?wp.0=Eiffel%20Tower&w
p.1=louvre%20museum&optmz=distance&output=xml&key=BingMapsKey";

WebClient xmlClient = new WebClient();

xmlClient.DownloadStringCompleted += new DownloadStringCompletedEventHandler(xmlCli
ent_DownloadStringCompleted);

xmlClient.DownloadStringAsync(
        new Uri(req, UriKind.Absolute));
```

The request is made by the client when DownloadStringAsync is called. This will raise the xmlClient_DownloadStringCompleted event when the result of the call is available. These details can be found in the DownloadStringCompletedEventArgs augment.

Here is how the event handler might look:

```
void xmlClient_DownloadStringCompleted(object sender, DownloadStringCompletedEventA
rgs e)
{
   if (e.Error == null)
   {
      XElement route = XElement.Parse(e.Result);

      ProcessRoute(route);
   }
}
```

As you can see, the event checks for an error and, if one hasn't occurred, loads the data into an XElement object ready for processing.

Location Service

There are three different ways in which you can query the location service: find a location by address, by point or via a query. This data can be used to identify location information (for example, you may have the postcode but no address information) and also provides geographical information relating to the area that has been identified. All forms of query return the same location data format.

All of the location service calls start with the following URI structure:

http://dev.virtualearth.net/REST/v1/Locations/

By Address:

The 'by address' query is one that you can perform if you only have address information but you need to know more about that address, such as its geo-location.

The address information that you provide can vary in detail but, the more information that is provided in the query, the more accurate the results you will get.

There are two ways of calling the address-based lookup services, the first using a structured approach where the lookup forms the URI. A UK-based example being:

http://dev.virtualearth.net/REST/v1/Locations/UK/Fitzalan%20Place/Cardiff/?output=xml&key=BingMapsKey

http://dev.virtualearth.net/REST/v1/Locations/UK/CF24 0EL?output=xml&key=BingMapsKey

The second method uses an un-structured approach whereby the address is provided within the query string. Again, here is an example:

http://dev.virtualearth.net/REST/v1/Locations?locality=London&postalCode=SW1A&key=BingMapsKey

By Point:

The 'by point' query is one that you can perform if you have the latitude and longitude of the location you wish to query. With this query, you can specify the types of entity that are returned in the results.

The by point query takes the following form:

http://dev.virtualearth.net/REST/v1/Locations/point?includeEntityTypes=entityTypes&key=BingMapsKey

The lookup values returned by the by point request can be refined by supplying a list of values into the optional include EntityTypes parameter of the request. These values include: Neighborhood, PopulatedPlace, Postcode1, AdminDivision1, AdminDivision2, CountryRegion. A combination of these values can be provided by separating the values with commas.

To specify a point, the latitude and longitude values of the point need to be provided. These are given as two decimal numbers separated by a comma.

Here is an example of the by point query:

http://dev.virtualearth.net/REST/v1/Locations/51.4464,-3.1662?o=xml&key=BingMapsKey

By Query

As with the by address lookup requests, the 'by query' requests can be in a structured or un-structured form:

http://dev.virtualearth.net/REST/v1/Locations/query?key=BingMapsKey

http://dev.virtualearth.net/REST/v1/Locations?query=query&key=BingMapsKey

The query can contain a full address or parts of an address. Obviously the more granular the data you provide for the location the better the results you will have returned.

An example would be a search for 'Cardiff'. This actually provides four different results, those being two for the UK (not sure why), one in the USA and one in New Zealand. If the query is 'UK Cardiff', it will return two results, and a query for UK Cardiff CF64 0EL will return a single address.

Here is how the requests can look for the above:

http://dev.virtualearth.net/REST/v1/Locations?query=cardiff&o=xml&key=BingMapsKey

http://dev.virtualearth.net/REST/v1/Locations/UK Cardiff?o=xml&key=BingMapsKey

http://dev.virtualearth.net/REST/v1/Locations?query=UK Cardiff CF64 0EL&o=xml&key=BingMapsKey

Results

Here is an example of a result set:

```
<?xml version="1.0" encoding="utf-8"?>
<Response xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.micro
soft.com/search/local/ws/rest/v1">
  <Copyright>Copyright © 2011 Microsoft and its suppliers. All rights reserved. This API cann
ot be accessed and the content and any results may not be used, reproduced or transmitted
in any manner without express written permission from Microsoft Corporation.</Copyright>
  <BrandLogoUri>http://dev.virtualearth.net/Branding/logo_powered_by.png</BrandLogoUr
i>
  <StatusCode>200</StatusCode>
  <StatusDescription>OK</StatusDescription>
  <AuthenticationResultCode>ValidCredentials</AuthenticationResultCode>
  <TraceId>90dafdd3a0aa461f9e89a873917e4f83|LTSM001158|02.00.82.2800|LTSMSNVM0
01472, LTSMSNVM001853</TraceId>
  <ResourceSets>
    <ResourceSet>
      <EstimatedTotal>1</EstimatedTotal>
      <Resources>
        <Location>
          <Name>Street, Penarth CF64 1</Name>
          <Point>
            <Latitude>51.4453911781311</Latitude>
            <Longitude>-3.1682091951370239</Longitude>
          </Point>
          <BoundingBox>
            <SouthLatitude>51.441528460560427</SouthLatitude>
            <WestLongitude>-3.176471962432013</WestLongitude>
            <NorthLatitude>51.44925389570178</NorthLatitude>
            <EastLongitude>-3.1599464278420348</EastLongitude>
          </BoundingBox>
          <EntityType>Address</EntityType>
          <Address>
            <AddressLine>Street</AddressLine>
            <AdminDistrict>Wales</AdminDistrict>
            <AdminDistrict2>Vale of Glamorgan</AdminDistrict2>
            <CountryRegion>United Kingdom</CountryRegion>
            <FormattedAddress>Street, Penarth CF64 1</FormattedAddress>
            <Locality>Penarth</Locality>
            <PostalCode>CF64 1</PostalCode>
          </Address>
          <Confidence>Medium</Confidence>
        </Location>
      </Resources>
    </ResourceSet>
  </ResourceSets>
</Response>
```

As you can see, the response includes a status code and description. As with most REST-based services, the status code matches the standard HTML status codes. The description helps supplement this.

The results returned by the service are provided in a ResourceSet collection. Within each ResourceSet is a collection of Location items. You can use LINQ to XML to navigate through the XML tree or simply use the XElement.Decendants() method to access the Locations collection.

Before you do this, you will need to declare the service namespace:

```
XNamespace xmlns =
 "http://schemas.microsoft.com/search/local/ws/rest/v1";
```

Now you can iterate the locations collection and, for example, add each Location node to a model object:

```
foreach (XElement locationData in route.Descendants(xmlns + "Location"))
{
   locationItems.Add(new LocationItem(locationData));
}
```

Here we have a LocationItem object that takes an XElement in the constructor which contains the location data. The rest of the LocationItem would implement the properties required by your application. For example, if you wanted to expose the location name:

Define the property in the LocationItem class:

```
public string Name { get; private set; }
```

Assign the property value in the constructor:

```
Name = data.Element(xmlns + "Name").Value;
```

To store the associated Point you can use the GeoCoordinate class from the System.Device.Location namespace:

Define the property in the LocationItem class:

```
public GeoCoordinate LocationPoint { get; private set; }
```

Assign the property value in the constructor:

