

Microsoft®

WINDOWS® PHONE 7

DEVELOPER GUIDE

Building connected
mobile applications
with Microsoft Silverlight®

Dominic Betts
Federico Boerr
Scott Densmore
Jose Gallardo Salazar
Alex Homer



patterns & practices

WINDOWS® PHONE 7 DEVELOPER GUIDE

Windows® Phone 7 Developer Guide

Building connected mobile applications
with Microsoft Silverlight®

Dominic Betts
Federico Boerr
Scott Densmore
Jose Gallardo Salazar
Alex Homer

ISBN: 978-0-7356-5609-3

Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Azure, Windows Live, Windows Vista, ActiveSync, Bing, Direct3D, DirectX, Excel, Expression Blend, Internet Explorer, MSDN, SharePoint, Silverlight, SQL Azure, Visual Basic, Visual C++, Visual C#, Visual Studio, Xbox, Xbox 360, XNA, and Zune are registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

Other product and company names herein may be trademarks of their respective owners.

Contents

FOREWORD	xiii
Charlie Kindel	
FOREWORD	xv
Istvan Cseri	
PREFACE	xvii
Who This Book Is For	xviii
Why This Book Is Pertinent Now	xviii
How This Book Is Structured	xix
The Example Application	xxi
What You Need to Use the Code	xxi
Who's Who	xxii
Where to Go for More Information	xxiiii
ACKNOWLEDGMENTS	xxv
1 Introducing Windows Phone 7	1
A Standardized Platform	2
Data-driven Applications	4
Developing for the Windows Phone 7 Platform	5
Resources for Developers	6
Resources for Silverlight Developers	7
Resources for XNA-Based Application Developers	8
Resources for Web and Service Developers	8
Terminology	9
More Information	14
Blogs, Code Samples, Training Kits, and Windows Marketplace	15

2	Designing Applications for Windows Phone 7	
	Basic Design Considerations	17
	Type of Application	19
	Design and Implementation	19
	Resource Management	19
	Remote Services	20
	Mobile Phone Client Applications	20
	Suitable Application Types for Mobile Phones	21
	Silverlight and Windows Phone 7	22
	Design Considerations for Windows Phone 7 Applications	23
	User Interface Design and Style Guidelines	24
	Application Deactivation and Tombstoning	25
	User Input Considerations	28
	Storage Considerations	29
	Connectivity Considerations	31
	Security Considerations	32
	Data Formats and Accessing Remote Services	34
	Data Format and Synchronization Considerations	35
	Microsoft Data and Synchronization Technologies for Windows Phone 7	36
	Resource Management and Performance	38
	Minimize Device Resource Usage	38
	Apply Good Practice Programming Techniques	38
	Optimize Memory Usage	39
	Maximize GPU Usage	41
	Availability of Components and Frameworks	42
	Questions	43
3	The Tailspin Scenario	45
	The Tailspin Company	45
	Tailspin's Strategy	45
	Tailspin's Goals and Concerns	46
	The Surveys Application Architecture	47
	The Actors	48
	Tailspin - the ISV	48
	Fabrikam and Adatum - the Subscribers	49
	The Surveyors - Windows Phone 7 Users	49
	The Business Model	49
	The Application Components	49

4 Building the Mobile Client	53
Overview of the Mobile Client Application	53
Goals and Requirements	53
Usability Goals	54
Non-Functional Goals	55
Development Process Goals	56
The Components of the Mobile Client Application	56
The Structure of the Tailspin Surveys client Application	57
Dependency Injection	59
The Contents of the TailSpin.PhoneClient Project	60
The Design of the User Interface	61
Navigation	61
User Interface Description	65
User Interface Elements	66
The Pivot Control	66
The Panorama Control	66
Styling and Control Templates	66
Context Menus	67
Using the Model-View-ViewModel Pattern	67
The Premise	68
Overview of MVVM	68
Benefits of MVVM	70
Connecting the View and the View Model	71
Inside the Implementation	71
Testing the Application	74
Inside the Implementation	74
Displaying Data	75
Inside the Implementation	75
Commands	85
Inside the Implementation	85
Handling Navigation Requests	88
Inside the Implementation	88
User Interface Notifications	91
Informational/Warning Notifications	91
Error Notifications	92
Inside the Implementation	92
Accessing Services	94
Questions	94
More Information	96

5 Using Services on the Phone	97
The Model Classes	97
Using Isolated Storage on the Phone	98
Overview of the Solution	99
Security	99
Storage Format	100
Inside the Implementation	100
Application Settings	101
Survey Data	103
Handling Activation and Deactivation	106
Overview of the Solution	106
Inside the Implementation	107
Reactivation and the Panorama Control	110
Handling Asynchronous Interactions	111
Using Reactive Extensions	111
Inside the Implementation	112
Synchronizing Data between the Phone and the Cloud	115
Overview of the Solution	116
Limitations of the Current Approach	117
Inside the Implementation	118
The getNewSurveys Task	120
The saveSurveyAnswersTask	122
Using Location Services on the Phone	123
Overview of the Solution	123
Inside the Implementation	124
Acquiring Image and Audio Data on the Phone	127
Overview of the Solution	127
Capturing Image Data	127
Recording Audio Data	127
Inside the Implementation	127
Using a Chooser to Capture Image Data	128
Using XNA Interop to Record Audio	130
Logging Errors and Diagnostic Information on the Phone	132
Conclusion	133
Questions	133
More Information	135

6 Connecting with Services	137
Installing the Mobile Client Application	137
Overview of the Solution	138
Inside the Implementation	138
Authenticating with the Surveys Service	139
Goals and Requirements	140
Overview of the Solution	140
A Future Claims-Based Approach	141
Inside the Implementation	143
Notifying the Mobile Client of New Surveys	146
Overview of the Solution	146
Inside the Implementation	148
Registering for Notifications	148
Sending Notifications	155
Notification Payloads	160
Accessing Data in the Cloud	162
Goals and Requirements	162
Overview of the Solution	162
Exposing the Data in the Cloud	162
Data Formats	163
Consuming the Data	163
Using SSL	164
Inside the Implementation	164
Creating a WCF REST Service in the Cloud	164
Consuming the Data in the Windows Phone 7 Client Application	167
Filtering Data	172
Overview of the Solution	172
Registering User Preferences	172
Identifying Which Devices to Notify	174
Selecting Surveys to Synchronize	175
Inside the Implementation	176
Storing Filter Data	177
Building a List of Devices to Receive Notifications	179
Building a List of Surveys to Synchronize with the Mobile Client	180
Summary	181
Questions	181
More Information	183

7	Interacting with Windows Marketplace	185
	The Application Development and Publishing Life Cycle	185
	Application Certification Requirements	187
	Application Code Restrictions	188
	Run-time Behavior, Performance, and Metrics	189
	User Opt-in and Privacy	189
	Application Media and Visual Content	190
	Packaging the Application	191
	The Windows Marketplace Repackaging Process	192
	Summary of the Submission and Validation Process	193
	Displaying Advertisements in an Application	195
	Accessing Windows Marketplace within an Application	196
	Questions	196
 APPENDICES		
A	TOOLS, FRAMEWORKS, AND PROCESSES	199
	Setting Up a Development Environment	
	for Windows Phone 7	199
	Expression Blend for Windows Phone	202
	Additional Silverlight Controls	
	for Windows Phone 7	203
	Using a Hardware Device during Development	204
	Connecting a Physical Device	204
	Registering and Unlocking the Device	204
	Deploying Applications to the Device	205
	Using the Windows Phone Connect Tool	205
	Developing Windows Phone 7 Applications	206
	Developing Trial Applications for Windows Phone 7	208
	Testing Trial Versions in an Emulator or Device	208
	Developing Web Applications for Windows Phone 7	209
	Debugging Windows Phone 7 Applications	209
	Unit Testing Windows Phone 7 Applications	211
	Automated Unit Testing	212
	Additional Tools and Frameworks	212
B	SILVERLIGHT AND XNA IN WINDOWS PHONE 7	215
	Basic Differences between Silverlight and XNA	215
	Execution Model	215
	Performance	215
	Sounds	216
	Screen Display	216
	Summary of the Basic Differences	216

The XNA Game Execution Model	217
Using Interop from Silverlight to XNA	218
Creating and Using an XNA Dispatcher Service	218
Excluded Classes and Assemblies	220
C LEVERAGING DEVICE CAPABILITIES	221
Scenarios for Device Capabilities	222
Accelerometer	223
Camera	226
Contacts and Messaging	228
Device Information	230
Location and Mapping	232
Asynchronous Location Service Operation	233
Synchronous Location Service Operation	235
Using Location Information	
and Displaying a Map	236
The Bing Maps Silverlight Control	236
Media	237
Selecting a Photo on the Device	239
Search	240
Sound Recording	240
Sound Playback	243
Touch and Gestures	244
Gesture Detection using Silverlight	
Manipulation Events	245
Gesture Detection Using the GestureListener Control	246
Gesture Detection Using XNA	250
Vibration Alerts	252
Web Browser	253
Windows Marketplace	253
Reactive Extensions	254
D PRISM LIBRARY FOR WINDOWS PHONE 7	257
About Prism for Windows Phone 7	258
Contents of Prism for Windows Phone 7 Library	258
Microsoft.Practices.Prism Namespace	259
Microsoft.Practices.Prism.Commands Namespace	260
Microsoft.Practices.Prism.Events Namespace	260
Microsoft.Practices.Prism.ViewModel Namespace	261

Microsoft.Practices.Prism.Interactivity Namespace	262
Microsoft.Practices.Prism.Interactivity.Interaction Request Namespace	263
E MICROSOFT SYNC FRAMEWORK AND WINDOWS PHONE 7	
About the Microsoft Sync Framework	265
Components of the Sync Framework	267
Sync Framework Providers	269
Using the Sync Framework	270
Synchronization for Windows Azure and Windows Phone 7	271
ANSWERS TO QUESTIONS	273
Chapter 2: Designing Windows Phone 7 Applications	273
Chapter 4: Building the Mobile Client	275
Chapter 5: Using Services on the Phone	278
Chapter 6: Connecting with Services	282
Chapter 7: Interacting with Windows Marketplace	285
INDEX	289

Foreword

Great achievements don't happen overnight—they evolve over time based on a series of successes that converge and drive you onward. My favorite soccer team, Seattle Sounders FC, started life way back in 1974 in the North American Soccer League and only achieved their recent success in Major League Soccer through working hard to raise their game and improve their results.

Here at Microsoft, we've always focused on raising our game. In my 20 years with the company, I've worked on projects ranging from the Microsoft® Windows® 2.0 SDK and COM, to Windows Media® Center and Windows Home Server. Each new generation of products raises the game for both users and developers. And now I'm proud to be part of the team that's driving our latest achievement, Windows Phone 7.

Windows Phone 7 is a different kind of phone, designed for life in motion. It's a change from the past that incorporates smart design and is aimed at users who need to manage their personal and business lives as an integrated experience. Or, to be more accurate, a series of integrated experiences that include People, Office, Pictures, Music and Videos, Windows Market Place, and Games.

When designing Windows Phone 7, we stepped back and thought hard about who our customers are and what they need from a phone. Everyone on the Windows Phone 7 team woke up every morning thinking "What can I do today to make the end-user experience great?" This was true of even the people focused on building the developer experience. The end user always came first; our mantra was "Enable end users to personalize their phone experience with great applications and games and ensure that developers can be profitable." It has been extremely gratifying to see the incredible innovation being brought forward by third-party developers building Windows Phone 7 applications and games.

Based on our experience building the Xbox 360®, Windows Media® Center, and Zune®, we built a phone that users can personalize and make their own, that helps developers be profitable, and that

enables cloud-powered experiences that align with the Microsoft vision for “three screens and the cloud” computing.

This book, with its practical scenario-based approach, will help you to be part of that vision. It will guide you through the process of understanding Windows Phone 7, getting started developing applications for the phone, and creating beautiful and engaging user experiences that achieve success in this new and exciting marketplace.

The book explores the four main areas of focus: the phone runtime (code that you write to run on the client), services (code that runs on the cloud), tools to help you design and develop your applications, and tools that help you ship and sell your applications. It does all this within the context of a fictional company that is extending its cloud-powered application to the phone.

As a developer, you need to be part of this new world where life happens on the move. We’ve made it easy to leverage your existing skills and apply them to the phone. This book will help you raise your game and score in this exciting new market.

Sincerely,
Charlie Kindel
General Manager, Windows Phone 7

Foreword

Microsoft® Windows® Phone 7 Series represents a major advance in mobile devices. We have created a completely new model that gives users a new class of phone. Everything from the underlying operating system and the application platform, to the delivery, style, and performance of applications has been engineered to provide a great user experience.

Windows Phone 7 is a major part of the Microsoft vision for mobile computing, and a core component in the “three screens and the cloud” philosophy. It allows users to enjoy an immersive and productive environment, full interaction with other Microsoft platforms and applications, and access to all existing web experiences.

We have made the phone intuitive for users and very familiar for developers writing applications. Developers can reuse their skills, tools, knowledge, and experience to create great mobile applications using the Microsoft .NET Framework, Silverlight®, XNA®, Visual Studio®, and more. These applications and the development environment are consistent across the desktop, the web, the cloud, and mobile devices.

This guide is a great starting point for your journey through Windows Phone 7 development. It provides a pragmatic, actionable approach to planning, designing, and building applications that can reach out into the cloud and take advantage of the power and capabilities available there for both business users and consumers.

In this guide, you will see how easy it is to build applications that work well for “life on the move”, provide a compelling and integrated experience for users, and allow you to create a revenue source. We have worked very hard to make Windows Phone 7 an indispensable part of the user’s lifestyle, and this guide will help you to play your part in this exciting new mobile world.

Sincerely,
Istvan Cseri
Distinguished Engineer, Windows Phone

Preface

Windows® Phone 7 provides an exciting new opportunity for companies and developers to build applications that travel with users, are interactive and attractive, and are available whenever and wherever users want to work with them.

By combining Windows Phone 7 applications with on-premises services and applications, or remote services and applications that run in the cloud (such as those using the Windows Azure™ technology platform), developers can create highly scalable, reliable, and powerful applications that extend the functionality beyond the traditional desktop or laptop; and into a truly portable and much more accessible environment.

This book describes a scenario around a fictitious company named Tailspin that has decided to encompass Windows Phone 7 as a client device for their existing cloud-based application. Their Windows Azure-based application named Surveys is described in detail in a previous book in this series, *Developing Applications for the Cloud on the Microsoft Windows Azure Platform*. For more information about that book, see the page by the same name on MSDN® at (<http://msdn.microsoft.com/en-us/library/ff966499.aspx>).

In addition to describing the client application, its integration with the remote services, and the decisions made during its design and implementation, this book discusses related factors, such as the design patterns used, the capabilities and use of Windows Phone 7, and the ways that the application could be extended or modified for other scenarios.

The result is that, after reading this book, you will be familiar with how to design and implement applications for Windows Phone 7 that take advantage of remote services to obtain and upload data while providing a great user experience on the device.

Who This Book Is For

This book is part of a series on Windows Azure service and client application development. However, it is not limited to only applications that run in Windows Azure. Windows Phone 7 applications can interact with almost any service—they use data exposed by any on-premises or remote service. Even if you are building applications for Windows Phone 7 that use other types of services (or no services at all), this book will help you to understand the Windows Phone 7 environment, the development process, and the capabilities of the device.

This book is intended for any architect, developer, or information technology (IT) professional who designs, builds, or operates applications and services for Windows Phone 7. It is written for people who work with Microsoft® Windows-based operating systems. You should be familiar with the Microsoft .NET Framework, Microsoft Visual Studio® development system, and Microsoft Visual C#®. You will also find it useful to have some experience with Microsoft Expression Blend® design software and Microsoft Silverlight®, although this is not a prerequisite.

Why This Book Is Pertinent Now

Mobile devices, and mobile phones in particular, are a part of the fundamental way of life for both consumers and business users. The rapidly increasing capabilities of these types of devices allow users to run applications that are only marginally less powerful, and in most cases equally (or even more) useful than the equivalent desktop applications. Typical examples in the business world are email, calendaring, document sharing, and other collaboration activities. In the consumer market, examples include access to social interaction sites, mapping, and games.

Windows Phone 7 is a new entry into this field, and it is very different from previous versions of Microsoft mobile operating systems. It has been built from the ground up to match the needs and aspirations of today's users, while standardizing the hardware to ensure that applications perform well on all Windows Phone 7 devices. The result is a consistent run-time environment and a reliable platform that uses familiar programming techniques.

Developers can use the tools they already know, such as Visual Studio, to write their applications. In addition, the Windows Phone Developer Tools provide a complete emulation environment and additional tools specially tailored for developing Windows Phone 7 applications. Developers can use these tools to write, test, and debug their applications locally before they deploy them to a real device for

final testing and acceptance. This book shows you how to use these tools in the context of a common scenario—extending an existing cloud-based application to Windows Phone 7.

How This Book Is Structured

You can choose to read the chapters in the order that suits your existing knowledge and experience, and select the sections that most interest you or are most applicable to your needs. However, the majority of the chapters follow a logical sequence that describes the development environment and the stages of designing and building the application. Outside of this main stream, other chapters and appendices provide information about more specialized topics, such as validating and selling your application or interacting with device capabilities, such as the camera, Global Positioning System (GPS), and other sensors. Figure 1 illustrates.

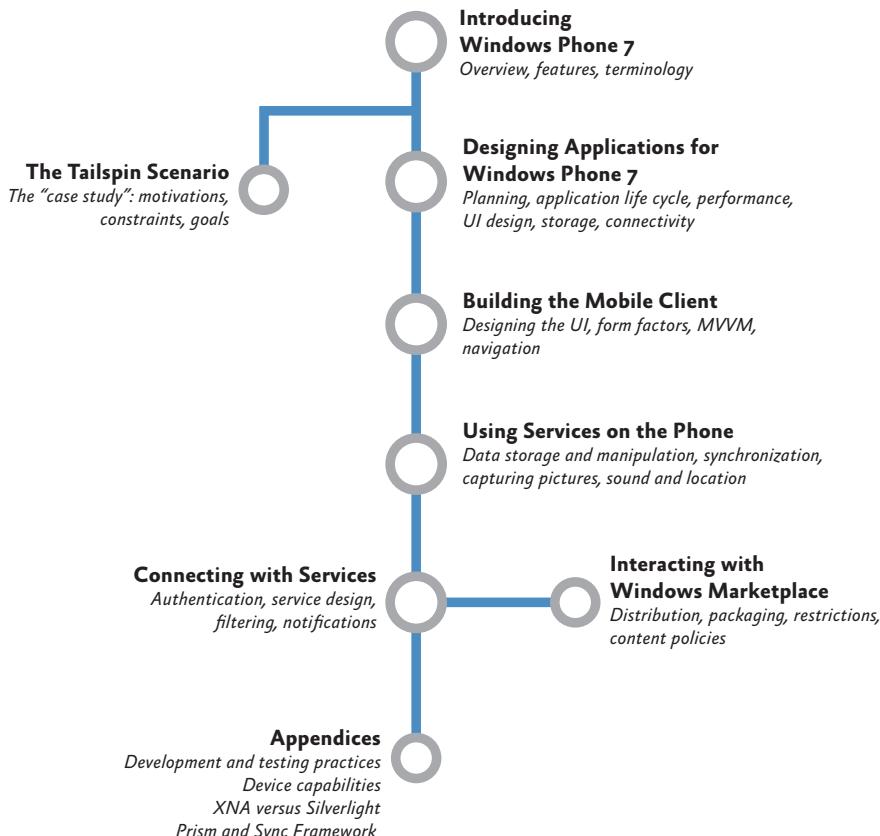


FIGURE 1
The book structure

Chapter 1, “Introducing Windows Phone 7,” provides an overview of the platform to help you understand the requirements and advantages of creating applications for Windows Phone 7. It provides a high-level description of the possibilities, features, and requirements for building applications for Windows Phone, and it includes references to useful information about designing and developing these types of applications. It also includes a glossary of terms commonly used in mobile application development. It’s probably a good idea to read this chapter before moving on to the scenarios.

Chapter 2, “Designing Windows Phone 7 Applications,” discusses planning and designing applications for Windows Phone 7. It covers the run-time environment and life cycle events for your application, how to maximize performance on the phone, and considerations for the user interface, resource management, storage, connectivity, and more.

Chapter 3, “The Tailspin Scenario,” introduces you to the Tailspin company and the Surveys application. It describes the decisions that the developers at Tailspin made when designing their application, and it discusses how the Windows Phone 7 client interacts with their existing Windows Azure-based services.

Chapter 4, “Building the Mobile Client,” describes the steps that Tailspin took when building the mobile client application for Windows Phone 7 that enables users to register for and download surveys, complete the surveys, and upload the results to the cloud-based service. It includes details of the overall structure of the application, the way that the Model-View-ViewModel (MVVM) pattern is implemented, and the way that the application displays data and manages commands and navigation between the pages. The following chapters describe the individual features of the application development in more detail.

Chapter 5, “Using Services on the Phone,” discusses the way that the Windows Phone 7 client application stores and manipulates data, manages activation and deactivation, synchronizes data with the server application, and captures picture and sound data.

Chapter 6, “Connecting with Services,” describes how the client application running on Windows Phone 7 uses the services exposed by the Windows Azure platform. This includes user authentication, how the client application accesses services and downloads data, the data formats that the application uses, filtering data on the server, and the push notification capabilities.

Chapter 7, “Interacting with Windows Marketplace,” describes how you can distribute and sell your applications through Windows Marketplace, and the restrictions and conditions Windows Marketplace places on your applications and content.

The appendices include additional useful information related to the topics described in the rest of the chapters. The appendices cover getting started with the Windows Phone developer tools; testing your applications; information about the development environments (Silverlight and XNA® development platform); a reference section for programming device capabilities, such as location services, messaging features, and the camera; information about the Prism Library for Windows Phone 7; and an overview of data and file synchronization using emerging technologies such as Microsoft Sync Framework.