```
LocationPoint = new GeoCoordinate(
   double.Parse(data.Element(xmlns + "Point").Element(xmlns + "Latitude").Value),
   double.Parse(data.Element(xmlns + "Point").Element(xmlns + "Longitude").Value));
```

You would create properties for all the values required within your application.

Routes

All of the route service calls start with the following URI structure:

http://dev.virtualearth.net/REST/v1/Routes

The routes service allows you to specify walking, transit or driving route types. This is specified as the final path segment of the request URI. All other parameters are specified as query string parameters.

An example route request would be:

http://dev.virtualearth.net/REST/V1/Routes/Walking?wp.0=London%20Paddington%20Station&wp.1=London%20Marble%20Arch&optmz=distance&output=xml&key=BingMapsKey

As you can see, this is a walking route from London Paddington railway station to Marble Arch. If you add the Bing Maps key that you are using and put this into the address bar of any browser, you will see the data that it returns.

As with the Location service, the route service returns a Resources collection containing the route(s) that has (have) been returned:

```
<ResourceSets>
        <ResourceSet>
                <EstimatedTotal>1</EstimatedTotal>
                <Resources>
                        <Route>
```

The route returned for this example would take up many pages so it won't be reproduced here.

The route information includes useful summary details including the bounding box of the route, along with the distance and approximate travel time.

The route itself is split into route legs, of which there will be one or more depending on the number of waypoints that are specified (three waypoints will provide two legs, and four will provide three etc.).

Each route leg contains the points that specify the start and end of the leg, and the distance and approximate time of that leg. The leg contains a set of itinerary items. These items provide instructions to the person following the route to help them reach their destination. Here is an example itinerary item:

```
<ItineraryItem>
        <TravelMode>Walking</TravelMode>
        <TravelDistance>0.095</TravelDistance>
        <TravelDuration>68</TravelDuration>
        <ManeuverPoint>
                <Latitude>51.517869159579277</Latitude>
                <Longitude>-0.17700769007205963</Longitude>
        </ManeuverPoint>
        <Instruction maneuverType="DepartStart">Depart from London Paddington station,
United Kingdom</Instruction>
        <CompassDirection>southeast</CompassDirection>
        <Detail>
                <ManeuverType>DepartStart</ManeuverType>
```

```
                <StartPathIndex>0</StartPathIndex>
                <EndPathIndex>1</EndPathIndex>
                <CompassDegrees>124</CompassDegrees>
                <Mode>Walking</Mode>
                <PreviousEntityId>0</PreviousEntityId>
                <NextEntityId>0</NextEntityId>
                <RoadType>Street</RoadType>
            </Detail>
            <Exit />
            <TollZone />
            <TransitTerminus />
            <IconType>Walk</IconType>
            <Time>0001-01-01T00:00:00</Time>
            <TransitStopId>0</TransitStopId>
            <TowardsRoadName>South Wharf Road</TowardsRoadName>
            <SideOfStreet>Unknown</SideOfStreet>
        </ItineraryItem>
```

As you can see, this information is quite detailed. Using the same technique as described in the 'Location Service' section, it is possible to turn this information into a model class that can be used to render a set of instructions to the user.

One thing that needs to be pointed out is that the itinerary items cannot be used to render the route on a map. This is because these items are points where a manoeuvre is made (such as a change from one road to another). The actual path that a road takes is not specified, so if you plot a line between each point it will not follow the road.

If you wish to render the route on a map, you need to specify the routePathOutput type in the request for data. In this case, there is only one value, 'Points'. So adding routePathOutput=Points to the request will provide points that can be used to construct a polyline on a map.

If you have specified that the results are to include the path, a list of points is provided within the Route element this is called the RoutePath:

```
<ResourceSets>
    <ResourceSet>
        <Resources>
            <Route>
                <RoutePath>
```

An example of the content is as follows:

```
<RoutePath>
    <Line>
        <Point>
            <Latitude>51.517869</Latitude>
            <Longitude>-0.177007</Longitude>
        </Point>
        <Point>
```

```
                    <Latitude>51.517230999999995</Latitude>
                    <Longitude>-0.176081</Longitude>
                </Point>
                ...
            </Line>
        </RoutePath>
```

Handily, there is a class for dealing with this kind of data; the LocationCollection class (part of the Microsoft.Phone.Controls.Maps namespace). The following code populates the location collection:

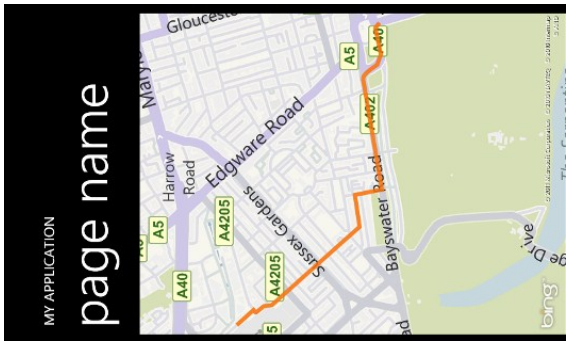
If we add a map to a page with a polyline as such:

```
<my:Map CredentialsProvider="BingMapsKey" >
<my:MapPolyline x:Name="plRoute"
                Stroke="#FFF5720D"
        Opacity="0.85"
        StrokeThickness="6" />
</my:Map>
```

We can make use of the following code to generate the LocationCollection and assign the locations to the polyline:

```
LocationCollection pointsCollection = new LocationCollection();

foreach (XElement pointData in returnData
   .Element(xmlns + "ResourceSets" )
   .Element(xmlns + "ResourceSet")
   .Element(xmlns + "Resources")
   .Element(xmlns + "Route")
   .Element(xmlns + "RoutePath")
   .Element(xmlns + "Line")
   .Elements(xmlns + "Point"))
{
   pointsCollection.Add( new GeoCoordinate(
      double.Parse(pointData.Element(xmlns + "Latitude").Value),
      double.Parse(pointData.Element(xmlns + "Longitude").Value)
      ));
}

plRoute.Locations = pointsCollection;
```

The example given above generates the following line:



Imagery Service

The imagery service is quite different to the other services as it returns images rather than data.

There are times when using the imagery service within a Windows Phone application (rather than using the Bing Maps Control) provides a more efficient method of showing the location of a point.

An example of this might be showing a list of places that includes a small map indicating the location of each place.

Images retrieved from the imagery service can have all sorts of data overlaid, such as traffic information etc. These are specified within the request URI. Because of this, we will only cover the essentials with regards to image retrieval.

All of the imagery service calls start with the following URI structure:

```
http://dev.virtualearth.net/REST/v1/Imagery/Map/
```

The first parameter the request takes is the imagery set parameter. This is where you specify the desired image type. The values for the imagery set are:

- Aerial – Aerial imagery.
- AerialWithLabels – Aerial imagery with a road overlay.
- Road – Roads without additional imagery.

An example imagery service call could be:

http://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial/wales%20millennium%20centre,cardiff?key=BingMapsKey

This will return the following image:

This image is the default 350x350 pixel size. Because the request is based upon a query, the image also includes a Pushpin showing the location of the item that has been found.

It might be easier to use the actual location of the centre point of the image:

http://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial/51.4650,-3.1630/17?mapSize=200,200&key=*BingMapsKey*

As you can see, this URI now includes the latitude and longitude of the centre point (51.4650-3.1630). The next parameter is the zoom level (17). This value can be between 1 and 22. This URI also specifies the size of the image that is to be returned (200x200 pixels). Here is the resulting image:

It is possible to use this image in your application with the BitmapImage class.

```
string imageReq = "http://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial/51.4650,-3.1630/17?mapSize=400,400&key=Ai7InFW4WnbOS9QK56YWS5X_BwWxQ6akZYICjQqJaPC22cqlLnMTGxSPtWN-W6NF";

BitmapImage mapImage = new BitmapImage(new Uri(imageReq));

LocationImage.Source = mapImage;
```

If you are using a pattern such as MVVM (Model View View Model), you will probably want to make the bitmap image a property of the model:

```
private BitmapImage _mapImage;
```

```
public BitmapImage MapImage
{
   get
   {
      if (_mapImage == null)
      {
         string imageReq =
string.Format("http://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial/{0},{1}/17?mapSize
=400,400&key=Ai7InFW4WnbOS9QK56YWS5X_BwWxQ6akZYICjQqJaPC22cqlLnMTGxSPtWN
-W6NF",
            LocationPoint.Latitude, LocationPoint.Longitude);

         _mapImage = new BitmapImage(new Uri(imageReq, UriKind.Absolute));
      }

      return _mapImage;
   }
}
```

Storing Data in SQL Server

Although your application is capable of storing its own data within isolated storage, it may be that you want this data to be shared in some way with other users.

Sharing data between users requires a storage mechanism on an external server. SQL Server 2005 and above have the ability to store, index and query spatial data. This is especially helpful if your application is going to provide users with a list of items within a specific area.

Using the geo data types to store locations

There are different spatial column types that SQL Server supports but this section will cover geography column type.

When storing data within a table it is recommended that you also store the latitude and longitude of the location in separate columns, along with the geography column. This will make it easier to present the data via a service such as WCF Data Services as the Entity Framework does not support the geography field types.

Here is a basic create table script for a table that stores location information:

```
CREATE TABLE [dbo].[GeoLocations](
      [GeoLocationID] [uniqueidentifier] NOT NULL,
      [TrailID] [uniqueidentifier] NOT NULL,
      [TrailOrder] [int] NULL,
      [Altitude] [float] NULL,
      [Course] [float] NULL,
      [Longitude] [float] NOT NULL,
      [Latitude] [float] NOT NULL,
      [GeoPoint] [geography] NULL)
```

As you can see, this table is designed to store trails that have been recorded by a user in a phone application.

Most ORMs don't support the geography data type so you cannot directly insert data into the GeoPoint field via the data context.

One way around this is by post processing the GeoLocation row after it has been added to the database. Let's assume that the row has been created so we have the ID of the location row and its original latitude and longitude.

We can update the GeoPoint column by using the SQL function geography::STGeomFromText as follows:

```
SqlCommand cmd = connection.CreateCommand();

cmd.CommandType = System.Data.CommandType.Text;

cmd.CommandText = string.Format("UPDATE GeoLocations SET GeoPoint =
geography::STGeomFromText('POINT ({0} {1})', 4326) WHERE GeoLocationID = '{2}' ",
newLocation.Longitude, newLocation.Latitude, newLocation.GeoLocationID);

cmd.ExecuteNonQuery();
```

The 4326 parameter that is part of the STGeomFromText function is the special reference identifier code 4326 being world coordinates.

Please note that this is a very rough use of the SqlCommand object that does not use parameters as it may be used in a function that is passed integers and objects only. You should add parameters to the SqlCommand if the data is less likely to be less strongly typed.

Location-based queries

Once the geography data has been populated, it is possible to perform spatial queries.

The geography type has many functions. One of the most useful is STDistance. This function calculates the distance between the location and a given point. Its use within SQL looks like this:

```
SELECT * FROM GeoLocations WHERE
[GeoPoint].STDistance(geography::STGeomFromText('POINT (51.4650 3.1630)', 4326)) <
1000
```

This will select all points that are less than 1000 metres from the given point.

As with insert / update, functions that are used to perform location-based queries cannot be performed by an ORM such as Entity Framework. Direct commands or Stored Procedures will be needed to perform these queries.

Sample Application Walkthrough

Using some of the features explained in this chapter, you'll see how to build a basic application easily which records the route taken by a runner. Essentially, the application will present a Bing Maps Control, along with two buttons, one to start recording the runner's route, the other to stop recording it. Once a user has started recording their run, the map will be updated to show the user's current position.

Note: this is not production code; it's purely to indicate functionality.

Generating the XAML for the run recording application

First, create a new Windows Phone Portrait Page.

Add a project reference to the Microsoft.Phone.Controls.Maps assembly.

Now add the following namespace declaration to the PhoneApplicationPage:

```
xmlns:map="clr-
namespace:Microsoft.Phone.Controls.Maps;assembly=Microsoft.Phone.Controls.Maps"
```

Next, paste the following XAML into the Grid element with the name ContentPanel:

```
<Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <map:Map
      Name="RunningMap"
      CredentialsProvider="Insert your Bing Map Account details here"
      Mode="Road"
      ZoomLevel="17"
      Center="53.32058,-2.73706"
      >
    </map:Map>
    <StackPanel Grid.Row="1" Orientation="Horizontal">
      <Button Name="StartRunning" Content="Start" Click="StartRunning_Click"/>
      <Button Name="StopRunning" Content="Stop" Click="StopRunning_Click"/>
    </StackPanel>
```

This XAML will construct a user interface containing the Bing Maps Control, and the Start and Stop buttons. The Bing Maps Control in this instance has been set to Road mode, and initialised to centre at a set GeoCoordinate. Although this isn't particularly necessary at this stage, it'll at least give you a clear indication of whether the control has been referenced correctly, and whether your Bing Maps credentials are valid. Note: don't build the project at this stage as you've yet to define the event handlers that the buttons use.

Generating the code behind to track the runner's route

Now open MainPage.xaml.cs and add the following instance level variables and property:

```
private GeoCoordinateWatcher geoWatcher;
```

```
private MapPolyline runnersPolyLine;
private List<GeoCoordinate> runnersCoordinates;
private bool CurrentlyRunning {get;set;}
```

These variables define the GeoCoordinateWatcher, which provide us with the location data, a MapPolyline which will be added to the map to show the runner's route and a list of GeoCoordinates which record the route taken by the user.

Next, paste in the following code:

```
public MainPage()
{
 InitializeComponent();
 geoWatcher = new  GeoCoordinateWatcher(GeoPositionAccuracy.High);
 geoWatcher.PositionChanged += new
EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher_PositionChanged)
;
 CurrentlyRunning = false;
 SetupRunnersPolyline();
 SwitchRunningButtons(CurrentlyRunning);
}

private void SetupRunnersPolyline()
 {
 runnersPolyLine = new MapPolyline();
 runnersPolyLine.Stroke = new SolidColorBrush(Colors.Blue);
 runnersPolyLine.StrokeThickness = 5;
 runnersPolyLine.Opacity = 0.7;
 runnersPolyLine.Locations = new LocationCollection();
 RunningMap.Children.Add(runnersPolyLine);
 }
```

The constructor just creates an instance of the GeoCoordinateWatcher class, subscribing to its PositionChanged event. The constructor then calls additional initialisation logic.

The SetupRunnersPolyline method creates a new MapPolyline, sets its colour and thickness and adds it as a child control of the Bing Map. It is worth noting that the MapPolyline has a locations collection. This collection stores a list of coordinates. Whenever a location is added to the collection, a line is drawn from the previous location in the collection to the newly added location.

Now paste in the following code:

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

private void GeoPositionChanged(GeoPositionChangedEventArgs<GeoCoordinate> e)
```

```
{
 if (CurrentlyRunning)
 {
        runnersCoordinates.Add(e.Position.Location);
runnersPolyLine.Locations.Add(new GeoCoordinate(e.Position.Location.Latitude,
e.Position.Location.Longitude));
RunningMap.Center = new GeoCoordinate(e.Position.Location.Latitude,
e.Position.Location.Longitude);

  }
}
```

Watcher_PositionChanged *is simply the method that handles the GeoCoordinateWatcher's* PositionChanged *event. In this case, it dispatches the coordinate details to the GeoPositionChanged method which runs on the UI thread.*

Remember that the events raised from GeoCoordinateWatcher don't return on the UI thread.