The Example Application

This book has an accompanying example application—the Surveys client that Tailspin built to expose their cloud-based surveys application on Windows Phone 7. You can download the application and run it on your own computer to see how it works and to experiment and reuse the code.

The application is provided in two versions to make it easier for you to see just the Windows Phone 7 client or the combined Windows Phone 7 and Windows Azure application. If you want to try only the Windows Phone 7 client, you can run the simplified version of the application that uses mock service implementations to provide the data required by the client application. You do not need to install the Windows Azure run-time environment and development fabric to use this version.

However, if you want to see the complete application in action, and work with the Windows Azure service, you can run the full version. For this, you must install the complete Windows Azure run-time environment and development fabric. The example includes a dependency checker application that will ensure you have all the prerequisites installed and configured for this version; it will also help you locate and install any prerequisites that are missing on your system.

To read more and download the application, see the patterns & practices Windows Phone 7 Developer Guide community site on CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

What You Need to Use the Code

These are the system requirements for running the scenarios:

- Microsoft Windows Vista® operating system (x86 and x64) with Service Pack 2 (all editions except Starter Edition) or Microsoft Windows 7 (x86 and x64) (all editions except Starter Edition)
- Windows Phone Developer Tools
- Microsoft Internet Information Services (IIS) 7.0
- Microsoft .NET Framework 4.0 or later

Alternatively, you can use Visual Studio 2010 to develop Windows Phone 7 applications instead of the version of Visual Studio Express that is included with the Windows Phone Developer Tools. However, you must still install the Windows Phone Developer Tools. Visual Studio provides additional capabilities for testing and debugging Windows Phone 7 applications and building more complex Windows Phone 7 application solutions.

If you want to run the full version of the example, which uses a Windows Azure service to provide the data and authentication services to the device, you must also install the following:

- Windows Azure Tools for Microsoft Visual Studio 2010
- Windows Identity Foundation

Who's Who

This book uses a set of scenarios that demonstrate designing and building the Windows Phone 7 client application and integrating it with cloud-based services. A panel of experts comments on the development efforts. The panel includes a Windows Phone 7 specialist, a software architect, a software developer, and an IT professional. The scenarios can be considered from each of these points of view. The following table lists the experts for these scenarios.



Christine is a phone specialist. She understands the special requirements inherent in applications designed to be used on small mobile devices. Her expertise is in advising architects and developers on the way they should plan the feature set and capabilities to make the application usable and suitable for these types of devices and scenarios.

To build successful applications that work well on the phone, you must understand the platform, the user's requirements, and the environment in which the application will be used.

Jana is a software architect. She plans the overall structure of an application. Her perspective is both practical and strategic. In other words, she considers not only what technical approaches are needed today, but also what direction a company needs to consider for the future.



It's not easy to balance the needs of the company, the users, the IT organization, the developers, and the technical platforms we rely on.



Markus is a senior software developer. He is analytical, detail-oriented, and methodical. He's focused on the task at hand, which is building a great application. He knows that he's the person who's ultimately responsible for the code.

I don't care what platform you want to use for the application, I'll make it work.

Poe is an IT professional who's an expert in deploying and running applications in a corporate data center. Poe has a keen interest in practical solutions; after all, he's the one who gets paged at 3:00 AM when there's a problem.

Integrating our server-based applications with mobile devices such as phones is a challenge, but it will broaden our reach and enable us to implement vital new capabilities for our applications and services.



If you have a particular area of interest, look for notes provided by the specialists whose interests align with yours.

Where to Go for More Information

There are a number of resources listed in text throughout the book. These resources will provide additional background, bring you up to speed on various technologies, and so forth. For your convenience, there is a bibliography online that contains all the links so that these resources are just a click away.

You can find the bibliography at:

<http://msdn.microsoft.com/en-us/library/gg490786.aspx>

Acknowledgments

When I joined Microsoft® patterns & practices in May 2004, my projects were all related to client development: smart clients and web clients, mostly. At that time, we considered it natural to extend our guidance to mobile clients. The result of that was the Mobile Client Software Factory, which was released in July 2006.

As I was preparing for this project, I looked back at the work we did at that time, and I was surprised in two very different and opposite ways. First, the list of technical challenges to cover was surprisingly similar. Both mentioned things like UI design and dealing with networks. Second, modern devices are light years ahead of what we had at that time: much more memory is available, graphics processor units (GPUs) now exist, there are more sophisticated sensors, and, of course, the cloud is a much more powerful back end. A lot has remained the same, and a lot has changed.

This book covers two extremes of the Microsoft Windows® platforms: the massive computing resources of Windows Azure™ and the personal, tailored experience of the Windows Phone 7. As we were developing this content, I was reminded of the richness of the Microsoft platform, and the opportunities it offers to developers today. Ideas that were merely seeds in our imagination a decade ago or that were available to only large corporations with huge resources, are now accessible to everyone with a PC. I feel privileged to have contributed, even a little bit, toward making this happen.

This guide follows the same scenario-based approach we used in our previous three guides on Windows Azure development and claims-based identity. We created a fictitious, yet realistic, sample that is used as a case study throughout the chapters. The sample and the guide are complementary. You will find that the guide covers tradeoffs and design considerations that go beyond what is implemented in code. Often, there are many ways to solve one particular technical challenge. We tried to surface those tradeoffs and the thinking behind our decisions to equip you with the tools to make your own decisions in your own environments.

In the code, you will find that we have chosen to solve many problems in ways that are new and perhaps unexpected. An example of this is the extensive use of the Reactive Extensions for .NET Framework for all the asynchronous network calls. We chose to do this because it is our mission to empower you with better tools and frameworks.

I want to start by thanking the following subject matter experts and main contributors to this guide: Dominic Betts, Federico Boerr, Bob Brumfield, Jose Gallardo Salazar, Scott Densmore, and Alex Homer. Dominic is a veteran of many patterns & practices guides. As I wrote before, Dominic has this unique ability to explain complex topics in simple terms without losing rigor. Federico has been a member of our team since the very first guide we wrote for Windows Azure and has both the technical expertise and the gift of empathy, an essential attribute required to write guidance. Bob is an outstanding developer who brought an incredible wealth of experience and knowledge about Microsoft Silverlight® development, the main framework used throughout the guide to build applications on the phone. Jose was one of the original developers of the Mobile Client Software Factory, and is a very experienced mobile developer, who at the same time “gets” what developing guidance is all about. I feel very privileged to have worked with Scott every day—his knowledge spans an amazing spectrum, from devices to Windows Azure™, which is exactly what we needed for this guide. For this project, he also brought the unique perspective of an iPhone developer. I share two passions with Alex Homer: software and railways. We were very lucky to count on Alex’s experience as a technical author; he contributed to the solid structure and flow of this guide.

Many thanks also to the project’s development and test teams for providing a good balance of technically sound, focused code: Federico Boerr (Southworks SRL), Bob Brumfield, Scott Densmore, Chris Keyser, Jose Gallardo Salazar (Clarius Consulting), Masashi Narumoto, Lavanya Selvaraj (Infosys Technologies Ltd.), Mani Krishnaswami (Infosys Technologies Ltd.), and Ravindra Varman (Infosys Technologies Ltd.).

The written content in this guide is the result of our great technical writing and editing team. I want to thank Dominic Betts (Content Master Ltd.), Tina Burden (TinaTech Inc.), RoAnn Corbisier, Alex Homer, and Nancy Michell (Content Master Ltd.).

The visual design concept used for this guide was originally developed by Roberta Leibovitz and Colin Campbell (Modeled Computation LLC) for *A Guide to Claims-Based Identity and Access Control*. Based on the excellent responses we received, we decided to reuse this design in our most recent titles, including this one. The book design was created by John Hubbard (eson). The cartoon faces

were drawn by the award-winning Seattle-based cartoonist Ellen Forney. The technical illustrations were adapted from my Tablet PC mockups by Katie Niemer (TinaTech Inc.).

This guide, just like all our guidance content, was broadly reviewed, commented on, scrutinized, and criticized by a large number of customers, partners, and colleagues. Once again, we were extremely fortunate to tap into the collective intellectual power of a very diverse and skillful group of readers.

I also want to thank all of the people who volunteered their time and expertise on our early content and drafts. Among them, I want to mention the exceptional contributions of Shy Cohen, Istvan Cseri, Markus Eilers, Jonas Follesø, David Golds, David Hill, Yochay Kiriaty, Joel Liefke, Steve Marx, Erik Meijer, Miguel Angel Ramos Barroso, Jaime Rodriguez, Soumitra Sengupta, Ben Schierman, Erwin van der Valke, and Matias Woloski. A very special thanks is in order for the entire patterns & practices Prism team: Larry Brader, Bob Brumfield, Geoff Cox (Southworks SRL), Nelly Delgado, David Hill, Meenakshi Krishnamoorthi (Infosys Technologies Ltd.), Brian Noyes (iDesign), Diego Poza (Southworks SRL), Michael Puleio, Karl Schifflett, Fernando Simonazzi (Clarius Consulting), Rathi Velusamy (Infosys Technologies Ltd.), and Blaine Wastell.

Last but not least, I'd like to thank Charlie Kindel, the executive sponsor for this project.

I hope you find this guide useful!

A handwritten signature in black ink, appearing to read "Eugenio Pace".

Eugenio Pace
Senior Program Manager – *patterns & practices*
Microsoft Corporation
Redmond, Wa. October 2010

1

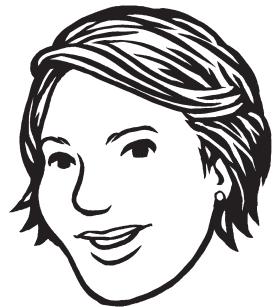
Introducing Windows Phone 7

Mobile computing has been growing in popularity over several years, while the mobile telephone has evolved into a ubiquitous device used by the majority of people today. The increasing capabilities of mobile phones, and the more universal coverage and higher bandwidth networks available to them, has meant that the combination of mobile computing and the phone has become a very persuasive and appealing platform for both consumer and enterprise applications.

Until relatively recently, people would have needed to carry around a personal digital assistant (PDA), a phone, a separate Global Positioning System (GPS) device, a pager or two-way text messaging device, and a digital camera. Now a single device provides all these capabilities and more, and at an affordable price.

It's now common to use a small mobile device to access phone services, music, video, Internet collaboration and information services, and communication systems such as email and messaging. Most modern smart phones and small form-factor devices have high resolution screens and advanced processing capabilities that support applications very similar to those more commonly seen on the desktop or in the domestic environment (such as MP3 players, TVs, and web browsers). Users can purchase and install applications from marketplace sites. In the future, corporations will also be able to distribute their own applications to selected users through private marketplaces.

Developers are increasingly exposing their enterprise and corporate applications and services to mobile devices such as phones to establish an "access everywhere" paradigm for all kinds of working scenarios, where the difference between a fixed and a mobile device is mainly usability and the availability of more advanced application features (and, of course, limited by screen size and battery life). Consumers also are adapting to a world where the primary mobile information, socializing, and entertainment device is a phone. How ubiquitous this is, is evident from the number of people who no longer have a fixed phone line, and who rely on a mobile phone for all their communication requirements, both business and social.



Using a phone as a comprehensive business tool is a perfect solution for mobile employees.

A Standardized Platform

In the past, the programming and run-time environments on the phone were often quite different from the almost universal model supported for the server, desktop, and laptop. Even where the development environments were similar (such as on the Microsoft® platform, where the ability to develop mobile applications using Microsoft Visual Studio®, Visual C++®, or Visual C# were similar), the wide range of screen resolutions, device capabilities, hardware nuances, and other incompatibilities made writing mobile applications a challenge. This tended to limit development of enterprise applications in the mobile device area.

However, with the advent of the typical form factor and design of consumer-oriented mobile devices, such as iPhone and Android-powered phones, it is possible for the programming model, code languages, run-time environment, and display technologies to converge so that the ideal of “write once, run everywhere” (though usually on only a specific manufacturer’s device) is slowly becoming a reality. A common design and set of capabilities means that it is much easier for developers to create applications that work well on that device, even if they are not directly portable to devices from other manufacturers.

Windows Phone 7 brings all the advantages of a standardized platform and a consistent developer experience to the Microsoft platform for devices from many different manufacturers. It is not a replacement for Windows Mobile, which continues to provide a powerful platform for a wide range of devices and application scenarios. Instead, Windows Phone 7 is a brand new mobile device that incorporates a comprehensive set of features necessary to build applications that satisfy the needs of business and consumers, to allow developers to easily create powerful interactive and attractive applications, and to reuse their skills and knowledge of existing modern development environments, such as the Microsoft Silverlight® and the Microsoft XNA® development platforms.

Windows Phone 7 incorporates the majority of features that users now expect to find on a mobile device, such as cloud service and media integration, easy and safe application installation, a stylish modern user interface (UI) that supports gestures and smooth animation, and device capabilities, such as location awareness, camera, sound recording, messaging, and multi-touch. Figure 1 shows the main features of a Windows Phone 7 device.



Windows Phone 7 provides a standardized platform that simplifies both design and development of applications that run on devices from multiple phone manufacturers.

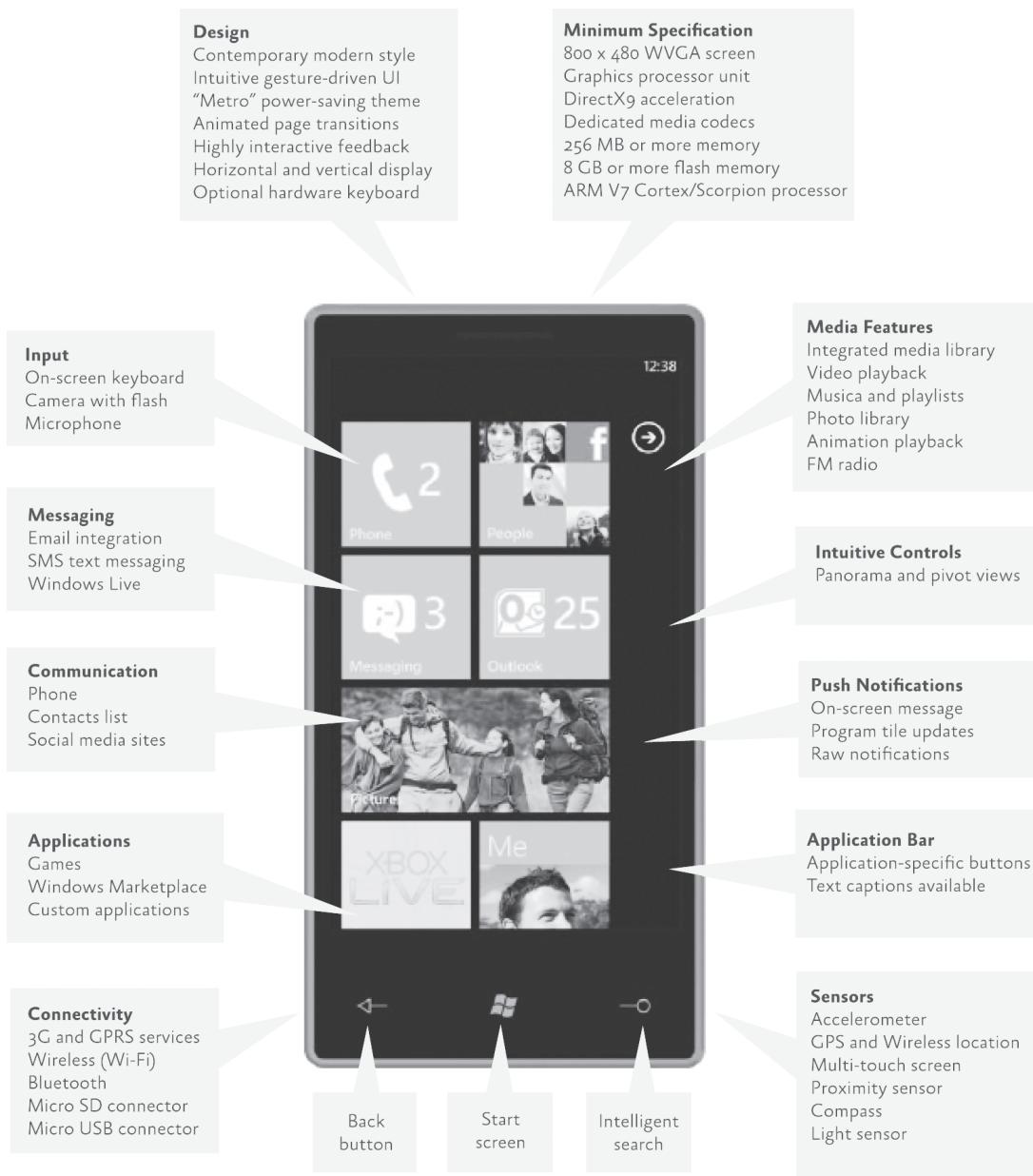


FIGURE 1
The features available on Windows Phone 7 devices

Windows Phone 7 is also an integrated part of the end-to-end application development and run-time story at Microsoft. The wide range of powerful Microsoft frameworks, applications, and services work with Windows Phone 7 to provide a consistent and complete environment for developing enterprise applications that extend the corporate presence to mobile workers, as well as to consumers.

In addition to integration with applications such as Microsoft Exchange, Windows Live® network of internet services, and Microsoft SharePoint® team services, developers can easily take advantage of reliable and scalable custom services that run in the cloud on the Windows Azure™ technology platform. Windows Phone 7 also uses services specifically designed to integrate with the device, such as the Location Service and Notifications Service provided by Microsoft, and it is likely that more of these types of services will appear in the future.

Windows Phone 7 also allows developers to easily distribute and sell their applications using a combination of the developer portal and the Windows Marketplace portal. Windows Marketplace provides a single consistent environment for users to obtain applications with the knowledge that they have been certified for use on the device and will properly integrate with it. In conjunction with the targeted advertisement opportunities available through the Microsoft Advertising Exchange for Mobile hub, this also allows developers to profit from creating Windows Phone 7 applications.

Data-driven Applications

Although a few simple applications may run wholly on the device without accessing remote services, almost all data-driven applications for both the enterprise and consumer market will connect to a remote system to obtain and upload data. A growing trend is to expose these data services from a cloud-based host, which provides resilience, performance, lower cost, elasticity to meet varying demand, and the possibility of global reach through multiple geographically separated installations of the service. In particular, because phones are relatively low-powered devices with limited facility for local data storage, using a remote or cloud service to drive the application can provide an experience comparable to far more powerful types of client devices.

Whereas in the past, most data was communicated using multiple proprietary formats, the common approach today is to use standard protocols and standard communication architecture styles, which allows developers to mix and match services more freely; in addition, it reduces dependencies between the device and the data service. The programming and run-time environments available on most modern



Data-driven applications for enterprise and consumer markets will need to connect to remote systems that expose the data they require.

mobile devices support these formats or provide tools and frameworks that make it easy to consume them.

The result of this remote service integration is a very persuasive scenario that allows mobile devices and their users to take advantage of a huge number of services. Examples range from social networking and location-aware services, to productivity applications such as office-style applications and information search, to enterprise messaging and line-of-business (LOB) applications.

Developing for the Windows Phone 7 Platform

The Microsoft Windows Phone 7 provides a platform for mobile computing on the phone. It has been designed to make programming applications, and sharing and reusing code, much easier by taking advantage of two common technologies that already have a comprehensive user base: Silverlight and XNA:

- **Silverlight.** This is derived from the Windows Presentation Foundation (WPF) technology. WPF uses Extensible Application Markup Language (XAML) to define the UI and code that uses the .NET Framework classes to implement the functionality of the application. Silverlight uses a reduced set of both WPF and the .NET Framework. It was designed to provide a lightweight, compelling, attractive, and interactive UI; and broad capabilities that take maximum advantage of the features of the device. It can be delivered to a Web browser and executed in a hosted control; or, like with Windows Phone 7, it can be packaged as a XAP file and run natively on devices that support it. Silverlight provides a development environment that is ideal for business applications and consumer applications that display, collect, and process information. For more information about Silverlight, see <http://www.silverlight.net/>.

Note: *The implementation of Silverlight used in Windows Phone 7 is a subset of the full Silverlight functionality, with the addition of some phone-specific APIs. For more details, see "Differences Between Silverlight on Windows and Windows Phone" on MSDN® ([http://msdn.microsoft.com/en-us/library/ff426930\(VS.96\).aspx](http://msdn.microsoft.com/en-us/library/ff426930(VS.96).aspx)).*

- **XNA.** This is composed of software, services, resources, and communities that focus on enabling developers of games and media-rich applications to be successful on Microsoft gaming platforms. These include the XNA Game Studio Express, the Microsoft DirectX® application programming interface display mechanism, and other XNA gaming tools and technologies; as



Every Windows Phone 7 phone is guaranteed to have a powerful graphics processor unit (GPU), GPS and wireless-driven location capabilities, an accelerometer, an 800 x 480 resolution multi-touch screen, a camera, a microphone, support for "push" notifications, an FM radio, and—of course—phone and text messaging capabilities.

Windows Phone development uses the .NET Framework and tools such as Visual Studio that most developers are already familiar with.

well as tutorials, white papers, samples, and more. XNA gives developers the capability to directly access features of the device such as the video and sound systems, where this is necessary to provide the performance required for highly interactive gaming and associated types of applications. For more information about XNA, see the XNA developer portal site at <http://msdn.microsoft.com/en-us/aa937791.aspx>.

Note: *The implementation of XNA used in Windows Phone 7 is a subset of the full XNA functionality. For more information, see "XNA Framework and Silverlight" on MSDN at ([http://msdn.microsoft.com/en-us/library/ff607286\(VS.96\).aspx](http://msdn.microsoft.com/en-us/library/ff607286(VS.96).aspx)).*

For more information about the differences between Silverlight and XNA and about how you can use XNA framework routines from a Silverlight application, see Appendix B, "Silverlight and XNA in Windows Phone 7."