The GeoPositionChanged method is responsible for adding the newly acquired location to an internal collection, as well as to the locations collection on the MapPolyline that we created earlier. Essentially, whenever this method is executed, the end-user will see the route of their run so far.

Now paste the following code:

```
private void StartRunning_Click(object sender, RoutedEventArgs e)
{
 StartRunningSession();
}

private void StartRunningSession()
{
 runnersCoordinates = new List<GeoCoordinate>();
 CurrentlyRunning = true;
 SwitchRunningButtons(true);
 geoWatcher.Start();
}

private void StopRunning_Click(object sender, RoutedEventArgs e)
{
 geoWatcher.Stop();
 CurrentlyRunning = false;
 SwitchRunningButtons(false);
 CreatePushpins();
}

private void SwitchRunningButtons(bool currentlyRunning)
{
 StartRunning.IsEnabled = !currentlyRunning;
```

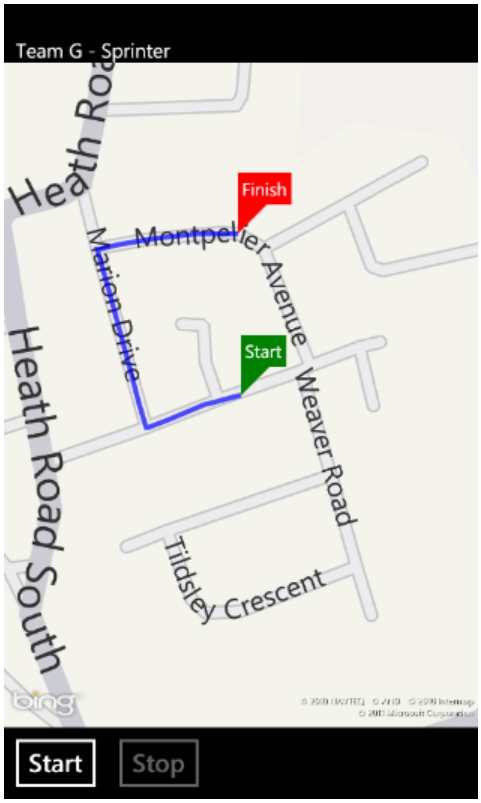
```
 StopRunning.IsEnabled = currentlyRunning;
}
```

This code essentially just starts and stops the GeoCoordinateWatcher, as well as updating some UI aspects of the application.

Finally, paste in the following code:

```
private void CreatePushpins()
{
Pushpin StartPin = CreatePushpin("Start", new   SolidColorBrush(Colors.Green),
runnersCoordinates.First());

Pushpin EndPin = CreatePushpin("Finish", new SolidColorBrush(Colors.Red),
runnersCoordinates.Last());

    RunningMap.Children.Add(StartPin);
    RunningMap.Children.Add(EndPin);

}

private Pushpin CreatePushpin(string pinText, Brush pinBackground, GeoCoordinate
pinCoordinate)
{
Pushpin pushpin = new Pushpin();
    pushpin.Location = pinCoordinate;
    pushpin.Content = pinText;
    pushpin.Background = pinBackground;
    return pushpin;
}
```

Upon stopping the run, besides actually stopping the GeoCoordinateWatcher, the CreatePushpins method is called. This method retrieves the first and last coordinates of the user's run from the internal collection, it then creates a Pushpin for each of these coordinates, with each Pushpin also detailing whether it is the start or end coordinate of the run.


Team G - Sprinter
Heath Road
Finish
Montpelier Avenue
Marion Drive
Start
Heath Road South
Weaver Road
Tildsley Crescent
bing
Start
Stop

Interlude: How much is that taxi journey?

Did you ever feel that you were getting ripped off by a taxi taking the long way to your destination? Were you ever stuck somewhere new and didn't know which taxi company to call? These are the problems that Trevor Daniel has solved with TaxiRoute. He has built a system that calculates the distance between two addresses and uses a database of local authority-approved fares to estimate the cost and suggest a local firm.

"People really love using it – in the last year, we've dealt with 240,816 enquiries worth nearly £12m," says Daniel. The original inspiration was a friend of his, who is an actual taxi driver. He started out with Vale of Glamorgan tariffs and compared the results with Geoff's actual mileage and charges – "It proved that the system was incredibly accurate." But the real job was compiling data for the other 360-plus local authorities in the UK.

When the new Windows Phone platform came out, Daniel saw an opportunity to offer a smartphone version, working with Mike Hole to develop a Windows Phone version. One neat feature is a one-touch 'get me home' button that uses phone features like geo-location to show the price, distance, time and closest cab firm to get you home. Just what you need on a Saturday night!

“The database is really valuable and unique – it’s my crown jewels,” says Daniel. Anyone can clone the app but they can’t clone the database and so they can’t offer accurate prices. The new app uses client-server architecture. The taxi fare database stays on the TaxiRoute.co.uk site and the phone client accesses it to get the latest figures. The use of a SOAP API for the backend and a Windows Phone client shows how easily Windows Phone apps can link back to corporate databases and backend systems.

Reactive Extensions for .NET

By Dominic Betts

Before you dive into the details of Reactive Extensions for .NET (Rx), it's useful to think a little about some of the characteristics of the environment you'll be developing for. Mobile phone users expect a very responsive and flexible user interface (UI); they never want to wait any significant period of time for the application to respond to user input, they always want to feel in control of what's going on and have the ability to switch to something else, and of course phone calls take priority over everything. At the same time, mobile applications are becoming increasingly sophisticated: using web and cloud services for data storage and processing. However, using the network from a phone application will not always give you reliable and consistently high-speed access to the services that your application uses. The classic response to this conflict between the need for a highly responsive UI and the risk of slow, intermittent network access is to do things asynchronously. This is the primary use case for Rx: managing asynchronous behaviour in your Windows Phone 7 (WP7) applications.

This chapter will introduce you to using Rx for handling asynchronous web service requests, but bear in mind that you can use Rx wherever you need asynchronous behaviour in your application.

WP7 applications in a Connected Environment

There are many reasons why your mobile application might need to use web services: accessing live travel information, storing data such as photographs collected on the phone, downloading detailed product information or authenticating with an identity provider. Whatever happens, you don't want your phone to freeze until the web service call completes. Therefore, you want to make sure that you make the call asynchronously.