To enable a consistent application platform, all Windows Phone 7 devices implement at least a minimum set of hardware and software features that provide an acceptable and consistent user experience across devices from different manufacturers. This simplifies development by allowing the application to run on any certified Windows Phone 7 device without the developer worrying about whether there is sufficient memory, how touch and orientation are supported, or whether additional features such as a camera or GPS are available.

Chapter 2, "Designing Applications for Windows Phone 7," describes the development platform provided by Windows Phone 7 in more detail and discusses the factors you must keep in mind when designing applications for the phone.

In addition, the enforced distribution and certification of third-party applications through Windows Marketplace ensures that the downloaded software meets a set of acceptable minimum criteria for quality and compatibility to give a consistent user experience. Chapter 7, "Interacting with Windows Marketplace," provides information about how you can submit your applications to Windows Marketplace for distribution to end users.

RESOURCES FOR DEVELOPERS

The primary tool for developing Windows Phone 7 applications is the familiar Visual Studio. Debugging, coding, testing, and so on, can all be performed from within the integrated development environment (IDE), just as you would in any other Microsoft-based application development scenario. The Windows Phone Developer Tools that you can download and install include a phone emulator that allows you to develop applications without having a physical phone, although you

should test your application on a real physical device before you deploy it.

The emulator is a virtual machine that runs a copy of the actual phone operating system, so it is an accurate simulation of the real phone runtime in all respects except for performance (the emulator will generally perform more slowly than a real device) and sensor availability (for example, location and acceleration emulation are not provided by the phone emulator).

Developing for Windows Phone is essentially the same as developing against the .NET Framework using the Silverlight and/or XNA libraries, although you can use only the C# .NET language to create your applications in the release version (Microsoft Visual Basic® will be supported in an upcoming release). The phone runtime includes specific libraries that provide access to capabilities such as sensors, location services, media capabilities, messaging services, the camera, and Windows Marketplace.

Resources for Silverlight Developers

To develop Silverlight applications for Windows Phone 7, you can use Visual Studio 2010, Visual Studio 2010 Express Edition, and the developer tools available for download from Microsoft. The main tool set is the Windows Phone Developer Tools, which includes specially adapted versions of Visual Studio 2010 Express Edition and Expression Blend® design software 4.0 (for developing the UI), integration components and templates for Visual Studio 2010 (which you can use instead of installing Visual Studio 2010 Express Edition), Silverlight tools, run-time debugging support, and an emulator for Windows Phone 7. You can obtain the Windows Phone Developer from the Windows Phone and Xbox 360® App Hub at http://create.msdn.com/en-us/home/getting_started.

Note: You will also need to install the Windows Phone Developer Tools October 2010 Update from <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=49B9DoC5-6597-4313-912A-F0CCA9C7D277&displaylang=en>.

For a simple walkthrough that shows how you can get started building Silverlight applications, see “How to: Create Your First Silverlight Application for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402526\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402526(VS.92).aspx)). A series of videos that describe development of Windows Phone 7 applications, for both Silverlight and XNA, is available from “Windows Phone 7 Jump Start Training” on the Windows Phone Developer Blog (http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/08/17/windows-phone-7-jump-start-training.aspx).



The emulator is very close to the real phone, but it is a good practice to test against a real device.



Silverlight for the phone is a subset of Silverlight 3.5 plus APIs for phone-specific capabilities.

For information and guidelines about the style recommended for Silverlight applications, see *Windows Phone UI Design and Interaction Guide* (a PDF document) available from the Microsoft Download Center (<http://go.microsoft.com/?linkid=9713252>).

Appendix A of this guide, “Tools, Frameworks, and Processes,” also contains information to help you get started developing Silverlight applications for Windows Phone 7.

Resources for XNA-Based Application Developers

This guidance concentrates on applications built using Silverlight. However, there are many resources available for developers who use XNA to build applications. The Windows Phone Developer Tools include XNA Game Studio, which developers can use to create XNA applications. For information about Game Studio, see “XNA Game Studio 4.0” on MSDN (<http://msdn.microsoft.com/en-us/library/bb200104.aspx>).

For a simple walkthrough that demonstrates how to create XNA applications, see “How to: Create Your First XNA Framework Application for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff472340\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff472340(VS.92).aspx)). For a series of videos that describe development of Windows Phone 7 applications, see “Windows Phone 7 Jump Start Training” in the Windows Phone Developer Blog (http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/08/17/windows-phone-7-jump-start-training.aspx); the videos include coverage of XNA application development. For more useful information about XNA development, see XNA Creators Club Online on the Microsoft App Hub website (<http://creators.xna.com/en-US/>).

Resources for Web and Service Developers

Windows Phone 7 includes comprehensive web browsing capabilities. In addition, ASP.NET allows you to detect the client device type making a request so that you can provide the content in a format and style appropriate to the device. For information about how Windows Phone 7 supports web browsing, and how to maximize your application experiences on Windows Phone 7, see *Designing Web Sites for Phone Browsers* (a PDF document) available from the Microsoft Download Center (<http://go.microsoft.com/?linkid=9713253>).

For information about mobile device support in ASP.NET, see “ASP.NET for Mobiles” on the Microsoft ASP.NET website (<http://www.asp.net/mobile/>) and “Walkthrough: Adding Support for Devices” on MSDN ([http://msdn.microsoft.com/en-us/library/wa642f6e\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/wa642f6e(VS.71).aspx)).

Mobile phone applications aimed at the enterprise and for consumer use will usually require access to remote services to be able to

obtain information, exchange data, or synchronize content. Typically, on the Microsoft platform, you will implement these services using the Windows Communication Foundation (WCF) technology that is part of the .NET Framework. For information about WCF, see "Windows Communication Foundation" on MSDN (<http://msdn.microsoft.com/en-us/netframework/aa663324.aspx>).

A common approach for delivering data or exposing services to mobile devices such as Windows Phone 7 is by using the Representational State Transfer (REST) architectural style for the service. For information about REST-based services, see "REST in Windows Communication Foundation (WCF)" on MSDN (<http://msdn.microsoft.com/en-us/netframework/cc950529.aspx>) and "An Introduction To RESTful Services With WCF" in *MSDN Magazine* (<http://msdn.microsoft.com/en-us/magazine/dd315413.aspx>). For a definition of the way that REST-based services work, see "Canonical REST Entity Service" on the Microsoft .NET website (<http://code.msdn.microsoft.com/canonicalRESTEntity>).

In other cases, such as exchanging small volumes of data between the client and server or calling specific methods on the server, you may decide to use SOAP instead. For more information about WCF, see "Getting Started with WCF" on MSDN (<http://msdn.microsoft.com/en-gb/library/ee354180.aspx>).

Terminology

The following list describes the technologies, patterns, frameworks, applications, and other terms commonly associated with developing for mobile devices such as Windows Phone 7:

- **Accelerometer.** A device capability that measures acceleration in three planes, and the direction of the force of gravity that indicates the attitude of the device. Can be used to detect movement, including gestures such as shaking the device.
- **Advanced Encryption Standard (AES).** A symmetric encryption algorithm available on Windows Phone 7 that can be used to encrypt data.
- **AllKeys API.** Allows your programs to request that all key presses be sent directly to the requesting application. Usually, some buttons are intercepted by the operating system for its own use, but games and input-intensive applications may want access to these buttons for their own use.
- **Application bar.** The small area at the bottom of the screen that contains buttons for commonly used functions of the currently executing application. This optional bar shows a set of icons by default, but the user can expand it to also show the text

captions for the buttons. An application can show a maximum of four buttons on the application bar.

- **Application Verifier (AppVerifier).** A software test tool used to check the stability of the application and detect common programming mistakes associated with memory management. AppVerifier can detect and pinpoint memory leaks, handle leaks, and heap corruption.
- **Atom Publishing Protocol (Atom Pub).** An XML-based format for data that uses a Representational State Transfer (REST) architectural style where the names of collections and entities are defined in the query string of a request.
- **Cloud services.** Services that run in one or more remote data centers on specially designed hardware and a virtual runtime (fabric) that provides very high availability, reliability through multiple instances, performance, and scalability. Generally, cloud services are cost-effective ways to provide local and global access to applications and services without requiring the investment, expertise, maintenance, administrative overhead, and run-time cost of an on-premises server infrastructure.
- **DPI.** Dots-per-inch (DPI) is a measure of video dot density, in particular the number of individual dots that can be placed within the span of one linear inch (2.54 cm.). It is related to image resolution.
- **Elasticity.** The capability of a service, such as a cloud-based application, to be expanded by adding service instances and shrunk by removing service instances to more closely match the current requirements and load. This minimizes cost by not requiring the acquisition of sufficient hardware, software, and bandwidth to satisfy peak demand that is then idle at other times.
- **File-based applications.** File-based applications store data in a file and often work as editors for specific file formats. Examples include word-processors and spreadsheet applications.
- **GAPI.** Game API (GAPI functions) provides solutions for developers who want to write high-performance, real-time games on devices running on Windows Mobile-based devices.

Note: *GAPI was deprecated in Windows Mobile 6.5, so developers should use the AllKeys, DirectDraw, and Direct3D® Mobile APIs instead.*

- **GPS.** See “Location service.”

- **Hash-based Message Authentication Code** (HMAC). A keyed hash algorithm that can be used to create a non-reversible hash value for data. Two versions that use a Secure Hash Algorithm (SHA) are available on Windows Phone 7: HMACSHA1 and HMACSHA256.
- **Home screen**. The point from which users start most of their applications, and the point that the user can return to with a press of the **Home** button. In Windows Phone 7, developers can customize the Home screen by providing tile images that the user can add to the page.
- **Ink Presenter control**. A primitive control that can display strokes within a **Canvas** control.
- **Location service**. A phone-based service that the phone uses to discover its geographical location based on a series of factors. These factors can include the built-in GPS capability, and triangulation of public Wi-Fi networks and phone signal towers.
- **Managed code**. Managed code is code compiled for the .NET Framework. Managed code is often written in Visual C# or Visual Basic .NET.
- **Metro theme**. The standard theme used in Windows Phone 7, and recommended for applications you develop so that they integrate seamlessly with the operating system and other applications. The theme is designed to provide a modern UI that is easy to use, while minimizing power consumption on the phone.
- **Microsoft Push Notification System** (MPNS). A service that allows developers to send notifications to users' phones that are displayed even if the application is not running, or will update a tile on the **Start** menu of the phone to indicate that new information is available.
- **Model-View-ViewModel (MVVM)**. A design pattern particularly suited to Windows Presentation Foundation (WPF) and Silverlight applications. It uses views to implement the UI, a model that holds the data for the application, and view models that access the model and populate the views using the powerful data-binding capabilities of WPF and Silverlight. It makes it easier to decouple sections of the application, develop the components, test them, and maintain the application post deployment.
- **MultiScale Image control**. A control that can be used to display a very large image without requiring it to be fully downloaded to memory. It uses a collection of multiple sub-images at

different resolutions. The control provides methods to pan and zoom over the image.

- **Non file-based application.** An application that does not need to store any data in a file. For example, a calculator application.
- **Notifications.** Messages that are sent from a server to the phone through the Windows Notification Service and can be received even when the application is not running. Notifications can display a message at the top of the screen or change a program tile on the Home screen. It is also possible to send raw notifications that the application itself must handle. Users must register to receive notifications on their phone.
- **Open Data Protocol (OData).** A web protocol for querying and updating data. OData builds on web technologies such as HTTP (<http://www.w3.org/Protocols/>), Atom Pub, and JavaScript Object Notation (JSON <http://json.org/>) to provide access to information from a variety of applications, services, and stores.
- **Panorama control.** A control that offers a way to view controls, data, and services by using a long horizontal canvas that extends beyond the confines of the screen. Areas of this canvas are viewed as smaller individual areas by panning sideways. Effectively, it provides a viewport over the larger area. The user can scroll the window horizontally across the larger view to see the complete content.
- **Pivot control.** A control that provides individual views over related sets of data. Typically, the individual views are related; for example, views that show new items, favorite items, and recently viewed items from the users stored photos. This control can be used to present filtered views of large data sets or to switch between application views. The user pans horizontally to navigate between the views.
- **Prism.** A free utility library from the Microsoft patterns & practices group. The components in this library can help developers build applications for WPF, Silverlight, and Windows Phone that are easier to maintain and update as requirements change.
- **Proximity sensor.** A device capability that detects whether the phone is close to an object, such as the human body. The operating system uses this to change the behavior when the device is used in phone mode.
- **Push notifications.** See “Notifications.”
- **Representational State Transfer (REST).** An architectural style used to expose data from services that allows simple access to,

and optional manipulation of, information without requiring the server to maintain state between requests.

- **Rfc2898DeriveBytes.** An algorithm available on Windows Phone 7 that takes a password, a salt, and an iteration count, and then generates encrypted keys of virtually unlimited length.
- **Secure Hash Algorithm (SHA).** An algorithm for creating a non-reversible hash value for data. Two versions are available on Windows Phone 7: SHA1 and SHA256.
- **Simple Object Access Protocol (SOAP).** A protocol used to allow clients to execute methods on a remote server to obtain information or perform processing. It is an XML-based format that packages the data in a SOAP envelope. SOAP messages carry both payload and metadata, which provides for transport-independent security and reliability and can add layers of functionality to the communication, such as atomic transactions or cross-system activity tracing.
- **Soft input panel.** An on-screen input method (software keyboard) for devices that do not include a hardware keyboard.
- **SQL Azure.** A solution that offers multiple highly reliable, scalable, cloud-based relational databases where you can store data used by your services and applications. Databases are provisioned on-demand and are charged according to usage. SQL Azure is part of the Windows Azure platform.
- **Secure Sockets Layer (SSL).** A cryptographic protocol for securing communication between clients and services over a TCP/IP network. Often described as HTTPS, it operates by default through port 443 instead of port 80. The more recent implementation is Transport Layer Security (TLS).
- **Transport Layer Security (TLS).** A cryptographic protocol for securing communication between clients and services over a TCP/IP network. Provides better overall security than SSL, and is often used to secure messaging communication as well as web requests and responses.
- **Tombstoning.** The process whereby the currently executing application must stop to allow another application or device feature (such as an incoming phone call) to execute. The application must save its state so that it can either be continued from where it left off or be terminated without losing data.
- **Touch input.** Windows Phone 7 supports touch and gestures for interaction with the phone and applications running on it. It recognizes up to four touch points and gestures such as tap, double-tap, pan, flick, pinch, stretch, and touch hold.

- **Windows Azure.** A highly reliable, scalable, cost-effective, and elastic run-time environment within the cloud that you can use to run applications that expose services or accept web requests.
- **Windows Azure Web Role.** An instance of a web service running within Windows Azure that accepts connections from clients and returns responses to them while performing some processing that is appropriate to the Windows Azure service that hosts the role.
- **Windows Azure Worker Role.** An instance of an application service running within Windows Azure. Worker roles run asynchronously and communicate with web roles through queues or messages; generally to perform background tasks or other processing that is not part of the web role.
- **Windows Marketplace.** Delivers an end-to-end solution for end-users to discover, purchase, and download Windows-based applications for the desktop and mobile devices. This includes an end-user experience on the device, a website, as well as a self-service portal for developers to submit their applications for listing in Windows Marketplace.
- **Windows Mobile Device Center.** An application that you can use together with Windows Vista® operating system or Windows 7. It offers device management features and lets you synchronize data between a Windows Mobile-based device and a computer. Windows Mobile Device Center replaces ActiveSync® technology, which was available on earlier desktop operating systems.
- **Zune Desktop Client.** A desktop synchronization application that is primarily used by consumers for synchronizing music and videos, but it is also used by developers to upload applications to the phone during the development and testing processes.

More Information

For a comprehensive overview of the Windows Phone platform and how it is designed to support many different types of application, see the video presentation “Overview of the Windows Phone 7 Series Application Platform” on the MIX website (<http://live.visitmix.com/MIX10/Sessions/CL13>).

To see a more detailed description of the underlying mechanisms and architecture of the Windows Phone 7 operating system and how you can interact with services, see the video presentation “Windows Phone Application Platform Architecture” on the MIX website (<http://live.visitmix.com/MIX10/Sessions/CL18>).

Microsoft provides a portal for developers to help them get started writing applications for Windows Phone 7. This includes information about the operating system and the device capabilities, developer resources, forums, tools for working with Windows Phone 7, and access to a market place for selling and distributing applications. You can find this portal on the Microsoft App Hub website (<http://developer.windowsphone.com/windows-phone-7/>).

In addition, the MSDN site includes full documentation for developing for Windows Phone 7. See “Windows Phone Development” ([http://msdn.microsoft.com/en-us/library/ff402535\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402535(VS.92).aspx)).

A useful frequently asked questions (FAQ) for getting started with Windows Phone 7 is available on the Microsoft Windows Phone forum (<http://social.msdn.microsoft.com/Forums/en-US/windowsphone7series/thread/2892a6f0-ab26-48d6-b63c-e38f62eda3b3>).

General information about the Windows Phone 7 platform is available from “Application Platform Overview for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402531\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402531(VS.92).aspx)).

BLOGS, CODE SAMPLES, TRAINING KITS, AND WINDOWS MARKETPLACE

Additional documentation and code samples are available to help you learn about both Silverlight and XNA on Windows Phone 7. See the following resources:

- “Code Samples for Windows Phone” on MSDN:
[http://msdn.microsoft.com/en-us/library/ff431744\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)
- “Windows Phone 7 Developer Training Kit” on Channel 9:
<http://channel9.msdn.com/learn/courses/WP7TrainingKit/>
- Windows Phone Team Blog:
http://windowsteamblog.com/windows_phone/
- Windows Phone Developer Forums:
<http://social.msdn.microsoft.com/Forums/en-US/category/windowsphone>
- “Windows Marketplace Frequently Asked Questions” on the Microsoft App Hubs website:
<http://developer.windowsphone.com/Help.aspx>
- “Windows Marketplace Registration” on the Microsoft App Hubs website:
<http://developer.windowsphone.com/Signup-Create-Account.aspx>

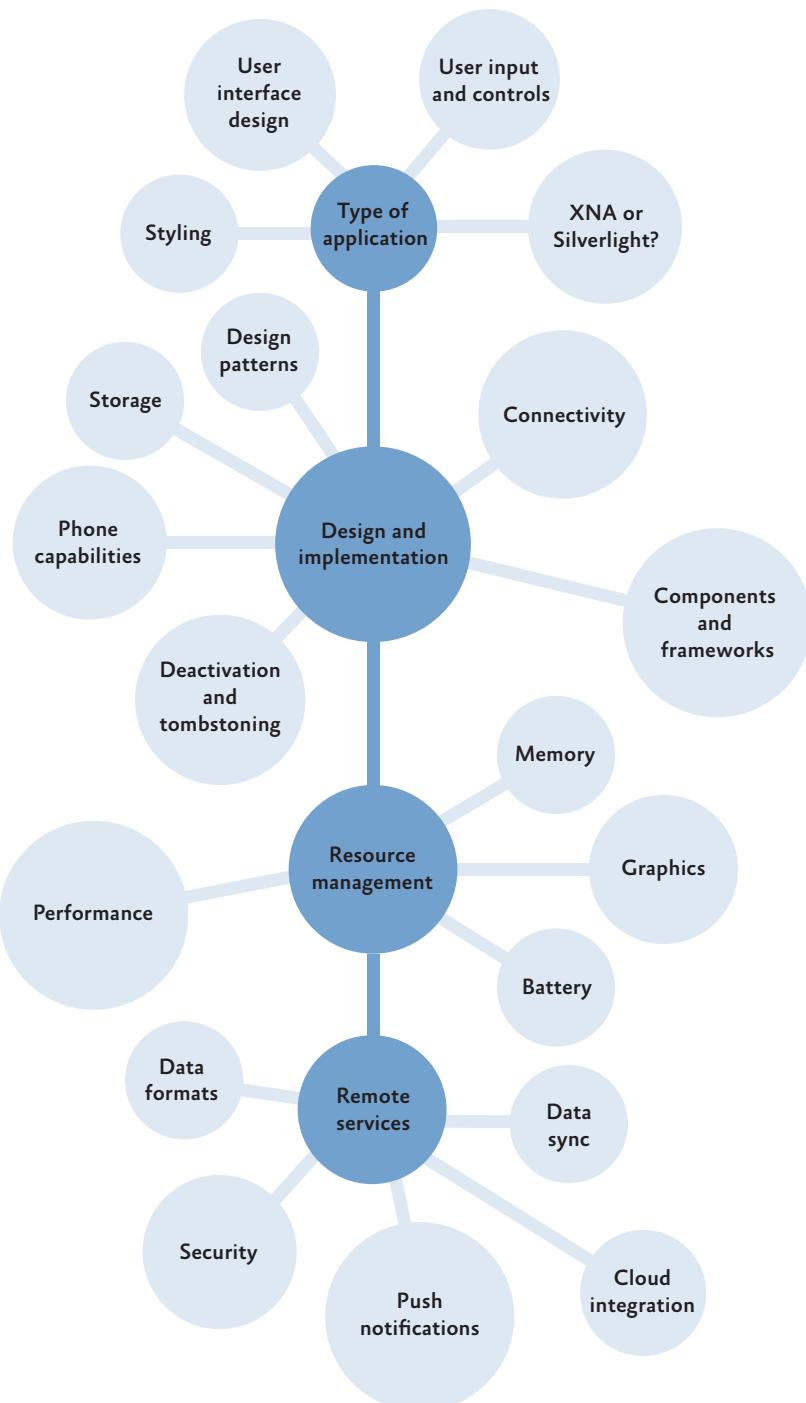
These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Windows® Phone 7 is a new and very versatile platform for building mobile client applications. However, when you design applications for Windows Phone 7, you must be aware of some of the inherent limitations placed on your applications by facets of the environment, such as the form factor, hardware resources, battery life, and intermittent connectivity common to all such devices.