The .NET Framework has a number of ways that you can manage asynchronous operations: look in the **System.Threading** and **System.Threading.Tasks** namespaces for plenty of examples. So why do you need yet another way? Rx has a number of features that make it attractive, especially on the WP7 platform:

- It's lightweight
- It has very concise syntax
- It makes extensive use of LINQ

Enumerable Sequences and Observable Sequences

When you learn something new, it often helps to relate it to something familiar. As a .NET developer, you probably use enumerable sequences all the time. For example, given a sequence like this:

```
int[] numbers = {1, 2, 3, 5, 8, 13, 21};
```

You could iterate over the enumerable sequence like this:

```
foreach (var number in numbers)
{
  Console.WriteLine("Number: {0}", number);
}
```

Rx uses the concept of observable sequences rather than enumerable sequences. With an enumerable sequence, you 'pull' each entry from the sequence when you're ready; with an observable sequence, it 'pushes' entries to you, whenever the sequence has new data.

If you rewrite this simple example using Rx, you will have something like this:

```
// Convert the enumerable sequence to an observable sequence
IObservable<int> observableNumbers = numbers.ToObservable();

// Subscribe to the sequence
observableNumbers.Subscribe(
  number => Console.WriteLine("Number: {0}", number),
  ex => Console.WriteLine("Something went wrong: {0}", ex.Message),
  () => Console.WriteLine("At the end of the sequence"));
```

Notice that you can have three separate blocks of code: one to handle each entry from the sequence, one to handle any errors that occur and one to handle the end of the sequence.

If you look at the Rx documentation, you'll see that the key interface is called **IObservable**.

The power of this approach becomes apparent when you consider that the items in the sequence that you subscribe to might be events in the UI, or messages arriving over the network: all of which you are now handling asynchronously.

Some Simple Examples

The next two examples illustrate how you could use Rx in other familiar contexts. These examples are not necessarily realistic, but they will help you understand how Rx works.

Filtering Items in an Observable Sequence

The first example is handling UI events: this illustrates another powerful feature of Rx, using LINQ syntax to filter the items in the observable sequence. The following code shows the standard way to handle events and shows how you might prevent a user from entering non-numeric characters in a textbox.

The textbox definition in XAML looks like this:

```
<TextBox Name="textBox1" KeyDown="textBox1_KeyDown" />
```

The event handler looks like this:

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
```

```
  if (e.Key < Key.NumPad0 || e.Key > Key.NumPad9)
  {
    e.Handled = true;
  }
}
```

The alternative, Rx-based approach works like this: use the **FromEvent** method to create an observable sequence from a standard event. Using LINQ syntax, you can filter the events to select only the invalid, non-alphanumeric keys:

```
var keys = from evt in
  Observable.FromEvent<KeyEventArgs>(textBox1, "KeyDown")
  where (evt.EventArgs.Key < Key.NumPad0 || evt.EventArgs.Key >
  Key.NumPad9)
  select evt.EventArgs;
```

You can subscribe to the observable sequence like this and handle the invalid keystokes:

```
// Ideally you should dispose of keySubscription
// when the Window is disposed.
var keysSubscription = keys.Subscribe(evt =>
{
  evt.Handled = true;
  label1.Content = "Invalid character: " + evt.Key.ToString();
});
```

Remember, this example is not supposed to be realistic; it's intended to illustrate some basic Rx syntax in a familiar context.

Background and UI Threads

The second example shows you how to use Rx to perform background tasks that may be needed to update the UI — again this isn't necessarily the approach you'd take, the examples are here to illustrate Rx syntax. The first code example shows a traditional approach in a desktop application that uses the **Task** class from the Task Parallel Library (note that the TPL is not available on the WP7 platform). Notice how this example uses the **Dispatcher.Invoke** method to update the UI from the worker thread.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
  Task task = new Task(doWork);
  task.Start();
}


delegate void ContentSetterDelegate(Label label, string text);

private void doWork()
```

```
{
  // Work really hard at something.
  Thread.Sleep(2000);
  this.Dispatcher.Invoke(
    new  ContentSetterDelegate(setContentProperty),
    label1, "Finished the work at last!");
}

private void setContentProperty(Label label, string text)
{
label.Content = text;
}

To achieve the same results with Rx you could use code like this:
var clicks = from evt in
  Observable.FromEvent<RoutedEventArgs>(button2, "Click")
  select evt;
var clicksSubscription = clicks.Subscribe(evt =>
{
  var backgroundTask = Observable.Start(() =>
  {
    // Work really hard at something.
    Thread.Sleep(3000);
  }).ObserveOnDispatcher();

  backgroundTask.Subscribe(
    _ => { },
    () => label1.Content = "It's all done now!"
  );
});
```

In this example, you first handle the button click event that's going to launch the background task. In the button click event handler, you define a new background task with the **Observable.Start** method. The **ObserveOnDispatcher** method ensures that your observer runs on the UI thread where it's safe to interact with controls in the UI. Nothing happens until we subscribe to the background task and, in this example, we're only interested in the result when the background task finishes: the **_ => { }** is shorthand for "I'm not interested in individual items in the sequence."

A Practical Example: Invoking a Web Service

The previous examples have been somewhat artificial, designed to show you the basics of Rx in the context of some familiar scenarios. The next example is more practical: it will show you how to invoke a web service asynchronously using Rx. It will include many of the concepts you've already seen, such as filters, asynchronous tasks and updating the UI safely.

In this example, the web service in question is a REST-style web service. The service endpoint is at *http://localhost/CustomerLists*, and the service returns a collection of **Customer** objects formatted as a JSON string. The example is split into two parts: first, an extension method to the

HttpWebRequest class to handle the web service call, and second a method to handle the results and update the UI.

Calling the REST-based Web Service

The following code sample shows the extension method that makes the asynchronous web service call:

```
public static IObservable<T> Get<T>(this HttpWebRequest request)
{
   request.Method = "GET";

   return
      Observable
         .FromAsyncPattern<WebResponse>(
           request.BeginGetResponse, request.EndGetResponse)()
         .Select(
            response =>
               {
                  using (var responseStream
                           = response.GetResponseStream())
                  {
                     var serializer =
                       new DataContractSerializer(typeof(Customer[]));
                     return (T)serializer.ReadObject(responseStream);
                  }
               });
}
```

So how does this work? The first thing to notice is that this method itself returns an observable sequence. In the method body, you can see the **FromAsyncPattern** method is used to instantiate an observable sequence. This method takes a pair of **BeginXXX** and **EndXXX** methods that are part of the standard .NET pattern for defining asynchronous behaviours and returns an observable sequence. The **Select** method enables you to take each item pushed at you from the sequence and project it into a different type. In this example, it converts the HTTP response to a collection of objects by de-serialising the JSON in the response payload.

Integrating with the UI

The second part of the sample shows how to invoke the extension method and use the results to update the UI:

```
public IObservable<Customer[]> GetCustomers()
{
   var serviceEnpoint = "http://localhost/CustomerLists";
   var customersServiceUri = new Uri(serviceEnpoint + "/all");

   return
       Observable.Return((HttpWebRequest)WebRequest.Create(
         customersServiceUri))
```

```
            .SelectMany(request => { return request.Get<Customer[]>(); },
                  (request, customers) => { return customers; })
            .ObserveOnDispatcher()
            .Catch((WebException ex) =>
                  {
                     var message = GetMessageForException(ex);
                     MessageBox.Show(message);
                     return Observable.Return(default(Customer[]));
                  });
}
```

This approach uses the **Return** method to create the web request that it invokes asynchronously by calling the **Get** extension method you saw in the previous section. However, this time you must use the **SelectMany** method instead of the **Select** method to handle the results. This is because the **Get** extension method already returns an observable sequence of **Customer** objects and using **Select** here would mean the **GetCustomers** method would return an observable sequence of observable sequences of customers. The **SelectMany** method 'flattens' an observable sequence of observable sequences into a simple observable sequence. You've already met the **ObserveOnDispatcher** method, and the **Catch** method catches any exceptions (as you'd expect).

The final piece of the puzzle is to see how the user invokes **GetCustomers** from the UI. The following sample shows a button click event handler and a method to update the UI:

```
private void UpdateCustomers(Customer[] customers)
{
   // Update the UI.
  …
}

private void button3_Click(object sender, RoutedEventArgs e)
{
   this.GetCustomers()
      .ObserveOnDispatcher()
      .Subscribe(this. UpdateCustomers);
}
```

Summary

In this chapter, you've seen a set of examples that illustrate some of the key features of Rx for .NET, including code that will make an asynchronous call to a web service and use the results to update the UI of your application. The **IObservable** interface is the key to understanding what you can achieve with Rx, but it always helps to keep in mind the idea that Rx uses a 'push' model as opposed to the more familiar 'pull' model of the **IEnumerable** interface.

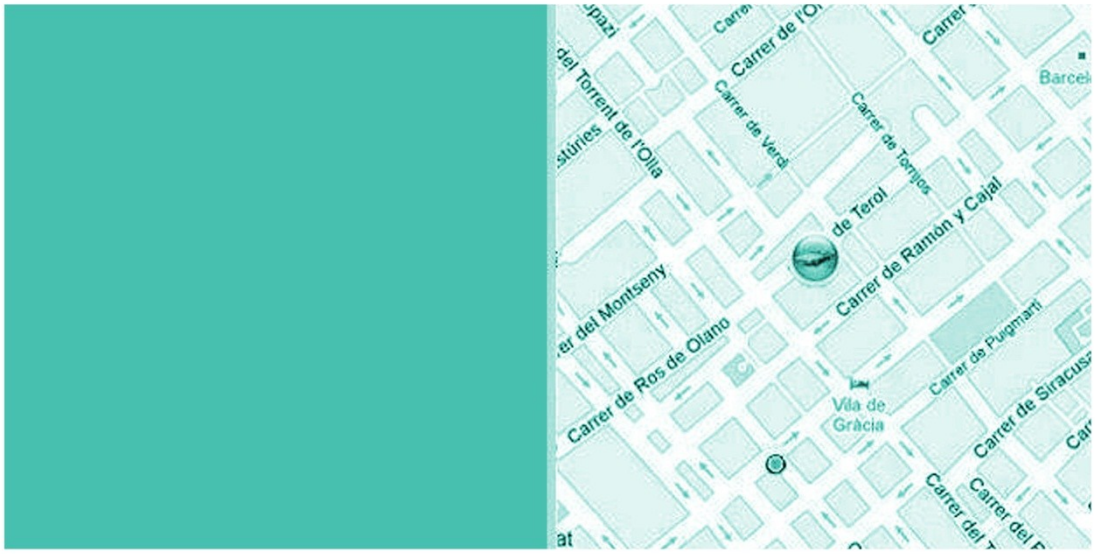
These resources will help you explore Rx in more depth:

Reactive Extensions for .NET Overview for Windows Phone:
http://msdn.microsoft.com/en-gb/library/ff431792(VS.92).aspx

Reactive Extensions:
http://msdn.microsoft.com/en-gb/data/gg577609

(not yet) 101 Rx Samples:
http://rxwiki.wikidot.com/101samples

Interlude: Dude, where's my car?



Every smartphone has to have a 'car finder' app and the team from Appamundi have created the signature Windows Phone version, Where's my car. We spoke to Appamundi's Andy Wigley about developing for the platform.

He is an experienced mobile app developer – he wrote The Microsoft Mobile Development Handbook and several Dot.net guides for Microsoft Press. As a result, he was one of the first Windows Phone 7 developers in the UK and the app was ready for the launch of the platform.

Moving from the old Windows Phone system to the new one required a big change in tools – there's obviously no backward compatibility. However, Wigley says "you still need to keep the mindset for developing phone applications: dealing with limited memory, small screens and patchy connectivity." Desktop Silverlight skills help, he says, but you have to be aggressive about making your code as lean as possible.

“Microsoft gives us a great set of tools to allow us to develop stunning looking apps and we have to put more emphasis on creating a really good, richer UI than we perhaps did before,” says Wigley. To help with this, he turned to a graphic designer to polish the look and feel of the app. Tools like Microsoft Expression Blend are a great help with this task.

Wigley and the Appamundi team took the skills they developed during the development of ‘Where’s my car?’ and applied to several more applications, including ‘APPA Mundi Tasks’. This brings Exchange tasks Tasks synchronization to Windows Phone. Wigley’s colleague Peter Foot implemented the code that allows the app to synchronise with Exchange Server directly from Windows Phone over the air in the same way as built-in apps such as Outlook Mail – no mean feat.

Although an on-phone is included in Windows Phone 7.5, the original system didn’t provide one. To get a phone-side database cache for apps with large data storage needs, Wigley ported an open source C# database to Windows Phone. As a result of this, the company had an enquiry from a large US company who wanted a Line of Business app for salesman which stored details of their 40,000 products and many thousands of customers on Phone and he was able to do it. The system is now live and in the field. Wigley says “This shows that Windows Phone is not just a platform for fun or productivity apps but also great for commercial applications for vertical markets or bespoke development”.

Marketplace – Designing for First Time Approval

By Sasha Kotlyar

When your application is completed, you can sell it on the Windows Phone Marketplace. Consumers can buy and download applications from a centralized location, and payments are automatically handled for you. To ensure that end-users of the Marketplace are provided with high-quality information and applications, Microsoft enforces a certification process that your application must pass to gain entry to the Marketplace.

One of the most frustrating experiences Windows Phone developers face is their app being rejected by the Marketplace tester team. Not only does it take up to five or more days for the rejection to occur, but in the meantime, developers are stuck waiting to see what will happen to the submission. In addition, if a bug or some leftover debug code is discovered after submitting the app, developers have to wait for the submission to be approved or rejected and can only then submit a corrected version of the app.

But fear not. There are ways to protect yourself from negative Marketplace submission experiences. The most important one, which I cannot overemphasize, is to read the Application Certification Requirements guide extremely carefully and to understand it properly.

You can view the requirements on the MSDN website:

http://msdn.microsoft.com/en-us/library/hh184843(v=VS.92).aspx

Skipping over sections that obviously do not apply to your app or that contain general information with which you are already familiar is fine. However, it's very important to read all the other parts of the document. Take the time to read the guide and to understand what its overall goals are.

If the aforementioned guide had to be summed up in one sentence, it would be: "Ensure that your app gives users a pleasant, smooth, and above all, predictable experience." The actual text of the entire guide simply explains Microsoft's vision for such an experience. If you take the time to understand it before you start writing your app, you will have very few, if any, issues to deal with when the time comes to submit it for certification.

Having said that, it must be noted that the Marketplace testers are human after all. They make mistakes, and even they can occasionally misinterpret some certification requirements. This is usually an issue only with edge-case scenarios, where an app has functionality that could possibly be construed as violating some rule in the guide. Luckily, Microsoft has anticipated such issues and, as a response, added a field in the submission process for tester notes. When this field isn't left empty, testers read it before performing any of their tests. I have never left this field empty for any new app or update that I have submitted, and I have (so far) had no rejections.