This chapter describes the differences between applications that work well on mobile devices, such as Windows Phone 7 and applications designed for the desktop, along with the factors you should consider when designing for mobile devices. It will help you to understand the development and run-time platform of Windows Phone 7 devices. It will also assist you in making appropriate decisions on a range of factors, such as user interface (UI) design, storage, connectivity, data formats, and maximizing performance.

Basic Design Considerations

Before you examine the scenarios and considerations for mobile phone applications, use Figure 1 to frame the main areas of concern. All of these areas are covered in this chapter, with additional information available throughout the subsequent chapters and appendices of this guide.

**FIGURE 1**

The basic design considerations for mobile phone client applications

TYPE OF APPLICATION

The initial decisions you must make involve the type of application you will build, and how well this type of application is suited to a mobile device such as Windows Phone 7. You should also consider how you will implement your vision using a style and interactivity pattern that matches the built-in applications on the device, and how you can use the types of controls available on the phone to build a usable interface. In Windows Phone 7, the development platform you choose (Microsoft® Silverlight® browser plug-in or XNA®) will influence the application design and the types of controls available. For more information, read the sections “Mobile Phone Client Applications,” “User Interface Design and Style Guidelines,” and “User Input Considerations” later in this chapter.

DESIGN AND IMPLEMENTATION

Before you start to design and build your application, you should make some basic decisions about the way you will use techniques such as test-driven development (TDD), design patterns such as Model-View-ViewModel and Dependency Injection, and whether you will take advantage of third-party components or custom frameworks that match your requirements. You must also consider how you will implement data storage on the device, whether you will take advantage of phone capabilities such as the camera and Global Positioning System (GPS), and how you will manage connectivity to the network. In addition, you must be aware of how Windows Phone 7 devices may deactivate and tombstone your application. Although you do not have to finalize all of these decisions before you start, thinking about them now can help to stabilize your design and reduce the number of dramatic changes required later. For more information, read the sections “Storage Considerations,” “Connectivity Considerations,” and “Application Deactivation and Tombstoning” later in this chapter, and the discussion of the Model-View-ViewModel and Dependency Injection patterns in Chapter 4, “Building the Mobile Client.”

RESOURCE MANAGEMENT

Designing and building applications for small mobile devices such as the phone requires you to be aware of the resource limitations inherent in these types of devices. You should aim to make your application behave as a “good citizen” on the phone by optimizing performance and resource usage to provide the best experience for users and by avoiding any behavior that would negatively affect other features and applications on the phone. You must pay particular attention to allocating and releasing resources you use, minimizing battery drain, and maximizing performance on lower-power processors, such as those used in mobile phones. You should also understand how to take

full advantage of the graphics processor in the device. For more information, read the section “Resource Management and Performance” later in this chapter and Chapter 5, “Using Services on the Phone.”

REMOTE SERVICES

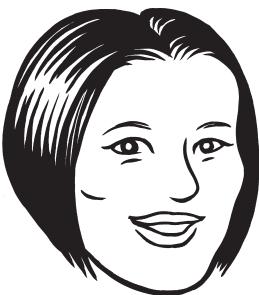
Finally, you must consider how you will interact with remote services that provide the data used by your application. You can interact with many types of services, including cloud services such as Windows Live® network of Internet services, and the Location Service and Notifications Services that are specifically designed to integrate with Windows Phone 7 devices. Technically, the format of the data you use to drive your application is irrelevant, though you should consider interoperability with other devices, communication security, and whether you need to perform data or file synchronization between the device and the server. However, it is vitally important to consider the bandwidth utilization, transfer volumes, and the costs of connectivity for your chosen approach. For more information, read the section “Data Formats and Accessing Remote Services” later in this chapter; Chapter 5, “Using Services on the Phone;” and Chapter 6, “Connecting with Services.”

Mobile Phone Client Applications

The physical properties of all mobile phones as a platform for building client applications apply certain limitations to the kinds of application you can build. For example, the device is unlikely to have access to large amounts of memory and storage, which prevents you from building applications that store large volumes of data. In addition, the likelihood of connectivity to the network being interrupted, and the lack of bandwidth available over some mobile connections, means that your applications must be able to either work with no connectivity or be able to manage interruptions and resynchronize when a connection is available.

The following are some of the differences between a desktop or notebook computer and a mobile phone device:

- **Processing power, memory, and storage.** Mobile devices typically use lower power CPUs, have very limited memory, and have much less storage space available.
- **Display size.** Mobile devices typically have a small screen. In Windows Phone 7, the maximum screen resolution is 800 x 480 pixels. The screen display can also rotate to match the orientation of the device.
- **Power consumption.** The small battery in mobile devices means that application must exert the minimum possible drain on the



There are significant differences between the phone and the desktop in terms of device resources and capabilities.

battery. Techniques such as minimizing use of the Location Service and minimizing network activity can assist in achieving this.

- **Application execution.** Mobile devices usually allow only a single foreground application to execute in order to minimize power consumption and maximize performance. Mobile phone applications must be designed to preserve their state when they are interrupted or terminated by the operating system.
- **Connectivity.** Mobile devices often experience periods of limited or no connectivity as the user moves from one location to another. The bandwidth available and the cost of data transfer can vary considerably, depending on the type of connection that is available in each location and across different geographical regions.
- **User interface.** Mobile device applications are typically operated using a touch screen. The location, spacing, and types of controls that are suitable for a touch-based interface differ from those in a desktop or notebook application that relies on a mouse and keyboard.

For information about the requirements and restrictions for applications that you upload to Windows Marketplace, including the maximum size and the performance metrics, see Chapter 7, “Interacting with Windows Marketplace.”

SUITABLE APPLICATION TYPES FOR MOBILE PHONES

You should also consider how and when the applications you build for a mobile phone will be used. In most cases, the user will be on the move, instead of sitting at a desk, and perhaps not in an ideal location for interacting with an application. Other factors, such as ambient lighting, noise, movement and vibration, and even interaction with other people, are common when using a mobile device.

Applications that are not likely to be successful on a mobile phone are those that have a complex interface, require multiple input steps to complete a task, or are unintuitive to use when adapted to a small screen. Users expect to be able to perform tasks quickly and without error in any environment, and they expect to be able to return to the application after interruptions, such as an incoming phone call.

The types of consumer-oriented applications that work well on mobile phones are entertainment applications such as games, music players, photo viewers and video players, and social interaction applications that perform tasks such as sending messages, updating status information, or posting photos to a website. For business or



Not all types of applications work well on a mobile phone. Complex applications, and applications with a busy UI that demands extreme concentration, are unlikely to be easy to use on a phone.

enterprise oriented applications, the following are common scenarios that lend themselves well to the phone form factor:

- Messaging applications, such as email and SMS text messaging
- Collaboration applications that allow users to access and modify documents from the phone
- Meeting planners and calendaring applications
- Task-oriented applications for activities such as booking facilities and travel planning, service and schedule notifications, completing expense forms, obtaining approvals, or simple data-entry tasks that can be accomplished using the limited UI offered by the phone platform
- Specialized applications for specific industries, such as those used for viewing stock availability, for warehouse order picking, for placing sales orders, or for updating data in response to changing requirements
- Mobile-oriented and location-aware applications, such as those used for satellite navigation, delivery confirmation, surveying, or tracking vehicle movements

SILVERLIGHT AND WINDOWS PHONE 7

In enterprise scenarios, and in many consumer scenarios other than media-rich applications and games, Silverlight is the optimum choice of development platform for Windows Phone 7 applications. The Tailspin Surveys example application described in this guide is a Silverlight application, although it does use interop to XNA to implement some features that are not accessible from Silverlight.

Windows Phone 7 supports almost the full capabilities of Silverlight 3, though there are some minor differences in the way it works, compared to Silverlight hosted in a web browser. For example, Windows Phone 7 does not support printing, and most of the visual controls are different to suit the UI requirements of the device, but the core programming platform is almost identical. If you are familiar with Silverlight, you will be immediately productive on Windows Phone 7.

UI design for the phone can be accomplished in Microsoft Visual Studio® 2010 development system and in Microsoft Expression Blend® 4 design software, both of which support creating Silverlight applications for Windows Phone 7.

In addition, tools and frameworks are increasingly available for working with and extending Silverlight on Windows Phone 7. These include testing tools, additional controls, and frameworks for implementing common design patterns. Some examples are the patterns & practices Prism Library, Ninject (see <http://ninject.org/>), and Moq (see <http://code.google.com/p/moq/>).



If you already know Silverlight, you will quickly be productive on Windows Phone 7.

This guide demonstrates how you can use well-known design patterns that are well suited to Silverlight, such as Model-View-View-Model and Dependency Injection. Common development techniques, such as data binding for UI controls, and test driven development can be applied to Windows Phone 7 applications in the same way as for desktop and server applications.

Note: *For information about developing Windows Phone 7 applications with Silverlight, see Appendix A, "Tools, Frameworks, and Processes." For information about the major differences between Silverlight and XNA, see Appendix B, "Silverlight and XNA in Windows Phone 7."*

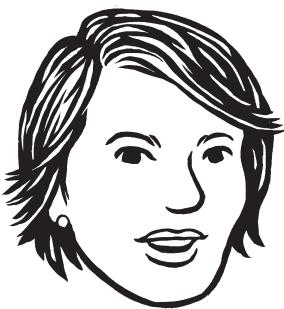
However, there are some challenges when using Silverlight on Windows Phone 7 compared to the experience when using it in a web browser. For example, multitasking capabilities are limited, and the program itself cannot run in the background. Events such as a phone call arriving on the device will stop your application from running, and you must preserve state so that your application can correctly resume when required or can gracefully exit if the system requires the memory or resources it is using.

Design Considerations for Windows Phone 7 Applications

In addition to deciding on the type of application you will build and the platform (such as Silverlight or XNA) that you will use, you must consider the capabilities of the device and understand how these may affect your design. The following are the major factors:

- UI design and style guidelines
- Application deactivation and tombstoning
- User input considerations
- Storage considerations
- Connectivity considerations
- Security considerations
- Data formats and accessing remote services
- Resource management and performance
- Availability of components and frameworks

The next sections of this chapter explore these factors in more depth.



You must be aware of the style and UI guidelines when you design applications for Windows Phone.

One of the major advantages of following the Metro theme is that the color scheme it uses helps to maximize battery life.

USER INTERFACE DESIGN AND STYLE GUIDELINES

The Windows Phone 7 Series uses a style named Metro that is designed to combine harmonious, functional, and attractive visual elements while clearly directing end users to the content they want. You should attempt to follow the same visual style and interaction paradigm for your application as is used in the built-in applications and the interface of Windows Phone 7 to ensure that your application is intuitive and easy to use.

Microsoft provides a guide to the Metro design style and the recommended approach for creating a more consistent and fluid UI experience, including descriptions of standard input functionality within the UI framework, the software interaction elements, and the system-based controls. If you want to sell your application through Windows Marketplace, you must follow these design guidelines.

Remember that any images you use in your application should be suitable when the device theme is changed. For example, if you use a white icon or image and the user changes from the default black theme to the white theme, your image will not be visible. One approach is to use code to switch the selected image based on the current theme, as demonstrated in the Tailspin Surveys sample application that accompanies this guide.

For more information, see the guide, *UI Design and Interaction Guide for Windows Phone 7*; from the Microsoft Download Center (<http://go.microsoft.com/fwlink/?LinkId=183218>). For information about the Metro design and to learn about design and development recommendations for your applications, see the video presentation, *Windows Phone UI and Design Language*, on the Mix10 website (<http://live.visitmix.com/MIX10/Sessions/CL14>). For information about the resources (including the set of application bar icons) for Windows Phone 7 applications, see “Design Resources for Windows Phone” on MSDN® ([http://msdn.microsoft.com/en-us/library/ff637515\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff637515(VS.92).aspx)). For a set of Adobe Photoshop design templates that help you create attractive applications that match the Metro theme, see the Microsoft Download Center (<http://go.microsoft.com/fwlink/?LinkId=196225>).

You must also consider how you will localize your application to achieve certification by Windows Marketplace. For a list of the supported languages for Windows Phone 7 applications, see “Globalization and Localization Overview for Windows Phone” on MSDN at [http://msdn.microsoft.com/en-us/library/ff462083\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff462083(VS.92).aspx); this page also provides a useful guide to the use of cultures, encodings, localizability, and links to other related resources.

APPLICATION DEACTIVATION AND TOMBSTONING

Unlike a traditional operating system on a desktop or server computer, Windows Phone 7 does not run multiple applications concurrently. Only one application—the foreground application—can be active and running at any one time. When the user switches away from the foreground application, or when an external event causes another application to become the foreground application, the original application is deactivated in a process referred to as “tombstoning.” When the user switches back to the application, it is reactivated.

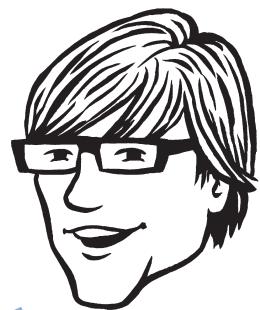
This means that if you want to preserve state (including page and control state) when your application stops being the foreground application, you must handle the events exposed by the operating system to save any state you require to implement a seamless resume experience (such as setting the contents of text boxes, a selected date, and so on). This saved state will be used to restore the application and page state when the application reactivates.

External events that may cause the application to be tombstoned include events such as a phone call being received; a system dialog box, such as the built-in photo selector, being displayed; or the phone lock screen being activated.

In addition, an application that is tombstoned may be completely removed from memory if, for example, the operating system detects that the device is running low on resources. In this scenario, when the user goes back to the application, it will restart. Therefore, you must make sure that you have preserved all persistent state to storage whenever your application is deactivated.

You handle the **Launching** event to retrieve state for your application from isolated storage on the device and/or from a remote service, as appropriate, when the user first starts your application from the phone’s Start screen. You also handle the **Closing** event that occurs when the user closes your application (or presses the Back button to navigate backward through the pages of the application past the application’s first page) to save any state your application requires to isolated storage on the device and/or to a remote service, as appropriate.

You must also handle the **Deactivated** and **Activated** events. The **Deactivated** event occurs before the application is tombstoned, and you should save any transient state to the phone application service so that it can be quickly reinstated when the application is resumed. The operating system provides two objects where you save that transient state. Save transient application data to the **PhoneApplicationService.State** property, and save transient page state to the **PhoneApplicationPage.State** property. There is a time limit for the **Deactivated** event to complete. The device may terminate the



When a phone call arrives or the user switches to another application, your application must save its state in case it is terminated by the operating system.

application if it takes longer than 10 seconds to save the transient state.

Note: *There is no guarantee that an application that was deactivated will be reactivated, so, in addition to the transient state, you should also save any persistent state to isolated storage or a remote service, as appropriate. Applications that require a large volume of persisted data may need to save that data incrementally while the application is running to ensure timely deactivation.*

The **Activated** event occurs when the user navigates back to your application using the Back button, or when a launcher or chooser that has interrupted execution to handle a phone call or to display the lock screen is deactivated. When your application is reactivated, you should retrieve the transient data from the **PhoneApplicationService.State** and **PhoneApplicationPage.State** property bags in the **Activated** event. As with the **Deactivated** event, there is a time limit for the **Activated** event to complete. The device may terminate the application if it takes longer than 10 seconds to reload the transient state and return the application to a running state.

Note: *Launchers and choosers allow other applications or operating system features to execute while your application is running. For example, the **PhoneCallTask** launcher activates when a phone call is received, or your application may use a launcher, such as the **PhotoChooserTask**, to select a photo from the device storage. For more information, see "Launchers and Choosers for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff769556\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769556(VS.92).aspx)).*

Figure 2 shows the life cycle, execution flow, and events you must handle to preserve state in a Windows Phone 7 application.

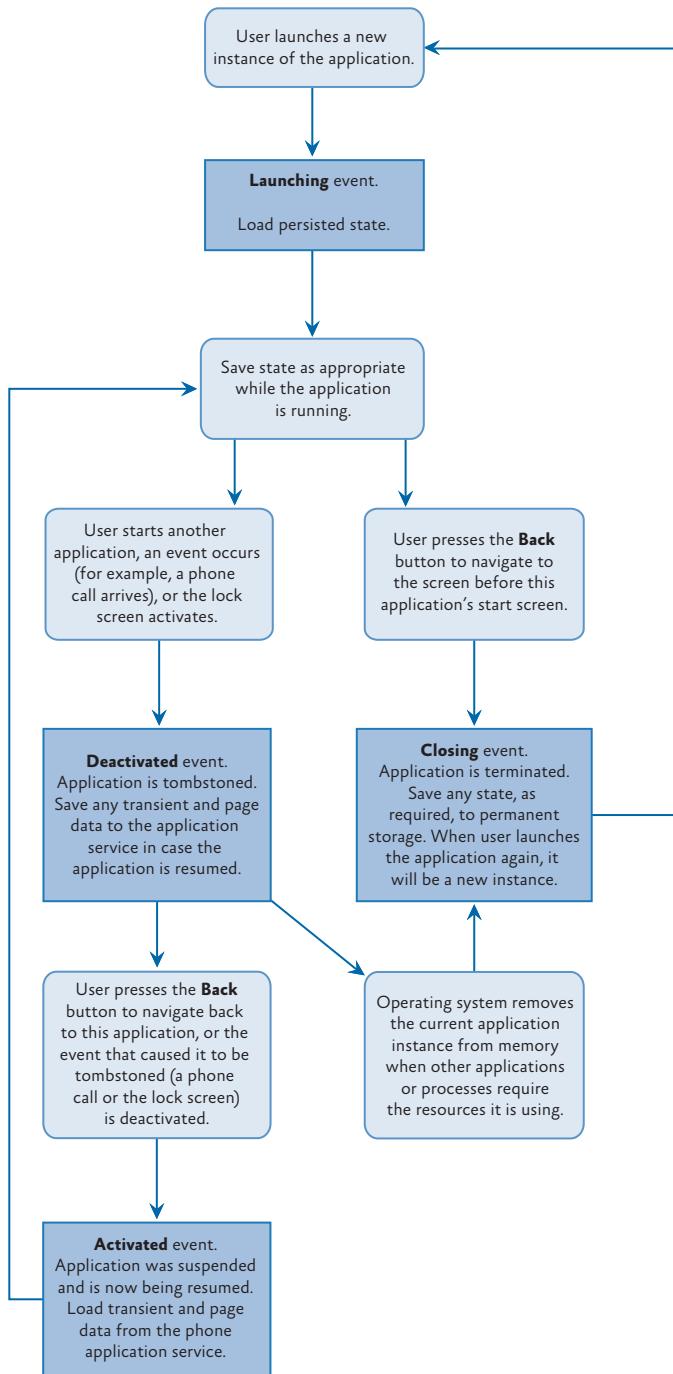


FIGURE 2
The life cycle and events for a Windows Phone 7 application



Designing UIs for Windows Phone 7 is very different from designing UIs for the desktop.

For a full explanation of the ways that you should handle the life cycle events in your application, see “Application Lifecycle” on Channel 9 (<http://channel9.msdn.com/learn/courses/WP7TrainingKit/WP7Silverlight/ApplicationLifetimeWP7Lab/>) and “Execution Model for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff769557\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769557(VS.92).aspx)). A useful resource that explains how to use the **PhoneApplicationService** class is “How to: Preserve and Restore Page State for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff967548\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff967548(VS.92).aspx)).

USER INPUT CONSIDERATIONS

The small screen on mobile devices and the requirement to support touch input present a challenge when designing UIs. Windows Phone 7 requires touch input and will not respond to pen-based or pointer input. In addition, the UI should be designed to work in both portrait and landscape modes. Some devices with a hardware keyboard will require the phone to be used in landscape mode for data entry tasks, as shown in Figure 3.

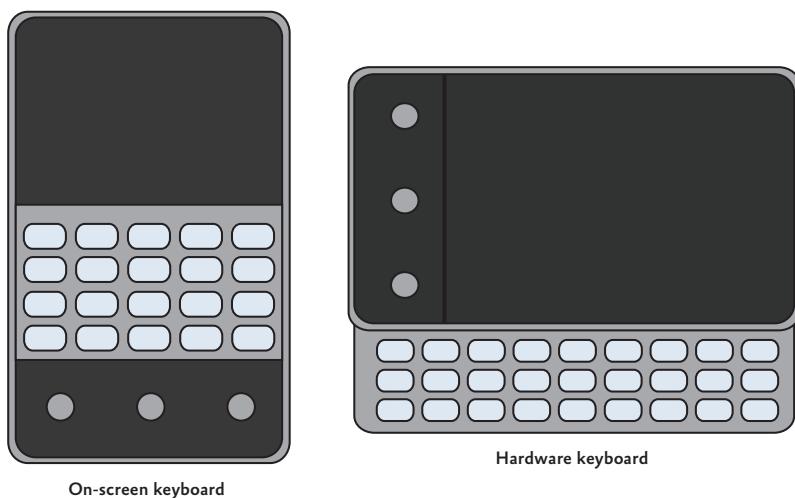


FIGURE 3
On-screen and optional hardware keyboard layouts in Windows Phone 7

The following are some of the things you should consider when designing the UI for a mobile phone application:

- Do not try to adapt an existing UI design directly from a web or desktop application. Start afresh by thinking about the flow of user interaction and how you can maintain simplicity and make the application intuitive for use in the phone environment.