Once you understand Microsoft's vision for app experiences, you will likely come across a few scenarios in which execution of this vision becomes somewhat problematic. One such scenario deals with navigation and the hardware back button.

Back Button

Back button behaviour is one of the most typical failures for Marketplace submission; the Application Certification Requirements cover this in 5.2.4.

http://msdn.microsoft.com/en-us/library/hh184840(v=VS.92).aspx

In order for an app to feel predictable, it must perform a very obvious 'go back' action when the back button is pressed (or in the case of dialogs, it must correspond to a 'cancel' action).

First-party apps always have subtle animations to indicate this and, conversely, a 'go forward' action. Such animations are not required, but they almost always enhance the navigation experience of any app.

Developers often run into issues in the occasional case where it makes no sense to go back to a certain page after it has already been visited. An example of this is a login page. Once credentials have been provided, they shouldn't be requested again when the user is trying to go back, especially to back out of the current application into the previous one. The solution to such an issue is to convert the login page into something else entirely: a popup control. Popups do not pollute the system back stack, and they can optionally be animated for a more pleasing look. Additionally, popups can be shown with minimal effort on any page, so if you are developing an app that requires a high level of security, you could potentially show a login popup whenever the user returns to your tombstoned app after turning off the phone or navigating to another app. That cannot be easily done with a separate login page.

Tombstoning

Speaking of tombstoning, that's another consistently painful issue for many developers. Although saving and loading the app state is not required for certification, it is rarely a good idea to leave this functionality out of apps. When testing your application's state handling in the context of tombstoning, be sure to test what happens when the user hits the back button after your application resumes. The reason is that, although all of your app's back stack pages are saved, their states are not loaded until the user actually navigates back to them. In other words, your pages should be prepared to load their previously saved states even when the app itself has resumed long ago.

Media Playback

Next, if your app uses the Silverlight MediaElement control or the XNA MediaPlayer class, you must check that the user is not playing any music before setting the MediaElement's source or playing anything with MediaPlayer. To do this, check the value of Microsoft.Xna.Framework.Media.MediaPlayer.GameHasControl. If you're doing this from Silverlight, be sure to link the project to the Microsoft.Xna.Framework assembly. Once you have performed this check, and background music is playing, you have two options. You can either silently skip any music playback and let the user's current music continue playing, or you can explicitly ask the user for

permission to stop playback. If the user agrees, you may safely stop the user's music playback and continue. If the user does not agree, you must handle the situation gracefully.

If your app absolutely requires music playback, you might tell the user to press the back button and exit your app until they are ready for their music to stop. However, if your app can still function without music playback, you should let it do so, as that would be the appropriate predictable action. Note that XNA's SoundEffect class is not subject to this restriction because it does not affect music playback.

Design

When submitting, you will need to provide screenshots and information about your application. These items are used in the Marketplace to give customers more information about the application before they download.

You are asked to provide at least one but up to eight screenshots for your application. Section 4.6 of the requirements often catches people off guard as they submit screenshots that are not of the correct size or they contain emulator chrome.

Screenshots should:

- Encompass the full 480 x 800 dimension
- Never contain emulator chrome (the back button, search button, windows button or the emulator frame)
- Be a direct capture of the phone screen or emulator
- Represent the correct aspect ratio.

A guide on how you can create good screenshots can be found over on MSDN:

http://msdn.microsoft.com/en-us/library/gg442300(v=VS.92).aspx

For each application, you must submit one icon to represent your application in the Windows Phone Marketplace catalogue. This icon must closely match the icon provided in the XAP package. Users see this icon when browsing the application catalogue on the phone before making a purchase. Many people have attempted to submit an application using the default icon installed when you create a project. This leads to the application being rejected. The simple way to avoid this is to create your own icons.

When developing an application, many people test it using the dark theme but then do not test the application with the light theme. This can often cause issues where controls or text become unusable or unreadable because there is not sufficient contrast with the background colour. Making sure you test all of your application screens in both light and dark themes will ensure that you do not fail this certification test.

Toast Notification

If you are using toast notifications in your application, be sure that you ask for the user's permission to do so. This is covered in section 6.2 of the Requirements.

http://msdn.microsoft.com/en-us/library/hh184838(v=VS.92).aspx

Most publishers have asked this question in the start-up of the application using a dialog box. It's important to note that you are only required to ask the user for permission on the first use of the BindToShellToast method. You are not required to ask the user for permission again. For example, if your application calls BindToShellToast each time your application loads, you would only prompt the user the first time the application is launched.

Locked Screen

If you are creating an application that you wish to run even when the phone is locked, you must ask the user for explicit permission. If you fail to do so, your application will be rejected. Most publishers are choosing to ask the user first if it is ok to run the application under a lock screen when the application starts for the first time. It is also a good idea to add a switch to the settings portion of your application so that they can alter this option at a later stage.

Use auto publish with caution

Finally, a small piece of advice. Unless you are 100% certain that the app you are submitting will have no bugs discovered two minutes after submission, uncheck the box for automatically publishing your app to the Marketplace. You'll get an email when the app is ready to be published, and once you're satisfied with its lack of last-minute bugs, you can publish it from the App Hub.

Interlude: Jobsite.co.uk goes mobile

Martyn Buckland built the Windows Mobile version of Jobsite.co.uk – the UK's leading online job board. The app lets job-hunters find jobs that meet their needs; for example, by looking for local jobs or part-time positions. It also has interview advice from some of the UK's leading bosses. Just what you need if you're looking for something new.

Buckland had already started exploring Windows Phone development – "It's new, fun and cutting edge and I was happy to put the time in." So when the opportunity came to build the Jobsite.co.uk app, he had a head start. His background in C#, .NET and web development was a massive advantage as the app had to be turned around in just over a month!

"The first 90 percent of the development was easy, but the last 10 percent was a challenge," says Buckland. One stumbling block was tombstoning the app if there's an incoming call on the user's

Windows Phone 7. Another was the problem that mobiles can lose the internet connection/signal at any time and deal with it gracefully. He recommends building a framework to deal with these issues. Subsequent apps will become much easier as a result.

Buckland concludes, “Developing mobile apps are a very beneficial investment for Jobsite, when comparing the traffic to Jobsite via mobile and desktop we can see a clear trend for people using their phones on the commute to and from work, and also later at night when working on a computer is less convenient. People don’t always want to use their PC at work to browse job sites and the Jobsite Jobs app on Windows Phone 7 enables job seekers to search for jobs whenever and wherever they want, ensuring they never miss an opportunity.” . Jobsite’s app shows that a well-written program can enhance a brand, offer a better user experience and differentiate a service.

A tour of libraries and samples

By Stuart Lodge

If you're a Windows Phone developer then the open source support is excellent in this chapter we will introduce you to some of the open source projects that you really need to know about, but there are plenty more beyond this list. The following sites are ones to keep your eyes on:

- http://codeplex.com
- http://github.com
- http://create.msdn.com
- http://silverlight.net

Many Windows Phone 7 projects are now increasingly available via NuGet. NuGet is a free, open source package management system for the .NET platform which simplifies adding 3rd party libraries into your application. Open source projects are compiled into Packages, you can search for new packages on the NuGet website (http://nuget.org/List/Packages) or if you have the Extension, via Visual Studio itself.

As for using open source packages in your application, The WP7 App Hub welcomes open source apps and libraries – with the sole exception of those licensed under the very restrictive GPLv3 family of licenses.

MVVM

Model-View-ViewModel is the design pattern used by many WP7 apps. It's a convenient way to keep the different layers of your application separate.

To learn more about MVVM this video from Laurent Bugnion is a great introduction.
http://bit.ly/MVVMIntro

There are several projects which aim to assist with the implementation of MVVM on WP7.

Perhaps the best known is GalaSoft's MVVM Light - http://mvvmlight.codeplex.com/ - with a getting started guide on http://galasoft.ch/mvvm/

This library includes:

- A base class for your ViewModels
- A Messenger framework to allow communication between different instances of classes in your apps
- A RelayCommand class with EventToCommand helper to allow XAML binding of actions to events.

If MVVM Light isn't quite what you need, then there are others to look at:

- Caliburn Micro - http://caliburnmicro.codeplex.com/

- Catel - http://catel.codeplex.com/
- Simple - http://simplemvvmtoolkit.codeplex.com/
- And more appearing all the time!

Dependency Injection

When creating applications you often want to reuse components in other applications, for this reason it's often beneficial to make sure your components do not have dependencies on other classes in your application. One way to avoid creating a dependency is to inject objects into a class rather than relying on the class to create the object itself.

If you want to use dependency injection there are several tools available for WP7 – in particular, check out:

- Ninject on https://github.com/ninject/ninject
- Funq on http://funq.codeplex.com/

UI Controls

There are several libraries of open source controls available.

Silverlight Toolkit

http://silverlight.codeplex.com/

This is the best known of the toolkits. The WP7 version contains many controls from the full framework plus a few WP7 specific ones – e.g. the Page Transitions helpers.

The current control list is:

- AutoCompleteBox
- ContextMenu
- DatePicker
- GestureService/GestureListener
- ListPicker
- LongListSelector
- Page Transitions
- PerformanceProgressBar
- TiltEffect
- TimePicker
- ToggleSwitch
- WrapPanel

Additionally, it's well worth WP7 developers knowing what is in the "full" Silverlight toolkit – as many additional controls can also be imported to WP7 – e.g. several apps have imported charting from the full toolkit.

Coding For Fun

The Coding for Fun site has released the open source control library
http://coding4fun.codeplex.com/

This C4F is documented in depth on http://www.windowsphonegeek.com/articles/12-wp7-coding4fun-toolkit-in-depth-articles-covering-all-controls

It includes controls for:

- About Prompt
- Input Prompt
- Progress Overlay
- Round Button
- Round Toggle Button
- Memory Counter
- TimeSpan Picker
- Toast Prompt

WP7 Contrib

http://wp7contrib.codeplex.com/

Very much a project that is open to contributions, this library provides a number of utilities to assist with MVVM and binding, plus some controls for:

- Page transitions
- DateTime
- Range Slider

- DeferredLoadListBox
- LowProfileImageLoader
- DynamicTextBox
- TiltEffect

Phoney Tools

http://phoney.codeplex.com/

These tools are all from Shawn Wildermuth's blog - http://wildermuth.com/wp7

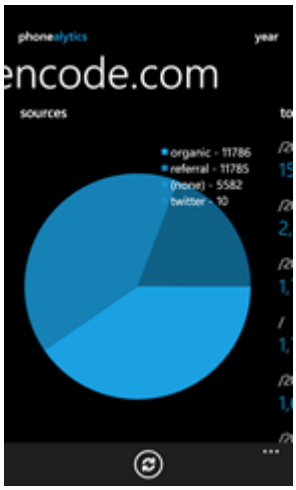
It includes controls:

- FadingMessage
- SelectSwitch
- SimpleLongListSelector

And it also includes lots of utilities like:

- BitlyHelper
- GravatarHelper
- InstaPaperHelper
- PhoneLogger
- Phone Resources classes
- GameTimer
- MicrophoneRecorder
- SoundEffectPlayer
- PhoneNetworking

UI Controls - Charting



When using chart controls, be careful to check the license - currently several of the charting libraries are on the slightly less liberal open source licenses.

Here are some of the most useful libraries I've found on the more liberal licenses

- AmCharts - http://wpf.amcharts.com/quick - MS-PL and used in a couple of great apps already!
- Silverlight Toolkit – there are some blog posts around about how to use the charting from the full Silverlight toolkit – e.g. see http://newtonapps.com/post/2010/05/05/Windows-Phone-7-2b-Silverlight-Toolkit-Data-Visualization.aspx

Services

If you want to connect your app to some existing network services, then here are some libraries and sample apps that you might find particularly of interest:

- DropBox - http://wp7dropbox.codeplex.com/ and http://sharpdropbox.codeplex.com/
- Twitter – http://twitt.codeplex.com/, http://mahtweetswp7.codeplex.com/ and http://loudtweets.codeplex.com/
- Facebook – emedded withing http://bewisephonecontrols.codeplex.com/
- TextToSpeech – Bing translator APIs - http://wptts.codeplex.com/
- FourSquare - http://4square.codeplex.com/
- Generic REST - http://hammock.codeplex.com/
- Generic ODATA - http://odata.codeplex.com/

Data storage

Most apps need to store something inside Isolated Storage.

If you're looking for relational data storage, then there are several embedded database solutions available

- Sterling - http://sterling.codeplex.com/
- WP7 DB - http://winphone7db.codeplex.com/
- SQLLite - http://wp7sqlite.codeplex.com/

These libraries provide table and entity CRUD operations, and provide great Linq access to your data.

For Serialization of more generic data objects, then some of the most popular options are:

- JSON – using NewtonSoft's JSON.Net library - http://json.codeplex.com/ or using the WP7 DataContractSerializer functionality.
- For when speed and size really matter (which is *lots* of mobile apps), then a binary format is perfect
    - There's one library available on http://whydoidoit.com/silverlight-serializer/
    - If you need versionable and process-interchangeable binary messages, then you might want to consider Google's Protocol Buffers - http://blog.chrishayuk.com/2010/12/protocol-buffers-generator.html - but beware that this might not be the easiest to use approach!

If you need to zip your data, then don't miss http://slsharpziplib.codeplex.com/ - but don't overuse this on the phone – remember that while zip saves "disk space" and "network bandwidth" it also comes with penalties of processing and RAM at runtime.

For examining the contents of your app's storage, then IsolatedStorage Explorer is an indispensable library and tool - http://wp7explorer.codeplex.com/

Tombstoning

For automated tombstoning support on your Silverlight application pages, the TombstoneHelper project can automatically serialise and then rehydrate your UI controls - http://tombstonehelper.codeplex.com/

Interesting Input

Your Windows Phone can provide you with lots of interesting ways for the user to interact:

- You can use the camera to view barcode – see the WP7 port of the ZXIng library at http://silverlightzxing.codeplex.com/
- You can use single and multitouch Gestures like Pinch and Swipe - http://multitouch.codeplex.com/ - plus see the Gesture Service inside the Silverlight Toolkit
- You can use the accelerometer to detect Shake Gestures - http://create.msdn.com/en-us/education/catalog/article/Recipe_Shake_Gesture_Library

Physics – For Gaming

If you're interested in gaming, then the farseer physics engine provides behaviours for physics – so you can code games straight from Expression Blend – see http://farseerphysics.codeplex.com/ plus several other projects that extend it and show you how to use it.

Beyond This List…

The WP7 open source community is publishing new projects every day – so do keep an eye on new releases on codeplex and github – one easy way to keep up is to follow @wpug.