- Consider the fact that the design must work in both portrait and landscape mode.
- Choose the appropriate types of controls for display and input. Special controls designed to work well on Windows Phone 7, and used in the Metro theme, include the Pivot and Panorama controls.
- Place menus and other option boxes at the bottom of the page so that the user's hand does not obstruct the rest of the page content. However, remember that the on-screen keyboard will obstruct a large proportion of the screen when the user selects a text control.
- Follow the Windows Phone 7 application design guidelines by ensuring that buttons and other controls that the user must interact with are large enough, and that they have enough separation to prevent accidental activation of the wrong one.
- Provide feedback when an action is initiated so that the user is not tempted to retry the operation. Use a visual or other indication to signify that the program has recognized the gesture and is responding to it.
- Validate input where appropriate, especially before initiating communication with the server or starting a resource-intensive process. This prevents invalid input from causing additional data transfers and unnecessary communication costs; it also prevents wasting processor time and increasing drain on the battery.
- Consider setting the **InputScope** property of text controls to simplify user interaction. This property determines the type of on-screen keyboard displayed for the text control. For example, setting **InputScope="Text"** displays a simple predictive text-enabled keyboard, while setting **InputScope="Url"** displays a keyboard containing the characters most used in URLs. For a list of supported values, see "InputScopeNameValue Enumeration" on MSDN ([http://msdn.microsoft.com/en-us/library/system.windows.input.inputscopenamevalue\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/system.windows.input.inputscopenamevalue(VS.95).aspx)).

STORAGE CONSIDERATIONS

Windows Phone 7 does not include a general-purpose file system or a local database. The only option for storing data on the phone is to use isolated storage via an API to access a virtual hierarchy of folders and files that is private to the application. Although third-party database mechanisms are likely to be available, you should consider whether the additional memory and resource overheads that a database will impose are necessary for your application. The majority of Windows Phone 7 applications will typically use only isolated storage.



Using serializable collections is a good way to store data in isolated storage on Windows Phone 7 devices.

Note: *In future releases, there may be a database available on the phone; or you may alternatively consider using a third-party database that you install with your application.*

When storing data in isolated storage, the most common approach is to store it as serializable collections. The Tailspin Surveys client application discussed in this book uses isolated storage to hold collections of surveys and their content.

Storage is limited on mobile devices when compared to desktop and notebook computers. The minimum specification for Windows Phone 7 devices is 256 MB of memory, though manufacturers are likely to install more than this. The phone can also accept extended memory of up to 8 GB using a flash card.

Isolated storage on the device is shared among operating system data, built-in applications, user content (such as downloaded media), and other applications that the user installs on the device. Being a “good citizen” is especially important on the phone because there is no limit to the amount of data that an application can store (there are no per-application storage quotas). This means that your applications must minimize their use of storage without causing a reduction in performance of the device and your application. The following are some recommendations for using storage on the device:

- Use the **DeviceExtendedProperties** class to see how much memory your application is using, and how much memory is installed and available. For more information about the **Device ExtendedProperties** class, see Appendix C, “Leveraging Device Capabilities.”
- Make sure that your application cleans up and releases storage when it is no longer required, especially when the application is tombstoned or terminated.
- Consider using caching to improve application performance, but only cache data that is regularly required and that does not affect the availability of isolated storage for other applications and services.
- Consider compressing data that is placed in storage, though keep in mind that this will require additional processing and reduce battery life. You may be able to use data formats that are more compact; for example you might store configuration information or other types of data in binary or JSON format instead of XML format.
- Ensure that sensitive data is encrypted, especially when it might be stored in removable media, such as a flash card.

Note: *Typically, you will see the following effects, depending on the serialization method you choose:*

XML: *Larger payload, higher CPU utilization*

JSON without compression: *Smaller payload, lower CPU utilization*

JSON with compression: *Smallest payload, higher CPU utilization*



CONNECTIVITY CONSIDERATIONS

When building traditional client-server and Web applications, a typical assumption is that connectivity between the server and the client will be available all the time. If the client cannot access the server, the application will usually stop working or produce an error.

However, when working with mobile devices, this assumption is no longer valid. The device may experience occasional loss of connectivity, depending on the availability of a sufficiently strong signal. It may also switch between networking methods, such as from Wi-Fi to GPRS (General Packet Radio Service), as signal strength and availability changes. In addition, some applications may be specifically designed to only download or synchronize data on demand, such as in the mornings and evenings when a travelling salesperson is within range of a corporate or home network, for example.

Therefore, mobile device applications must be designed in such a way that they can download or synchronize data, store it locally, and then upload local changes and resynchronize with the remote data store again at the appropriate times. This may be done automatically when a connection is available or on a predefined schedule, and take into consideration the communication costs of the different types of connections. It may be done only on demand, when the user has completed specific tasks, or it may be a mixture of the two approaches, such as uploading new sales data automatically, but synchronizing the product catalog only on demand.

In addition, some activities may not be possible, depending on the current network type. For example, if only Wi-Fi is available, the device will not be able to use the phone or SMS features. Also consider the cost of data transfer. Reducing the volume of data that you transfer over the network can considerably reduce the user's costs, especially when using the cellular phone network. Data transfer costs also vary considerably in different regions and countries of the world.

Connectivity to the server will not always be available, so you must design your services and clients to support synchronization when connectivity becomes available.

SECURITY CONSIDERATIONS

Windows Phone 7 applications take advantage of several fundamental features of the platform and the phone environment to maximize security. These include the certification of all applications that can be installed (all applications must be installed from Windows Marketplace), the use of only .NET Framework managed code, a sandbox for each application, and an execution manager that monitors resource usage and the behavior of applications.

However, when designing the data storage approach you will use, and the communication mechanism for accessing remote services, you must consider how you will secure the data. Even though the phone requires a user PIN to access it, and data in isolated storage is protected from access by other users of the phone, you should consider encrypting sensitive data that you store on the phone. In addition, unless you are accessing a public service, such as a site that exposes a list of movies currently showing at a local cinema, you must protect the data and the content of messages exchanged with the server from interception and tampering.

Therefore, when designing applications for Windows Phone 7, you should consider the following factors to maximize security:

- Use HTTPS (SSL) when connecting to services where sensitive data is exchanged. If the server certificate is not valid or trusted by a certification authority installed on the phone, the connection will be blocked (you can test a service by navigating to it using the version of Microsoft Internet Explorer® installed on the phone and the emulator). Users can add a certificate authority to the trusted authorities list in the phone, but they cannot add client Secure Sockets Layer (SSL) certificates to support client authentication by the server.
- Encrypt data that you store on the phone, and consider encrypting any particularly sensitive data that you transmit over the network—even when using SSL. Always encrypt sensitive data that you send over non-secure connections, and send only hashed versions of passwords over the network for verification on the server. You can use the AES, HMACSHA1, HMAC-SHA256, Rfc2898DeriveBytes, SHA1, and SHA256 algorithms on the phone.
- If your server application communicates with the Microsoft Push Notification Service (MPNS) to send notifications to the phone, consider using SSL when communicating with the MPNS server. This is possible only when you register for the full version of MPNS.

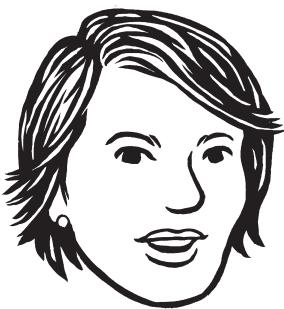
- Take advantage of the tools available for use as part of a Security Development Lifecycle (SDL). These include tools such as the Microsoft Threat Modeling Tool, FxCop, and BinScope.

If you decide that you need to store or encrypt your application's data on the phone, you must be aware of the following points:

- Windows Integrated Security Authentication is not supported on Windows Phone 7. You must implement authentication with remote services and servers using a technique such as Open Authentication (OAuth).
- The Windows Phone 7 API does not include an equivalent to the DPAPI that is available in other Windows operating systems for securing passwords and encryption keys. This means that there is no way to securely store data on a Windows Phone 7 device without requiring the user to enter a password or PIN at some point.
- If you decide to store confidential data on the server instead of on the Windows Phone 7 device, the device must authenticate with the server. This means that you must again prompt the user for a password or PIN at some point.
- You must consider the usability of your application and minimize the number of times that it prompts the user for a password or PIN to access the data. However, if you cache data on the device (including hashed data such as passwords) it may be vulnerable if the device is stolen.
- Encrypting and decrypting data will cause your application to consume more battery power.

For more information about securing Windows Phone 7 applications, see "Security for Windows Phone" at on MSDN ([http://msdn.microsoft.com/en-us/library/ff402533\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402533(VS.92).aspx)). For details of the cryptographic capabilities supported in Silverlight on Windows Phone 7, see "Cryptographic Services in Silverlight" on MSDN ([http://msdn.microsoft.com/en-us/library/cc265159\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/cc265159(VS.95).aspx)).

Note: Microsoft offers a Find My Phone service (<http://www.microsoft.com/windowsphone/en-us/howto/wp7/start/find-a-lost-phone.aspx>) that will help locate missing phones by displaying their location and initiating a call to them, or locking and wiping the phone so that information cannot be accessed by others.



Windows Phone 7 applications can use data exposed by almost any remote service over HTTP in almost any format if you create a suitable proxy in your application.



Remember that you should validate all data received during communication with a host computer and during over-the-air communication to prevent errors occurring in your application.

DATA FORMATS AND ACCESSING REMOTE SERVICES

The Microsoft vision of “three screens and the cloud” encompasses access to information and entertainment, irrespective of the type of device, the screen size, and the user’s location. Applications that use data exposed by cloud services can provide a compelling user experience that is appropriate for the device the consumer is using (such as a phone, desktop computer, or large screen television), and in any fixed or mobile location. This guide focuses on consuming services exposed by cloud-based services, and in particular Windows Azure-hosted services.

Windows Phone 7 applications can consume data from services implemented on any platform and operating system, using any data mechanism that exposes a suitable format and communication protocol. However, it is likely that you will aim to use standard data formats and protocols in your applications to provide decoupling and promote reuse of services. This is especially appropriate for scenarios where you expose data that may be consumed by many different types of devices.

Microsoft is evolving an end-to-end strategy for simplifying data communication between the server and mobile devices based on open standards that allow developers to target a wide range of data stores and devices. This strategy encompasses the typical requirements of receiving, sending, and synchronizing data between the server and the device. The data protocol that Microsoft is supporting as the core protocol for data transmission across the majority of its frameworks and technologies, and specifically between servers and mobile devices, is the Open Data Protocol (OData).

Note: *OData defines a Representational State Transfer (REST)-based mechanism that works over HTTP, and supports Atom, Atom Publishing (Atom Pub), and JavaScript Object Notation (JSON) formats. Atom is an XML-based data syndication format, while JSON provides opportunities for compact serialized data transmission. For more information about OData, see the Open Data Protocol website (<http://www.odata.org/>).*

OData uses an HTTP verb extension named MERGE; it defines this in a way that is compliant with the protocol. Whereas the PUT verb requires all fields of a record to be specified when updating data, the MERGE verb will accept changes and use the existing values for other fields. This is beneficial because it can reduce the volume of data transmitted between the phone and the server.

Data Format and Synchronization Considerations

One way to reduce memory, storage, and communication bandwidth requirements is by choosing an appropriate format for the data and compressing the data. OData and other protocols support JSON serialization, or you can implement your own serialization for communicating and storing data. JSON serialization reduces the size of the data sent over the wire when compared to XML formats, which can reduce transmission costs. Compressing data sent between the server and the device is also a useful way to minimize data transfer costs when using slow and/or expensive network connections such as GPRS or 3G (when compared to cheaper and faster Wi-Fi connection).

However, serialization—particularly for the compressed JSON format and XML format—consumes processor resources. You must balance the communication cost and the processor load factors when you choose a protocol or data format.

Note: *The Microsoft OData Client Library (beta) for Windows Phone 7 does not support JSON serialization. You must use the Atom Pub format with this proxy component.*

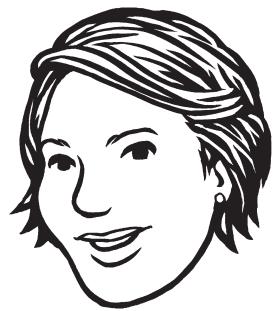
In addition to simply retrieving and uploading data, mobile devices often have a requirement to perform synchronization between data stored on the device and data stored on the server.

Synchronization must take into account occasional connectivity with the network, transmission errors, conflicts between the two data stores due to concurrent updates, and the correct application of keys and relationships between relational data rows on the database server. Synchronization is also often required between types of data that are not relational in nature. This may include log files, media files, or other types of information.

However, before you decide to implement synchronization, consider whether the additional effort it requires can be avoided by applying a simpler pattern. In particular, bi-directional synchronization can be difficult to implement. You may decide to use the Ledger design pattern where, when a conflict occurs, the server or the client version overrides conflict, the content is duplicated, or the user specifies values to resolve the conflict. Alternatively, you may decide to implement synchronization tokens to simplify your design. For information about patterns for moving and synchronizing data, see “Data Patterns” on MSDN (<http://msdn.microsoft.com/en-us/library/ff648502.aspx>).

Alternatively, consider using a framework that implements synchronization if a suitable one is available. An example is the Microsoft Sync Framework, which is described in the following section.

Windows Azure provides a scalable, reliable, and cost-effective solution for exposing data to client applications. For more information about building Windows Azure services, see the Windows Azure Guidance site on CodePlex (<http://wag.codeplex.com/>).



Data synchronization is an important requirement in many mobile applications that use data exposed by a remote service. Consider whether you can simplify your synchronization requirements to avoid the additional complexity it can add to your design.

Microsoft Data and Synchronization Technologies for Windows Phone 7

Microsoft provides a range of technologies for exposing data in standard formats. Typically, on the Windows platform, you will expose data through a SOAP-based or a REST-based service; using a technology such as Windows Communication Foundation (WCF) to implement SOAP-based services or a framework such as WCF Data Services for REST-based services. Frameworks such as WCF Data Services and WCF RIA (Rich Internet Application) Services can expose data in the OData, Atom, Atom Pub, and JSON formats, and support a full REST-based service interface for updating, deleting, and adding new data to the server-based store. Other server frameworks for Windows and other platforms can do the same.

On the client device, proxy components are available that work with the OData protocol to expose a simpler and more useful API to application developers. These components abstract the complexity of interacting with the OData service and allow developers to concentrate on the application code itself. Proxy components are available for Windows Phone 7 applications. In addition, the OData Developer site includes links to OData SDKs for PHP, Java, Objective C (Apple iPhone and Mac), and JavaScript (AJAX and Palm WebOS).

Note: *For information about the OData SDKs and providers that are currently available, see the Open Data Protocol website (<http://www.odata.org/developers/odata-sdk>) and the post, "OData interoperability with .NET, Java, PHP, iPhone and more," on the Interoperability @ Microsoft blog (<http://blogs.msdn.com/b/interoperability/archive/2010/03/16/odata-interoperability-with-net-java-php-iphone-and-more.aspx>).*

Mobile devices can also communicate directly with SQL Azure™ databases, although this requires use of an ADO.NET provider that understands the tabular data stream (TDS) protocol. It also requires more effort on the client side to write the data access code because the ADO.NET SQL client libraries are not available on the phone, and offers no opportunity to include server-based business logic outside of the database operations. However, it may be a useful technique in some application scenarios.

Note: *The implementation of Silverlight on Windows Phone 7 does not directly support all the networking features of Silverlight 4.0 and the Silverlight 4.0 SDK. For more information, see "Networking for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff637320\(VS.96\).aspx](http://msdn.microsoft.com/en-us/library/ff637320(VS.96).aspx)).*

Built on top of OData is the OData Sync protocol. This protocol is specifically designed to make synchronizing data across networks much simpler by using pluggable providers for each type of data store or device. A central orchestration component managed by the application configures endpoints and controls the transfer of data changes between the endpoints.

To support the OData and OData Sync protocols, Microsoft and many third-party companies offer frameworks, providers, connectors, proxies, and other components that run both on Windows and other platforms. Figure 4 shows a high-level overview of the data communication and synchronization mechanisms build around the OData and OData Sync protocols.

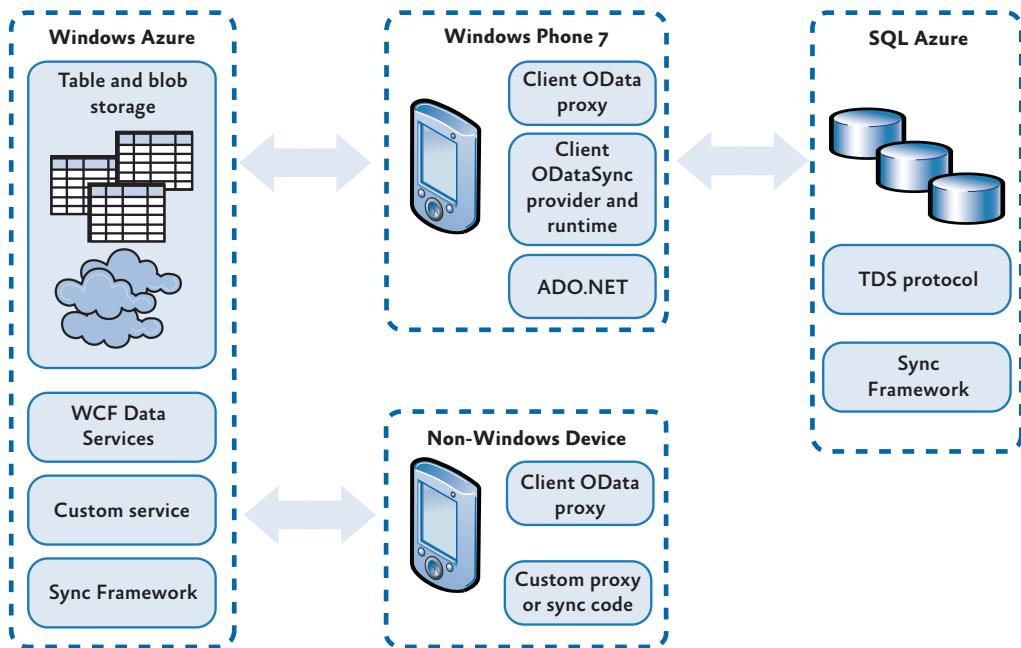
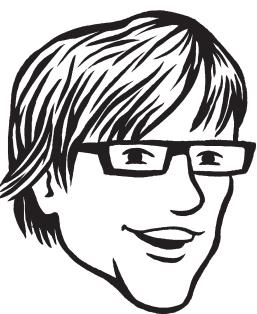


FIGURE 4
Overview of data communication and synchronization for Windows Phone 7 and Windows Azure

Note: For more information about the Microsoft Sync Framework, see Appendix E, "Microsoft Sync Framework and Windows Phone 7."



Failing to use the resources of the phone intelligently can considerably reduce the performance of your application



Look at the recommendations earlier in this chapter for minimizing performance impact when communicating and storing data.

RESOURCE MANAGEMENT AND PERFORMANCE

Like all mobile devices, Windows Phone 7 devices have limited processing, memory, and storage facilities compared to a desktop or laptop computer. It is important to take this into account when building applications that will run on Windows Phone 7.

The main areas of concern when designing mobile device applications are to manage device resource usage effectively and maximize performance on the device. Typically, maximizing performance involves optimizing memory utilization, maximizing use of the graphics processing unit (GPU), and more general issues, such as good programming practice. The following sections provide guidance on achieving these aims in your applications.

Minimize Device Resource Usage

Many hardware features of the device draw additional current and reduce battery life. You should consider limiting use of these features when not necessary. For example, you should access the location service (which may initiate the built-in GPS) only when necessary, and turn it off when not required. Consider implementing options or power profiles in your applications that allow the user to minimize power use or turn off features of the device when not required.

Also consider how you can minimize the use of the network, Bluetooth, storage, video, sound, and background tasks. All of these require processor and memory resources, and they impose an additional load on the battery. For information about using the device capabilities, such as location services, and minimizing their impact on performance, see Appendix C, "Leveraging Device Capabilities."

Apply Good Practice Programming Techniques

You must be aware of the work your application is doing and the use of resources within your code. Where possible, you should attempt to do the following:

- Apply the optimizations you would use in any other .NET Framework code, such as caching the appropriate types and volumes of data, avoiding boxing (which occurs when you treat a value type as an object), and using lazy loading so that memory is only allocated to objects if they are actually used by the application.
- Use asynchronous programming techniques to perform any complex processing tasks on background threads instead of on the UI thread. Use **Dispatcher.BeginInvoke** to initiate operations on the UI thread, and pass data between tasks on different threads using local variables, a memory buffer, or isolated storage. Also consider using the Reactive Extensions

(Rx) to implement asynchronous execution for tasks. Rx can simplify asynchronous programming tasks and make code easier to read and assimilate.

- Minimize the visual tree of objects and code execution. Less activity in the application will give better performance and reduce battery drain.
- Minimize the size of objects that you redraw or animate within the UI and the number of operations (such as transitions or effects) you perform on the objects. If you use code to drive animations with per-frame callbacks, these execute on the UI thread and may appear less smooth than those on the compositor thread when the UI thread is busy.
- Ensure that media you display or play (such as video or audio) is encoded at the appropriate size for display, and that it has the minimum acceptable frame or audio bit rate.
- Minimize the processing that occurs in the **Loaded** event of the initial page, which will delay initial display of the application. Alternatively, consider using the splash screen (SplashScreen Image.jpg) that is included in each Windows Phone 7 project template. You can modify it to show a suitable image while the application loads.
- Test regularly on a physical device, not just by using the emulator, because the performance of the two will vary dramatically based on the hardware available to each.

Optimize Memory Usage

Your application will perform better and be a “good citizen” when running on Windows Phone 7 if you carefully control use of memory and isolated storage. You must also be aware that garbage collection in Windows Phone 7 devices works differently than it does in desktop and server versions of the .NET Framework. In Windows Phone 7, the garbage collector runs only when memory usage exceeds a specified level, and it can interrupt execution—which can, therefore, affect performance.

Where possible, you should attempt to do the following:

- Optimize data downloads and store only data that is actually required on the device.
- When accessing large sets of remote data, consider using data paging so that only the data that will be viewed is sent to the device.
- If you will reuse objects you create in your code, consider holding them in memory instead of allowing them to be garbage collected and then recreating them. This reduces the frequency

of garbage collection, which can interrupt program execution to a greater extent than in a desktop application. However, do not hold onto objects that will not be regularly reused, and consider minimizing the total number of object instances that you create in order to reduce memory requirements. Using a few large objects instead of a large number of small objects can also reduce garbage collection overhead, but you must also consider whether the performance gain outweighs the benefits of smaller, more-focused and decoupled objects that follow the single responsibility principle.

- Consider creating or loading the objects and data you need when your application starts. If you minimize the requirement to allocate memory for new objects as the application runs, the garbage collector will run less often. If you use mainly value types and arrays of value types, you will minimize the time that the garbage collector takes to execute. However, you must balance the effects, depending on whether you require fast startup or faster execution after startup. Alternatively, consider downloading the data you need asynchronously on a background thread and allowing the application to start with some limit to its functionality and then gradually enable features as the data that they need becomes available. A dependency injection container can help you to manage object creation.
- Consider calling the **GC.Collect** method to force garbage collection when the application is at a point where there is no interaction with the user and no major tasks are running in your application. This may be immediately after initializing when the main screen is displayed or at a suitable point where the user is reading content you have displayed. This reduces the chance that it will run at a more intrusive time and affect the interactivity of your application.
- Consider simplifying the structure of your application components when using design patterns that provide decoupling (such as MVVM or Dependency Injection) if you can do so without fundamentally affecting the architecture or objectives of the patterns. For example, you may be able to share a view model between two or more views.
- Minimize the size of the application assemblies. Use a **Build Action** of **Content** (instead of **Resource**) for media, images, and other content to include them as files in the XAP file instead of as resources embedded within the assembly. However, if you are building a reusable component, you will probably need to embed the resources within the assembly for easier distribution.

Note: You can query the current memory availability and usage by your application using the **DeviceExtendedProperties** class in Windows Phone 7. For more information, see Appendix C of this book, “Leveraging Device Capabilities,” and “Device Information for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff941122\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941122(VS.92).aspx)).

Maximize GPU Usage

Windows Phone 7 provides a background compositor thread that generates output for display and a relatively powerful GPU that maximizes performance for graphics and maintains an acceptable frame rate. However, you must take care to not negate this advantage through the use of code that cannot run on the GPU.

Where possible, you should attempt to do the following:

- Minimize the code you execute on the UI thread, which handles input, the drawing of new visual elements, and calls to user code. Use the compositor thread to execute transitions and animations, and apply caching for bitmaps. The compositor thread handles opacity, scale transforms, translate transforms, rotate transforms, and plane projection. It does not handle opacity masks, non-rectangular clips, and texture for animated objects greater than 2000 × 2000 pixels.
- Consider caching complex visual elements if they will be regularly reused. When you hide a visual element (visibility = collapsed) it is discarded and requires no processor utilization. This means that it must be completely recreated when you show that element again. Instead, consider setting the opacity to zero. This causes the device to keep the object, while still minimizing processor usage. However, you must balance this with memory usage.
- Consider using images instead of creating complex visual elements in Extensible Application Markup Language (XAML) or code. This can considerably reduce processor load. If you need transparency, you must use a PNG image. If you do not need transparency, use a JPEG image because the render speed is higher for this image format.
- Images larger than 2000 × 2000 pixels will be sampled at a lower resolution when displayed, with subsequent loss of quality. Store larger images in a **WriteableBitmap** and display only the appropriate section.

There are several modes you can enable when debugging. These are especially useful when maximizing graphics performance; they are automatically included in the code generated by the phone applica-

tion template in Visual Studio. Graphics debug mode is set by the following properties of the **Application.Current.Host.Settings** class in the startup code in your App.xaml.cs file:

- **EnableFrameRateCounter.** This displays a frame rate counter, memory usage information, and other useful data about the performance of the device. It shows the frame rates for the UI and the compositor threads, the amount of texture memory in use, the number of surfaces, and the fill rate. Ensure that the **SystemTray.IsVisible** static property is set to **false** in your application so that the system tray does not obscure the frame rate counter.
- **EnableCacheVisualization.** This shows the graphics items that are being drawn by applying tint and transparency to each one to show where overlapping textures are being drawn on each frame update. Each tinted area represents a texture that is handed off to the GPU for composition.
- **EnableRedrawRegions.** This shows the graphical items that are being redrawn in each frame update.

For more detailed information about the background compositor thread and maximizing performance on Windows Phone 7, see *Creating High Performing Silverlight Applications for Windows Phone* from the Microsoft Download Center (<http://download.microsoft.com/download/B/2/7/B2748D7A-F368-4C33-BoF2-844CFE085193/SilverlightForWindowsPhonePerformance.zip>). For more background information about performance optimization, see the video presentation, *Silverlight Performance on Windows Phone*, on the Mix 10 website (<http://live.visitmix.com/MIX10/Sessions/CL60>) and “WP7/Silverlight Graphics Performance” on andybeaulieu.com (<http://www.andybeaulieu.com/Home/tabid/67/EntryID/196/Default.aspx>).

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

AVAILABILITY OF COMPONENTS AND FRAMEWORKS

In addition to the challenges developers face in managing connectivity and limited device resources, the fact that mobile device technologies are reasonably new and continually evolving makes technology choices more difficult. Although most server technologies are stable, the Windows Phone 7 is brand new and—at the time of this writing—many of the interoperability components and technologies are still under development or available only as preview versions.

As you will see when you read in this guide about the decisions Tailspin had to make during development of their Surveys application, the selection of data communication and synchronization technologies is influenced by the availability and stability of these technologies.

For example, if you choose to use WCF Data Services to expose your data in the standard Atom Pub REST format for Windows Phone 7 devices to consume, you can use the Microsoft OData Client Library for Windows Phone 7 instead of creating a custom proxy. This library is implemented in the assembly **System.Data.Services.Client.dll**, but is not yet (at the time of this writing) available as a final release version for Windows Phone 7.

In addition, other frameworks or components originally designed for the desktop or server, or specifically aimed at Silverlight applications running in a web browser, may not be compatible with the Windows Phone 7 operating system. You should test all components and frameworks on both an emulator and on a physical device before committing to use them in your application designs.

Questions

1. Which of the following should you consider when designing the UI for a Windows Phone 7 application?
 - a. Making it work in both portrait and landscape mode.
 - b. Making it colorful and highly configurable.
 - c. Maximizing the number of controls on each page.
 - d. Reusing an existing web or desktop application UI.
2. Which of the following languages and frameworks can be used to create Windows Phone 7 applications?
 - a. Silverlight.
 - b. C++.
 - c. XNA.
 - d. Microsoft Visual Basic®.
3. Which of the following are **not** events that may occur as part of the tombstoning and reactivation process for a Windows Phone 7 application?
 - a. Activated.
 - b. Loading.
 - c. Resuming.
 - d. Deactivated.

4. Which of the following ways would you consider storing the data used by an application?
 - a. Using isolated storage on the phone.
 - b. Using the built-in SQL Server® Compact Edition database on the phone.
 - c. Using the file system on the phone.
 - d. Storing it on the server and synchronizing it with the phone.
5. Which of the following techniques should you consider for securing a Windows Phone 7 application?
 - a. Never store any data on the phone.
 - b. Encrypt all sensitive data stored on the phone.
 - c. Use HTTPS for all requests containing sensitive data when communicating with the server.
 - d. Perform a security analysis of your design and the code.
6. Which of the following are useful techniques to maximize performance of a Windows Phone 7 application?
 - a. Turn off features such as the Location Service if you do not require them, or turn them off as soon as possible after using them.
 - b. Use single-letter names for variables in your code.
 - c. Optimize your code using good coding practice and simplifications where appropriate.
 - d. Make full use of the graphics processing unit (GPU) by avoiding code that cannot run on it.

This chapter introduces a fictitious company named Tailspin. Tailspin's flagship product is an online service named Surveys that enables other companies or individuals to conduct their own online surveys. Tailspin wants to extend this service to mobile users, enabling subscribers to the Surveys application to publish surveys to people with Windows® Phone 7 devices. These people can use the mobile application to capture survey data from the field. As with any company embarking on a new project, there are many issues to consider and challenges to meet, particularly because Tailspin has not used the Windows Phone 7 platform before. The chapters that follow show, step by step, how Tailspin designed and built a survey client application to run on Windows Phone 7 devices.

The Tailspin Company

Tailspin is a startup ISV company of approximately 20 employees that specializes in developing solutions using Microsoft® technologies. The developers at Tailspin are knowledgeable about various Microsoft products and technologies, including .NET Framework, Windows Mobile, Windows Azure™ technology platform, Silverlight® browser plug-in, and Microsoft® Visual Studio® development system. The Surveys mobile client was the first application that the developers at Tailspin created for the Windows Phone 7 platform.

The Surveys mobile client application is the first of several innovative Windows Phone 7 applications that Tailspin wants to take to market. Tailspin hopes the innovative approach to collecting survey data that the mobile client application offers will help it to grow its market share and increase its revenues.

TAILSPIN'S STRATEGY

Tailspin is an innovative and agile organization; it is well-placed to exploit new technologies, the business opportunities offered by the cloud, and the increasing sophistication of mobile phones. As a startup company, Tailspin is willing to take risks and use new

technologies when it implements applications. Tailspin's plan is to gain a competitive advantage as an early adopter of new technologies, especially in mobile devices and the cloud. It hopes to rapidly gain some experience, and then quickly expand on what it learns. This strategy can be described as "try, fail fast, learn, and then try again." Tailspin has decided to start with the Surveys mobile client application as its first Windows Phone 7 offering.

TAILSPIN'S GOALS AND CONCERNs

The Surveys application has been a great success for Tailspin, but Tailspin wanted to further improve its market position by extending the application's functionality. Based on feedback from its subscribers, Tailspin identified four key areas where they could enhance the Surveys application:

1. The application should support a wider range of question types and enable respondents to include additional data, such as pictures, audio, and location data, as a part of their survey responses.
2. People should be able to provide survey responses when they are away from their computers. A convenient time to respond to a survey might not be a convenient time to be using a computer, for example during a commute or while waiting in a checkout line.
3. Subscribers wanted to be able to capture a geographical location for the respondents answering a survey.
4. Subscribers wanted to be able to proactively find survey respondents. Instead of waiting for respondents to come to the survey website by following a link on a web page or in an email, subscribers wanted other ways of finding survey respondents. For example, subscribers wanted to be able to use surveyors who can go out and interview people. In the future, Tailspin would like people to be able to launch surveys by photographing a barcode or QRCode on an advertisement or on product packaging.



The developers at Tailspin already have Silverlight skills that helped them get started with the new platform.

Windows Phone 7 was a new platform for the developers at Tailspin and the unfamiliarity introduced some additional risk to the project. The developers first had to understand the capabilities of the Windows Phone 7 platform to determine how best to architect and design both the mobile client application and the new elements of the application in the cloud. Three key areas of concern for Tailspin in using the new platform were reliability, security, and connectivity.

Windows Phone 7 devices may be only intermittently connected to the Internet, so the mobile application had to be capable of reliably storing the collected data until it could be sent to the cloud

application. Tailspin also wanted to make sure that any data held on the Windows Phone 7 device was stored securely.

For some surveys, subscribers wanted to be able to determine the identity of the person submitting the survey data to the cloud application.

Tailspin also wanted to implement a service endpoint in Windows Azure that best supports the requirements of the Windows Phone 7 devices. The developers at Tailspin had to make decisions about the connectivity between the mobile application and the back end, such as whether to use Representational State Transfer (REST)-style or SOAP-style web services, how “chatty” the interface should be, and how to handle retries when sending a message failed.

Finally, Tailspin would like to be able to leverage the existing skills of its developers and minimize any retraining necessary to build the Surveys application.

The Surveys Application Architecture

Figure 1 shows a high-level view of the architecture of the extended Surveys application.

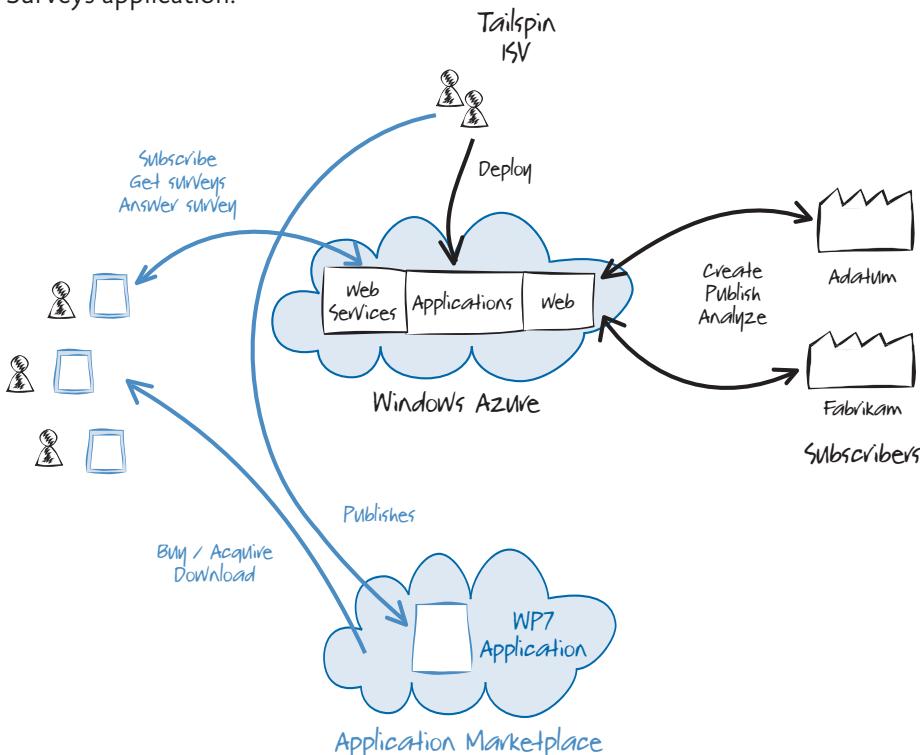
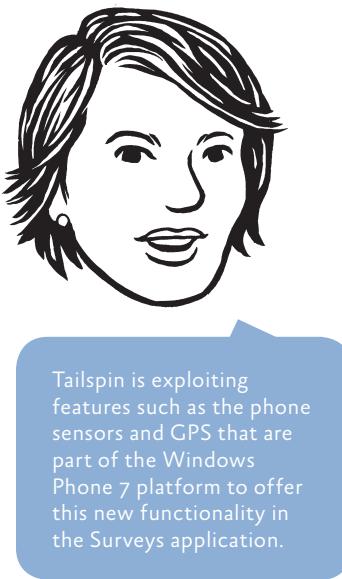


FIGURE 1
Architectural view of the Tailspin Surveys application



There are two, top-level components in the Surveys application. The first is the back end that Tailspin hosts in Windows Azure and that enables subscribers to create, publish, and analyze surveys. This back end is described in the book, *Developing Applications for the Cloud on the Microsoft Windows Azure Platform*, available on the MSDN® website (<http://msdn.microsoft.com/en-us/library/ff966499.aspx>). The second component, which is the focus of this guide, is a mobile client front end that runs on Windows Phone 7 devices and that enables surveyors to collect survey response data and send it to the back end. This guide also describes the changes to the back-end cloud application that were necessary to support the mobile client.

Tailspin is launching the mobile client application to support new features in the Surveys service. These new features include the following:

- The capability for surveyors to be able to filter available surveys on different criteria
- The ability to collect rich data from survey respondents, such as the respondent's location, voice recordings, pictures, barcodes, and physical measurements captured from the device's sensors, as part of the survey
- The capability of the application to notify surveyors that new surveys are available

THE ACTORS

There are three actors in the scenario supported by the architecture, the ISV, the subscribers, and the surveyors.

Tailspin - the ISV

Tailspin has developed a multi-tenant, Software as a Service (SaaS) application named Surveys that it runs in the cloud. A range of subscribers—from individuals, through small companies, to large enterprises—uses the Surveys service to run custom surveys. Tailspin has also developed the mobile client application for Windows Phone 7 devices described in this guide that it makes available to surveyors through Windows Marketplace for Mobile.

Fabrikam and Adatum - the Subscribers

In the scenario, Fabrikam and Adatum are also fictitious companies who play the role of subscribers to the Surveys service. They design and launch surveys using the Surveys service, wait for responses, and then analyze the results that the Surveys application collects.

The Surveyors - Windows Phone 7 Users

The surveyors, who typically work from home, subscribe to surveys based on a predefined criteria and are notified when new surveys are published. Using a Windows Phone 7 device, they can either answer the survey questions themselves, or they can interview other people and use the device to capture the survey response data. For example, a surveyor could use the device to record traffic patterns at different times of the day or to go door-to-door collecting survey responses.

THE BUSINESS MODEL

Tailspin's business model is to charge subscribers a monthly fee for access to the Surveys application, and Tailspin must then pay the actual costs of running the application. The Surveys mobile client is free to surveyors, and surveyors who are collecting multiple responses to surveys can also be compensated. A Surveys subscriber, like Adatum, could either pay a surveyor for the number of submitted surveys or offer discount coupons. For this to work, it must be possible to identify the surveyor who submitted the survey responses. Tailspin is currently investigating ways they can send the coupons directly to the Windows Phone 7 device for scanning.

Tailspin is also planning to use the Microsoft Advertising SDK for Windows Phone 7 to embed advertisements in the mobile client as an additional way of generating revenue. You can download the SDK from the Microsoft Download Center (<http://go.microsoft.com/fwlink/?LinkId=198440>).

Note: *The sample application that you can download to go with this guide doesn't implement all these capabilities; however, it is likely that a real-world version of this application would consider doing something like this.*

The initial version of the application will allow surveyors to filter surveys based on the tenant, but Tailspin plans to extend this in the future to include filters on factors such as survey length, target audience, and location.



By using surveyors, Tailspin plans to target surveys more effectively and improve the response rate.

Tailspin will make the mobile client application available for free.

THE APPLICATION COMPONENTS

Figure 2 illustrates the key functional components of the mobile application and the relationships between them.

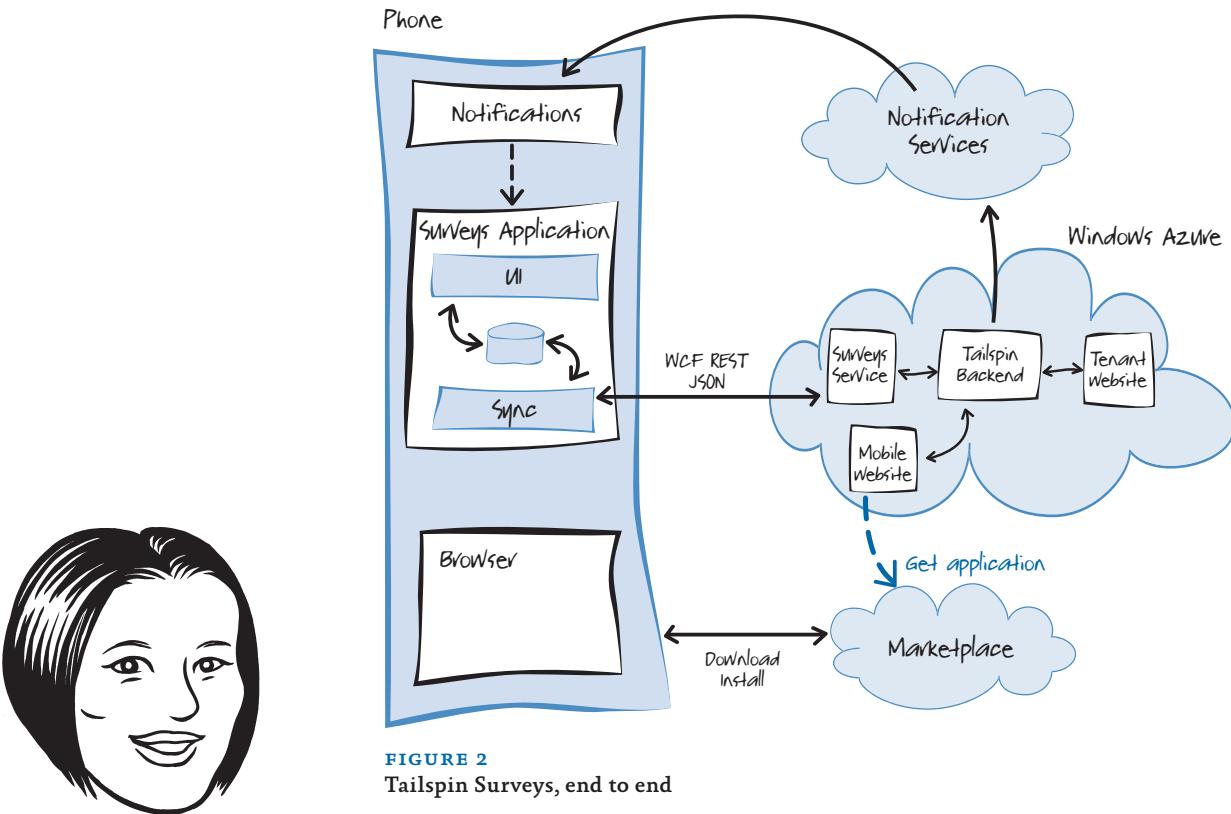


FIGURE 2
Tailspin Surveys, end to end

The Windows Phone 7 client application is an alternative to using the web as a mechanism for collecting survey responses. Tailspin anticipates using additional platforms in the future.

Developing Applications for the Cloud on the Microsoft Windows Azure Platform describes the Tailspin back end and Subscriber website architecture, design, and implementation in detail. These components, which run on Windows Azure, enable subscribers to design new surveys and to analyze the responses that the application collects. The book also describes a public website that people can use to complete surveys by using a web browser. The scenario described in this guidance focuses on an application running on the Windows Phone 7 device that provides an additional way for Tailspin to capture survey results.

The Surveys application on the Windows Phone 7 device comprises three components. A user interface (UI) enables the user to complete surveys and perform other tasks. A storage repository holds survey definitions and survey responses. A synchronization compo-

nent is responsible for downloading survey definitions from the Tailspin back end and for uploading completed survey data.

To enable the Windows Phone 7 application to communicate with the back end, the cloud components now include an API that exposes the functionality that the mobile application requires. Tailspin decided to use Windows Communication Foundation (WCF) REST to transport the data over the network. The Windows Phone 7 application must also authenticate with the back end so that the back end can determine which surveys it should make available to the mobile client and can track which responses come from which user. In the scenario described in this guidance, the mobile client authenticates with the back end using basic authentication, but it is designed in such a way that it could be extended to accept more sophisticated mechanisms, such as a claims-based approach.

The application uses push notifications to inform the mobile client that there are new surveys available to download. These push notifications will reach the Windows Phone 7 device even when the mobile Surveys application is not running.

The Surveys application also includes a website for mobile clients that it uses to enable the Windows Phone 7 device to download and install the client application from Windows Marketplace. Tailspin plans to use this website to host additional resources, such as help and guidance notes for users of the mobile application.

Later chapters in this guide describe these three components in more detail.

All links in this book are accessible from the book's online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, "Where to Go for More Information."



This chapter describes how the developers at Tailspin built the user interface (UI) components of the application. It begins by discussing some of the goals and requirements that Tailspin identified for the application before discussing, at a high level, the structure and key components of the application.

The chapter then discusses navigation and UI controls in more detail, and it describes how and why Tailspin implemented the Model-View-ViewModel (MVVM) pattern. The chapter also gives an overview of the MVVM pattern itself.

The chapter includes discussions of the design Tailspin adopted for the application as well as detailed descriptions of the implementation.

Overview of the Mobile Client Application

This section provides an overview of the mobile client application to help you understand its overall structure before you examine the components that make up the application in more detail. Also, to help you understand some of the design decisions made by the developers at Tailspin, it describes some of the goals and requirements that Tailspin identified for the application.

GOALS AND REQUIREMENTS

The Windows® Phone 7 device is a brand new platform that offers a wealth of features to developers and designers. The team at Tailspin wanted to ensure that their mobile client application makes the best possible use of the platform and also plays by the rules. The application follows the recommended usability guidelines to ensure the optimal user experience and the “good citizen” best practices guidelines to ensure that the application makes efficient use of resources on the device in the context of the phone’s functionality and other installed applications. They identified three sets of goals for the design and development of the application: usability goals, non-functional goals, and development process goals.

The Tailspin mobile client application follows usability and good citizen best practices.

Note: For more information about Windows Phone 7 UI design guidelines, see *UI Design and Interaction Guide for Windows Phone 7* from the Microsoft® Download Center (<http://go.microsoft.com/fwlink/?LinkId=183218>).

Usability Goals

The usability goals are designed to ensure that the user's experience of the application meets her expectations for applications on a Windows Phone 7 device. The following table lists some examples.

Goal description	Example
Take advantage of the appearance and behavior (look and feel) of the Windows Phone 7 platform.	The application uses standard input gestures to enter data, uses the standard system colors, and includes icons and graphics designed to match the phone's theme. The application can update its appearance to blend with the phone's standard Light and Dark themes. For more information, see the ThemedResourceLocator class in the Resources folder.
Use the standard controls that Windows Phone 7 users are familiar with.	The application uses standard controls, including the ApplicationBar , Pivot , and Panorama controls, to make the user feel at home and to minimize the learning curve.
Follow other Windows Phone 7 UI guidelines, such as those on the use of the hardware Back button and the behavior of the application when the user answers a call or switches to another application.	The Back button navigates backward in a way that matches the user's expectations. The application restores the UI to its previous state after the user finishes answering an incoming call.
Integrate with the sensors on the phone.	The application uses the location services on the phone to establish its geographical location, and it uses the camera and microphone to collect data for some survey questions.
Handle changes in screen orientation.	The application automatically updates the display orientation when the user changes the phone's orientation.
Handle standard screen resolutions.	The application displays correctly in the standard screen resolutions for Windows Phone 7 devices.
The application should always have a responsive UI.	The application performs long-running tasks, such as synchronizing with the Surveys service, asynchronously. The application remains responsive when it has a large number of surveys saved locally and when it is displaying a survey with a large number of questions.

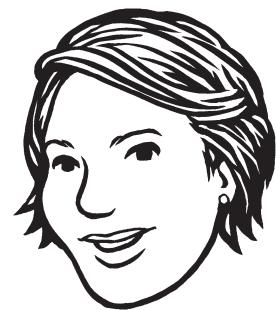


Users will prefer an application that fits well with the phone's UI design and theme. You will also have to comply with certain UI design guidelines if you want to distribute your application through Windows Marketplace.

Non-Functional Goals

The non-functional goals describe expected behaviors for the application, including some good citizen behaviors that relate to the limited resources on the device. The following table lists some examples.

Goal description	Example
The application should continue to operate even when it is not connected to the back end in the cloud.	The application stores survey definitions and user responses in local storage, and it synchronizes with the cloud back end store when connectivity is restored.
The application should not rely on specific network capabilities or assume a minimum available bandwidth.	The UI always interacts with local storage. The application uses an asynchronous call to synchronize with the Tailspin Surveys remote service and uses a store and forward pattern.
The application should try to minimize the costs associated with using the network.	The application tries to minimize the amount of data transferred over the network by using JSON serialization instead of XML. The application does not compress the data because of the additional CPU overhead and battery consumption that this requires. The application should also check the current network interface type and modify its behavior accordingly, but this is not implemented in the sample application.
The application should try to minimize battery usage when the battery level is low.	If the phone is running low on resources, it will deactivate your application and you should take care to save any critical data at this point.
The application should respond to out of memory issues gracefully.	The application should check for out-of-memory conditions. You can also use the DeviceExtendedProperties class to check your application's memory use.
The application should proactively notify users of significant events occurring in the back end.	The back end uses the Microsoft Push Notification Service to notify users of new surveys available for their phones.
The application should use memory efficiently and, for performance, minimize memory allocations.	The sample application uses a dependency injection container to manage which objects are cached to improve performance and which objects are recreated whenever they are used.
As a "good citizen," the application should minimize its use of isolated storage, a shared resource on the phone.	The application removes completed surveys from isolated storage after the data successfully synchronizes with the Tailspin Surveys service. It also uses the JSON serializer when it saves data to isolated storage.



You should always be aware of how your application consumes the limited resources on the phone, such as bandwidth, memory, and battery power. These factors are far more significant on the phone than on the desktop.



The Windows Phone 7 platform is new, so expect it to change and design your application so that you can easily modify it to use new features.

Development Process Goals

Tailspin also identified a number of goals that relate to their own development processes. The following table lists some examples.

Goal description	Example
Tailspin wants to have highly testable code.	A significant advantage of the MVVM pattern is that it makes the code more testable.
Tailspin wants to be able to support other mobile platforms in the future.	Using standards-based approaches to interact with the back end makes it easier to develop other clients for other platforms.
Tailspin wants to have an efficient development process.	Developers and designers can work in parallel. Designers can prototype and build the UI using Microsoft Expression Blend® design software while the developers focus on the application's logic.
Tailspin wants to be able to adapt the application to work with any new capabilities of future versions of the Windows Phone 7 platform.	The current version of the application uses an abstract persistence model to wrap local isolated storage on the device. Tailspin could easily change this to use Microsoft SQL Server® Compact Edition in the future.

THE COMPONENTS OF THE MOBILE CLIENT APPLICATION

Figure 1 shows the main components that comprise the Tailspin Surveys client application.

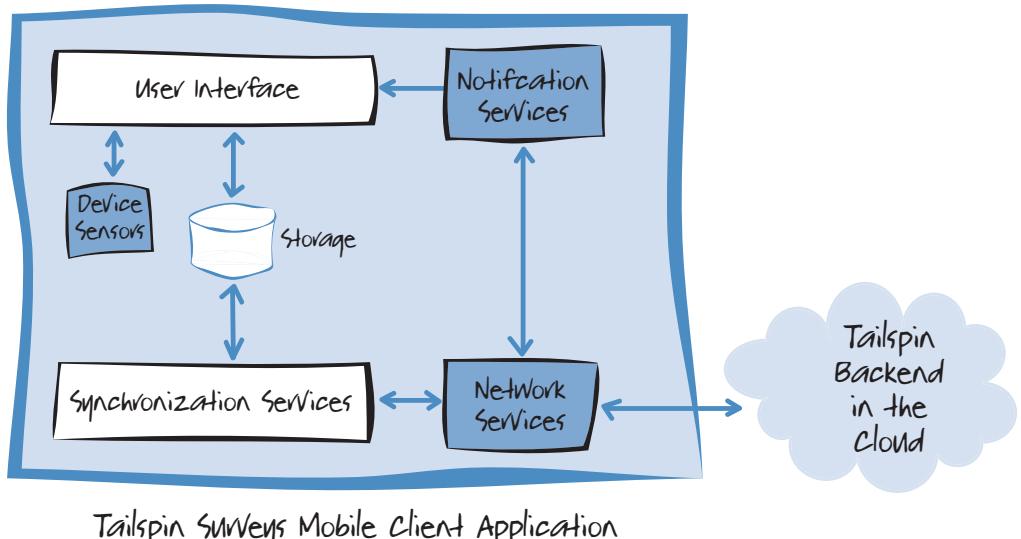


FIGURE 1
The Tailspin Surveys client application

The developers at Tailspin built three key components of the application: the UI, the storage sub-system, and the synchronization service. The application also uses some components of the Windows Phone 7 platform, in particular the on-board device sensors, the notification services, and the network services that the application uses to communicate with the back-end web services.

This chapter focuses on the UI components and also describes how the application components are linked together through Tailspin's implementation of the MVVM pattern. Chapter 5, "Using Services on the Phone," will examine the storage and synchronization components, and Chapter 6, "Connecting with Services," will look at the notification process and the integration with the back end in more detail.

There is some discussion of device sensors in this chapter, but you will find more in-depth information in Appendix C, "Leveraging Device Capabilities."

The application uses a number of features offered by the Windows Phone 7 platform.

THE STRUCTURE OF THE TAILSPIN SURVEYS CLIENT APPLICATION

Figure 2 shows the structure of the Tailspin Surveys mobile client application in more detail. For clarity, the diagram does not show all the links between all the components. In particular, multiple links exist between the model components and the view model and the application services, but showing all of these would unnecessarily clutter the diagram.

Views

```
WMAppManifest.xaml
<Task>
<DefaultTaskName = "_default" NavigationPage= "Views/SurveyList/SurveyListView.xaml">
</Tasks>
```

AppSettingsView.xaml

FilterSettingsView.xaml

SurveyListView.xaml

TakeSurveyView.xaml

Question Views

View Models

AppSettingsViewModel

FilterSettingsViewModel

SurveyListViewModel

TakeSurveyViewModel

Models

TenantItem

Question

QuestionAnswer

QuestionType

SurveyAnswer

SurveyTemplate

OpenQuestionViewModel

MultipleChoiceQuestionViewModel

FiveStarsQuestionViewModel

PictureQuestionViewModel

VoiceQuestionViewModel

Application Services

App.xaml

ViewModelLocator

Notification Service

Surveys Synchronization Service

Other Infrastructure Services

Settings Store

Tombstoning

Surveys Store

Network

FIGURE 2
Tailspin Surveys mobile client application structure

To understand how Tailspin built the UI components (such as the **SurveyListView** page and the **AppSettingsView** page), how the navigation between the pages work, and how the application determines which page to display to the user when the user launches the application, you should read the section, “The Design of the User Interface,” later in this chapter.

To understand how and why Tailspin uses the MVVM pattern, you should read the section, “Using the Model-View-ViewModel Pattern,” later in this chapter. This section explains the roles of the view, view model, and model components and how they are linked together, including the role of the **ViewModelLocator** class. This section also describes some data binding scenarios in the application, including the way the application uses the **Pivot** control on the **SurveyListView** page and the **Panorama** control on the **TakeSurveyView** page.

To understand how the application manages its state when it’s tombstoned, you should read the section, “Handling Activation and Deactivation,” in Chapter 5, “Using Services on the Phone.”

Note: *An application is tombstoned by the Windows Phone 7 device when, for example, the user navigates to another application or answers a call while using the application. The application should preserve any state it needs to return to the correct state when it is reactivated.*

To understand how the application manages persistent data on the phone, such as application settings and survey responses, you should read the section, “Using Isolated Storage on the Phone,” in Chapter 5, “Using Services on the Phone.”

To understand how the Tailspin Surveys cloud application can notify the mobile client of new surveys by using the push notification service, you should read Chapter 6, “Connecting with Services.”

To understand how the application transfers survey data between the mobile client application and the cloud application, you should read Chapter 6, “Connecting with Services.”



Dependency injection enables decoupling of concrete types from the code that depends on these types. It uses a container that holds a list of registrations and mappings between interfaces and abstract types and the concrete types that implement or extend these types.

Dependency Injection

The developers at Tailspin use a dependency injection container to manage the instantiation of many of the classes, including the view model classes, in the application.

Tailspin uses the Funq dependency injection container (available at <http://funq.codeplex.com/>) instead of the Unity Application Block (Unity) because Unity requires the ability to emit Microsoft intermediate language (MSIL) code, which is not currently possible on the Windows Phone 7 platform. The Funq dependency injection container is also lightweight and fast.

The **ContainerLocator** class shows how the application creates the registrations and mappings in the Funq dependency injection container. In the Tailspin mobile client application, the **ViewModelLocator** instantiates the **ContainerLocator** object and is the only class in the application that holds a reference to a **ContainerLocator** object.



You should consider carefully which objects you should cache and which you should instantiate on demand. Caching objects improves the application's performance at the expense of memory utilization.

By default, the Funq dependency injection container registers instances as shared components. This means that the container will cache the instance on behalf of the application.

The Contents of the TailSpin.PhoneClient Project

The TailSpin.PhoneClient project organizes the source code and other resources into folders. The following table outlines what is contained in each folder and provides references to where this guide describes the content in more detail.

Project folder	Description
Root	In the root folder of the project, you'll find the App.xaml file that every Microsoft Silverlight® project must include. This defines some of the startup behavior of the application. The root folder also contains some image files that all Windows Phone 7 applications must include.
Properties	In this folder, you'll find two manifest files and the AssemblyInfo.cs file. The WMAppManifest.xml file describes some of the capabilities of the application and specifies the initial screen to display when the user launches the application.
Resources	This folder holds various image files that the application uses and some utility classes that perform conversions to types that are used in UI.
Views	This folder contains the XAML files that define the screens in the application. The section "Using the Model-View-ViewModel Pattern" in this chapter describes the role of views in this pattern and highlights the fact that there should be little or no code in the code-behind files.
ViewModels	This folder contains the C# files that implement the view models. The section "Using the Model-View-ViewModel Pattern" in this chapter describes the role of view models in this pattern. You will find more view models than views because individual controls may also have their own view models.
Models	This folder contains C# files that implement the models. The section "Using the Model-View-View Model Pattern" in this chapter describes the role of models in this pattern.

Themes	The XAML files in this folder contain style definitions.
Services	This folder contains wrappers for various services that the application uses. Those that relate to navigation and the core application structure are discussed in this chapter. The rest are described in Chapter 5, “Using Services on the Phone.”
Services/ RegistrationService	These folders contain web service client implementations that interact with the Tailspin Surveys service in the cloud. Chapter 6, “Connecting with Services,” describes the web service clients in this folder.
Services/Stores	This folder contains wrappers for the isolated storage API that the application uses. Chapter 5, “Using Services on the Phone,” describes these stores.
Infrastructure	This folder contains “plumbing” code that facilitates implementing the MVVM pattern on the Windows Phone 7 platform and other utility code required by the application. The section “Commands” in this chapter describes how some of classes in this folder are used by the commands implemented in the view model.

The Design of the User Interface

The Tailspin Surveys mobile client application follows the UI design guidance published in the *UI Design and Interaction Guide for Windows Phone 7*, which you can download from the Microsoft Download Center (<http://go.microsoft.com/fwlink/?LinkId=183218>).

This section describes how users navigate between the different pages of the mobile application and outlines the controls that Tailspin uses in the UI of the mobile client application.

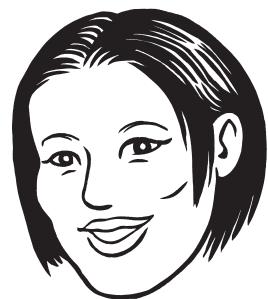
UI Design and Interaction Guide for Windows Phone 7 describes best practices for designing the UI of a Windows Phone 7 application.

NAVIGATION

The Tailspin mobile client application uses only a small number of pages, with a limited number of navigation routes between those pages.

Figure 3 shows how the user navigates within the application on the phone.

Applications for the phone should be task-based. Users will pick up the device, use the application, and then get on with something else. Users don’t want a complicated application with a lot of different pages.



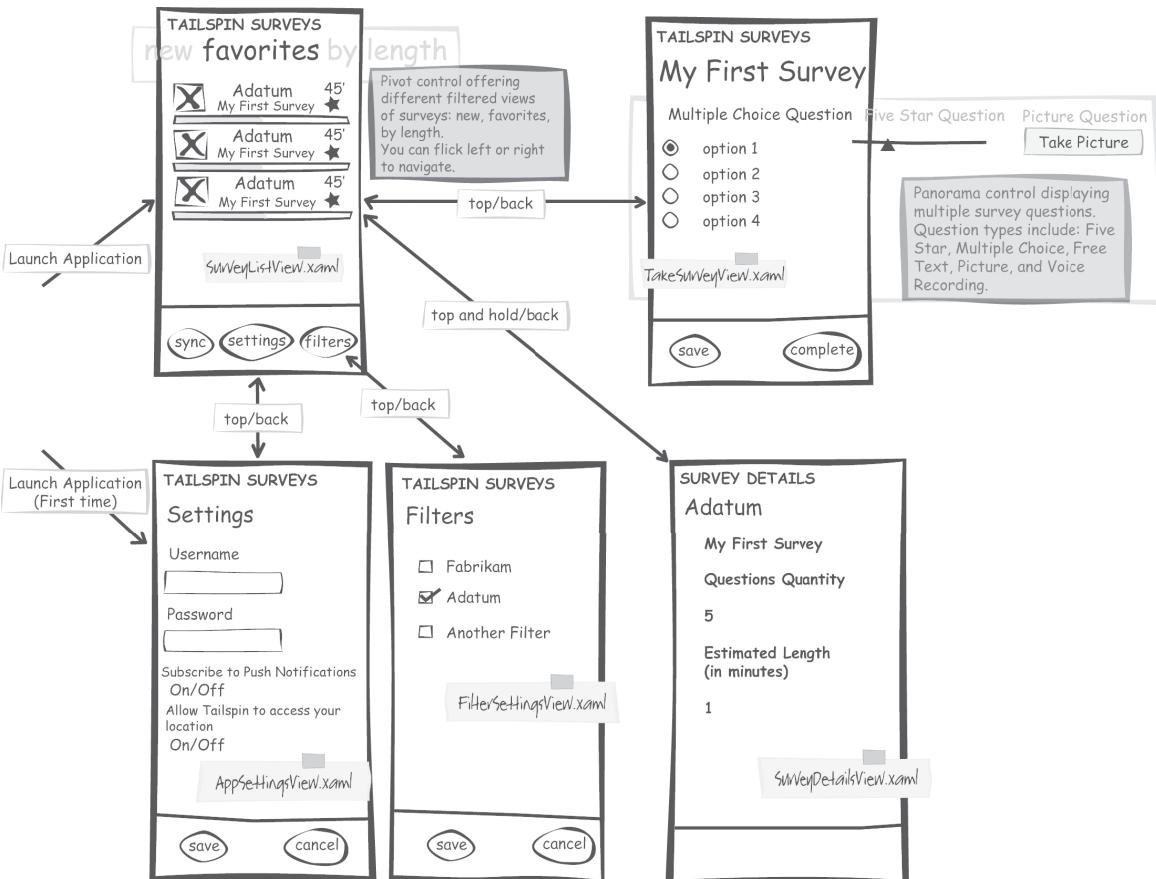


FIGURE 3
Navigation in the Surveys client application

Note: Navigating to the SurveyDetailsView page is done through a context menu that appears when the user taps and holds on a survey name in the list.

The application determines the initial screen to display, based on the **DefaultTask** element in the WMAppManifest.xml file. The following code example shows how the application is configured to first display the SurveyListView page to the user.

```
XML
<Tasks>
  <DefaultTask  Name ="_default"
    NavigationPage="Views/SurveyList/SurveyListView.xaml"/>
</Tasks>
```

Before users can use the application, they must enter the credentials that will be used when the application synchronizes with the Tailspin Surveys service. The developers at Tailspin considered automatically navigating users to the AppSettingsView page if they haven't already supplied their credentials, but this introduced an issue with the way navigation behaves in the application. If the application automatically navigates the user to the AppSettingsView page from the SurveyListView page, and if the user then decides he or she doesn't want to enter credentials (maybe the application was started by mistake), the user will press the Back button and expect to leave the application. A simple approach to navigating will have left the SurveyListView page on the navigation stack, so that's where the user will end up. For some possible solutions to this problem, see the post, "Redirecting an initial navigation," on Peter Torr's blog (<http://blogs.msdn.com/b/ptorr/archive/2010/08/28/redirecting-an-initial-navigation.aspx>).

The current version of the application does not automatically navigate users to the **AppSettingsView** page; instead, it displays a message that explains to users that they must provide their credentials. The following code example from the SurveyListView.xaml file shows how the visibility of the message is controlled based on the value of the **SettingAreNotConfigured** property.

XAML

```
<StackPanel x:Name="SettingNotConfiguredPanel" Grid.Row="0"
    Margin="24,24,0,12"
    Visibility="{Binding SettingAreNotConfigured,
    Converter={StaticResource VisibilityConverter}}">
    <TextBlock x:Name="ApplicationTitle" Text="TAILSPIN SURVEYS"
        Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="Surveys" Margin="-3,-8,0,0"
        Style="{StaticResource PhoneTextTitle1Style}"/>
    <ContentControl Template="{StaticResource
        SettingsNotConfiguredTextBlock}" />
</StackPanel>
```

The following code example from the Styles.xaml file shows the template that defines the message.

XAML

```
<ControlTemplate x:Key="SettingsNotConfiguredTextBlock">
    <TextBlock
        HorizontalAlignment="Stretch" VerticalAlignment="Top"
        Margin="50,50,50,0"
        Style="{StaticResource PhoneTextLargeStyle}"
        Foreground="{StaticResource PhoneSubtleBrush}">
```

```

Text="To start using TailSpin for Windows Phone 7,
configure your Settings."
TextWrapping="Wrap"/>
</ControlTemplate>

```

The **NavigationService** class automatically manages the behavior of the Back button.

When you navigate using the **NavigationService** class, the behavior of the Back button is automatically determined, so using the Back button on the SurveyListView page causes the application to exit, and using the Back button on the AppSettingsView page returns the user to the SurveyListView page.

The following code example from the **AppSettingsViewModel** class shows how the application implements the navigation away from the AppSettingsView page in code for the Cancel button. Notice how the **Cancel** method uses the **NavigationService** class. This class is described in detail later in this chapter in the section, “Handling Navigation Requests.”

```

C#
public void Cancel()
{
    this.NavigationService.GoBack();
}

```

Navigating from the survey list screen to an individual survey is a little more complicated because the application must display the survey that the user currently has selected in the list. Furthermore, the application must respond to the user tapping on an item in a **ListBox** control, on a **PivotItem** in a **PivotControl**.

When the user taps a survey name in the list, the navigation to the TakeSurveyView page is accomplished using a custom behavior in the **SurveyDataTemplate** data template. The following code example from the Styles.xaml page shows this data template definition.

```

XAML
<DataTemplate x:Key="SurveyDataTemplate">
    ...
    <Custom:Interaction.Behaviors>
        <pag:FrameworkElementClickCommand
            CommandBinding="{Binding TakeSurveyCommand}"/>
    </Custom:Interaction.Behaviors>
    ...
</DataTemplate>

```

The custom behavior that enables navigation from a click on an item in the **ListBox** control is defined in the **FrameworkElement-**

ClickCommand infrastructure class. This infrastructure class binds a command to a user tap on an element in the view—in this example, a tap on an item in a list to the **TakeSurveyCommand** command in the **SurveyTemplateViewModel** view model class.

For an explanation of how the **TakeSurveyView** page displays the correct survey, based on the survey selected in the list, see the section, “Connecting the View and the View-model,” later in this chapter.

Tapping and holding on a survey name on the **SurveyListView** page displays a context menu with two options: the user can toggle whether the survey is a favorite to display in the favorites list or the user can display the **SurveyDetailsView** page.

USER INTERFACE DESCRIPTION

Figure 3, earlier in this chapter, shows a mockup of the UI and the navigation routes supported by the application between the pages. There are a few items in the UI that require some additional explanation.

The **SurveyListView** page displays six pieces of information about each survey:

- The survey creator’s logo
- The survey creator’s name
- The survey’s title
- A star to indicate whether the survey is one of the user’s favorites
- A number to indicate how many times the user has completed the survey
- A progress bar to indicate how many questions have been answered so far

The **SurveyDetailsView** page displays some additional information about a survey:

- The survey’s title
- The number of questions in the survey
- An estimate of how long it should take to complete the survey

On the **TakeSurveyView** page, there are two buttons on the application bar. The **Save** button saves the current answers to the survey and allows the user to return to the survey later to amend existing answers and to add additional answers. The **Complete** button saves the current answers and marks the survey as complete and ready for synchronization. You cannot return to or change a completed survey, but you can complete a survey multiple times.

USER INTERFACE ELEMENTS

The application uses standard controls throughout so that the appearance and behavior of the application matches the Windows Phone 7 standard appearance and behavior. The section, “Displaying Data,” later in this chapter describes how the application implements data binding with the **Pivot** and **Panorama** controls.

The Pivot Control

The application uses a **Pivot** control on the SurveyListView page to enable the user to view different filtered lists of surveys, such as all surveys, new surveys, or favorite surveys, or the list of surveys sorted by length. The control allows the user to navigate between the different lists by panning left or right, or by using flick gestures in the application in a way that is consistent with the user’s expectations in the Windows Phone 7 UI. The developers at Tailspin chose to use the **Pivot** control here because it enables you to display a set of items that all have the same data type: in Tailspin Surveys, each **PivotItem** control displays a list of surveys.

The Panorama Control

The application uses the **Panorama** control on the TakeSurveyView page to display the survey questions and collect the survey responses. The developers at Tailspin chose to use the **Panorama** control here because its large scrollable area offers a great way to display a complete survey and long text items, with an intuitive way to navigate by scrolling left or right through the questions. The **Panorama** control also works well when you need to display a set of items with different data types: in the Tailspin Surveys application, the **Panorama** control displays different question types on the same page.

Styling and Control Templates

The file Styles.xaml in the Themes folder contains some styling information for several of the controls used on the pages in the application. The **ListBox** controls that the application uses to display lists of surveys on the SurveyListView page use the SurveyTemplateItemStyle style.

The SurveyListView page uses the NoItemsTextBlock control template to display a message when there are no surveys to display in the **ListBox** control.

The SurveyListView page uses the SettingNotConfiguredPanel control template to display a message when the user hasn’t configured his settings in the application.

The **ThemedResourceLocator** class in the Resources folder shows how the application handles UI changes if the user chooses either the Dark or Light Windows Phone 7 themes. Although most

controls automatically adjust to different UI themes, there are a few places in the Tailspin mobile client that need some additional logic, such as where the application uses custom icons.

Context Menus

On the SurveyListView page, if the user taps a survey name, the application navigates to the TakeSurveyView page and displays the survey questions. If the user taps and holds on a survey name on the SurveyListView page, the application displays a context menu.

Note: You can find the behavior that causes the context menu to display in the **ContextMenu** control itself.

Tailspin uses the **ContextMenu** control from the Microsoft Silverlight® for Windows Phone Toolkit, which is available on the Silverlight Toolkit page on CodePlex (<http://silverlight.codeplex.com>).

The following code example from the data template in the Styles.xaml file shows how Tailspin declares the **ContextMenu** control that displays the context menu when the user taps and holds on a survey name.

XAML

```
<toolkit:ContextMenuService.ContextMenu>
  <toolkit:ContextMenu>
    <toolkit:MenuItem Header="mark as favorite"
      Command="{Binding MarkFavoriteCommand}"
      Visibility="{Binding IsFavorite,
      Converter={StaticResource NegativeVisibilityConverter}}"/>
    <toolkit:MenuItem Header="remove mark as favorite"
      Command="{Binding RemoveFavoriteCommand}"
      Visibility="{Binding IsFavorite,
      Converter={StaticResource VisibilityConverter}}"/>
    <toolkit:MenuItem Header="survey details"
      Command="{Binding ShowDetailsCommand}"/>
  </toolkit:ContextMenu>
</toolkit:ContextMenuService.ContextMenu>
```

Using the Model-View-ViewModel Pattern

Now that you've seen how Tailspin designed the UI of the application, including the choice of controls, and the navigation through the application, it's time to look behind the scenes to discover how Tailspin structured the mobile client for the Windows Phone 7 platform.

The developers at Tailspin decided to adopt a Model-View-ViewModel (MVVM) approach to structuring the Windows Phone 7 application. This section provides an overview of MVVM, explains

The application is designed and built using the MVVM pattern.



MVVM uses concepts familiar to developers who have used other presentation model patterns such as MVC.

why it is appropriate for Windows Phone 7 applications, and highlights some of the decisions made by the developers at Tailspin about the implementation.

THE PREMISE

The MVVM approach lends itself naturally to Silverlight applications. This is because the MVVM pattern is a specialization of the Presentation Model pattern that leverages some of the specific capabilities of Silverlight, such as data binding, commands, and behaviors. MVVM is similar to many other patterns that separate the responsibility for the appearance and layout of the UI from the responsibility for the presentation logic; for example, if you're familiar with the Model-View-Controller (MVC) pattern, you'll find that MVVM has many similar concepts.

Note: *Commands are not supported directly by Silverlight for the Windows Phone 7 platform, but by using behaviors, you can implement them in a friendly way. For more information, see the section, "Commands," later in this chapter.*

Overview of MVVM

There are three core components in the MVVM pattern: the model, the view, and the view model. Figure 4 illustrates the relationships between these three components.

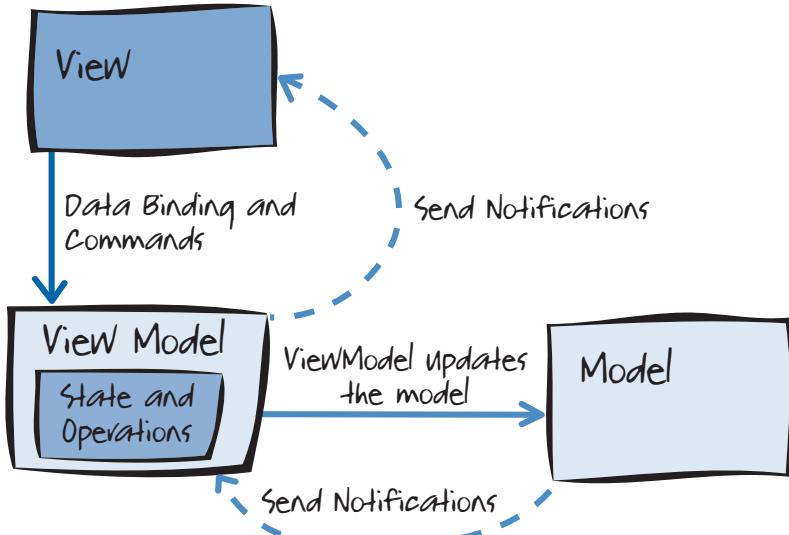


FIGURE 4
The Model-View-ViewModel pattern

The view is responsible for defining the structure, layout, and appearance of what the user sees on the screen. Ideally, the view is defined purely with XAML, with no code-behind other than the standard call to the **InitializeComponent** method in the constructor, although as you've already seen, this is not always possible.

The model in MVVM is an implementation of the application's domain model that can include a data model along with business and validation logic. Often, the model will include a data access layer. In the case of a Windows Phone 7 application, the data access layer could support retrieving and updating data by using a web service or local storage.

The view model acts as an intermediary between the view and the model, and is responsible for handling the view logic. The view model provides data in a form that the view can easily use. The view model retrieves data from the model and then makes that data available to the view, and may reformat the data in some way that makes it simpler for the view to handle. The view model also provides implementations of commands that a user of the application initiates in the view. For example, when a user clicks a button in the UI, that action can trigger a command in the view model. The view model may also be responsible for defining logical state changes that affect some aspect of the display in the view, such as an indication that some operation is pending.

In addition to understanding the responsibilities of the three components, it's also important to understand how the components interact with each other. At the highest level, the view "knows about" the view model, and the view model "knows about" the model, but the model is unaware of the view model, and the view model is unaware of the view.

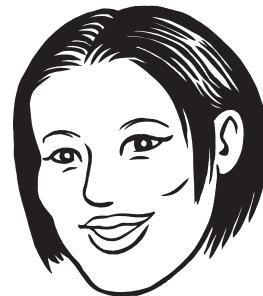
MVVM relies on the data binding capabilities in Silverlight to manage the link between the view and view model. The following are important features of an MVVM application that uses these data binding capabilities:

- The view can display richly formatted data to the user by binding to properties of the view model instance. If the view subscribes to events in the view model, the view model can also notify the view of any changes to its property values using these events.
- The view can initiate operations by using commands to invoke methods in the view model.
- If the view subscribes to events defined in the view model, the view model can notify the view of any validation errors using these events.



In some scenarios, the view model may call a web service directly instead of using a model class that itself makes a call to the web service. For example, if a web service retrieves a collection of **Person** objects that you can deserialize and use directly in the view model, there's no need to define another **Person** class in the model.

The view model isolates the view from the model classes and allows the model to evolve independently of the view.



In Silverlight for the Windows Phone 7 platform, both commands and events are implemented using behaviors.

Note: Tailspin considered two other approaches for linking controls to code. The first was to use custom attached properties, but this meant that they would need to build an attached property for each control's event that they wanted to bind to a command. The second was to invoke a method instead of a command from a binding, but this made it harder to handle the "can execute" type of information that commands neatly encapsulate. The "can execute" information is useful for enabling and disabling controls in the UI.

Typically, the view model interacts with the model by invoking methods in the model classes. The model may also need to be able to report errors back to the view model by using a standard set of events that the view model subscribes to (remember that the model is unaware of the view model). In some scenarios, the model may need to be able to report changes in the underlying data back to the view model; again, the model should do this using events.

This chapter focuses on the view and view model components of the Tailspin mobile client application. Chapter 5, "Using Services on the Phone," includes a description of the model components in the application.

Benefits of MVVM

The following are some of the benefits of clear separation of responsibilities in the MVVM pattern:

- During the development process, developers and designers can work more independently and concurrently on their components. The designers can concentrate on the view, and if they are using Expression Blend, they can easily generate sample data to work with, while the developers can work on the view model and model components.
- The developers can create unit tests for the view model and the model without using the view. The unit tests for the view model can exercise exactly the same functionality as used by the view.
- It is easy to redesign the UI of the application without touching the code because the view is implemented entirely in XAML. A new version of the view should work with the existing view model.
- If there is an existing implementation of the model that encapsulates existing business logic, it may be difficult or risky to change. In this scenario, the view model acts as an adapter for the model classes and enables you to avoid making any major changes to the model code.

Although the benefits of MVVM are clear for a complex application with a relatively long shelf life, the additional work needed to implement the MVVM pattern may not be worthwhile for simple applications or applications with shorter expected lifetimes.

You've seen a high-level description of the MVVM pattern, and the reasons that Tailspin decided to adopt it for their Windows Phone 7 client. The next sections describe the following implementation choices made by the developers at Tailspin when they implemented MVVM for the Surveys application:

- How Tailspin connects the view and the view model components
- Examples of how Tailspin tests components of the application
- How Tailspin implements commands, asynchronous operations, and notifications to the user
- Details of data binding and navigation

For more information about MVVM, see the Prism page on CodePlex (<http://compositewpf.codeplex.com/>).

CONNECTING THE VIEW AND THE VIEW MODEL

Now is a good time to walk through the code that implements the MVVM pattern in the Windows Phone 7 application in more detail. As you go through this section, you may want to download the Microsoft Visual Studio® development system solution for the Windows Phone 7 Tailspin Surveys application; it is available on the Windows Phone 7 Developer Guide community site on CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

There are several ways to connect the view model to the view, including direct relations and data template relations. The developers at Tailspin chose to use a view model locator because this approach means that the application has a single class that is responsible for hooking up views to view models. This still leaves developers free to choose to manually perform the hookup within the view model locator, or by using a dependency injection container.

Inside the Implementation

The mobile client application uses a view model locator to connect view models to views. The following code example from the App.xaml file shows how the view model locator class is identified and made available to the application. The application declares the **TailSpin.PhoneClient.ViewModels.ViewModelLocator** class in the <Application.Resources> section of the App.xaml file.



You should evaluate carefully whether MVVM is appropriate for your application, considering the initial overheads of using this approach.



Tailspin adopted the view model locator approach to connecting the view to a view model. This approach works well for applications with a limited number of screens (which is often the case with Windows Phone 7 applications), but is not always appropriate in larger applications.

XAML

```
<Application
    x:Class="TailSpin.PhoneClient.App"
    ...
    xmlns:viewmodels=
        "clr-namespace:TailSpin.PhoneClient.ViewModels"
    ... >

    <!--Application Resources-->
    <Application.Resources>
        ...
        <viewmodels:ViewModelLocator x:Key="ViewModelLocator"/>
    </Application.Resources>

    ...
</Application>
```

The following code example from the SurveyListView.xaml file shows how a view can then reference the **ViewModelLocator** class as a static resource in its data context bindings.

XAML

```
<phone:PhoneApplicationPage
    x:Class="
        TailSpin.PhoneClient.Views.SurveyList.SurveyListView"
    ...
    DataContext=
        "{Binding Source={StaticResource ViewModelLocator},
        Path=SurveyListViewModel}"
    ...
    >

    ...
</phone:PhoneApplicationPage>
```

The **Path** attribute identifies the property of the **ViewModelLocator** instance that returns the view model associated with the current page.

The following code example shows the parts of the **ViewModelLocator** class that return the **SurveyListViewModel** instance.

C#

```
public class ViewModelLocator : IDisposable
{
    private readonly ContainerLocator containerLocator;
    ...
```

```

public SurveyListViewModel SurveyListViewModel
{
    get
    {
        return this.containerLocator.Container
            .Resolve<SurveyListViewModel>();
    }
}
}

```

Notice how the Tailspin phone client uses the Funq dependency injection container to instantiate the view model.

The following code example shows the parts of the **ViewModelLocator** class that return the **TakeSurveyViewModel** instance when the user selects a survey from the survey list. In this example, the view model is instantiated with details of the currently selected survey so that the application can display the survey questions on the **Take SurveyView** page.

*The **ViewModelLocator** class makes sure that the correct survey is displayed when the user selects a survey from the list.*

C#

```

public TakeSurveyViewModel TakeSurveyViewModel
{
    get
    {
        var templateViewModel =
            this.SurveyListViewModel.SelectedSurveyTemplate;
        var vm = new TakeSurveyViewModel(
            this.containerLocator.Container
            .Resolve<INavigationService>())
        {
            SurveyStoreLocator = this.containerLocator.Container
                .Resolve<ISurveyStoreLocator>(),
            LocationService = this.containerLocator.Container
                .Resolve<ILocationService>(),
            TemplateViewModel = templateViewModel,
            SurveyAnswer = this.containerLocator.Container
                .Resolve<ISurveyStoreLocator>().GetStore()
                .GetCurrentAnswerForTemplate(templateViewModel.Template)
                ?? templateViewModel.Template.CreateSurveyAnswers(
                    this.containerLocator.Container
                    .Resolve<ILocationService>())
        };
        vm.Initialize();
        return vm;
    }
}

```

Note: The view locator is also responsible for instantiating the store and passing it to the view model, which in turn passes it on to the model.

TESTING THE APPLICATION

One of the advantages of the MVVM approach is that it promotes the testability of the application, making it easy to create tests that exercise the view model.

Inside the Implementation

Tailspin uses the Silverlight unit test framework for Windows Phone 7 and Silverlight 3 that's described in Appendix A, "Tools, Frameworks, and Processes."

The Windows Phone 7 project named Tailspin.PhoneClient.Tests contains the unit tests for the Surveys mobile client application. To run the tests, first build and then deploy this project either to the Windows Phone 7 emulator or to a real device. On the Windows Phone 7 device, you can now launch an application named Tailspin.PhoneClient.Tests, and then select the unit tests you want to run.

The following code example shows a unit test method from the **SurveyListViewModelFixture** class that tests that the view model returns a list of all the surveys that are marked as favorites.

Tailspin uses a Silverlight unit-testing framework to run unit tests on the phone emulator and on real devices.



```
C#
[TestMethod]
public void FavoritesSectionShowsFavoritedItems()
{
    var store = new SurveyStoreMock();
    var surveyStoreLocator = new SurveyStoreLocatorMock(store);
    store.Initialize();
    var allSurveys = store.GetSurveyTemplates();

    var vm = new SurveyListViewModel(surveyStoreLocator,
        new SurveysSynchronizationServiceMock());
    vm.Refresh();

    var favoriteSurveys =
        vm.FavoriteItems.Cast<SurveyTemplateViewModel>().ToList();
    CollectionAssert.AreEquivalent(
        allSurveys.Where(p => p.IsFavorite).ToArray(),
        favoriteSurveys.Select(t => t.Template).ToArray());
}
```

This method first instantiates a mock survey store and store locator objects to use in the test. The method then instantiates the view model object from the Tailspin.PhoneClient project to test, passing in the mock store locator object. The method then executes the test on the view model instance and verifies that the favorite surveys in the view are the same as the ones in the underlying database.

DISPLAYING DATA

The application displays data by binding elements of the view to properties in the view model. For example, the **Pivot** control on the SurveyListView page binds to the **SelectedPivotIndex** property in the **SurveyListViewModel** class.

The view can automatically update the values it displays in response to changes in the underlying view model if the view model implements the **INotifyPropertyChanged** interface. In the Tailspin mobile client, the abstract **ViewModel** class inherits from the **NotificationObject** class in the Prism Library. The **NotificationObject** class implements the **INotifyPropertyChanged** interface. All the view model classes in the Tailspin mobile client application inherit from the abstract **ViewModel** class. The application also uses the **ObservableCollection** class from the **System.Collections.ObjectModel** name-space that also implements the **INotifyPropertyChanged** interface.

Note: To learn more about Prism, see the Prism CodePlex site (<http://compositewpf.codeplex.com/>) and “Prism (Composite Client Application Guidance)” on the MSDN® developer program website (<http://msdn.microsoft.com/en-us/library/ff648465.aspx>).

*The view model implements the **INotifyPropertyChanged** interface indirectly through the **NotificationObject** class from the Prism Library.*

Inside the Implementation

The following sections describe examples of data binding in the application. The first section describes a simple scenario on the AppSettingsView page, the next sections describe more complex examples using **Pivot** and **Panorama** controls, and the last section describes how Tailspin addressed an issue with focus events on the phone.

Data Binding on the Settings Screen

The AppSettingsView page illustrates a simple scenario for binding a view to a view model. On this screen, the controls on the screen must display property values from the **AppSettingsViewModel** class, and set the property values in the view model when the user edits the settings values.

Define your data bindings to the view model in the view's XAML.

The following code example shows the **DataContext** attribute and some of the control definitions in the AppSettingsView.xaml file. Tailspin chose to use the **ToggleSwitch** control in place of a standard **CheckBox** control because it better conveys the idea of switching something on and off instead of selecting something. The **Toggle Switch** control is part of the Microsoft Silverlight Windows Phone Toolkit available on the Silverlight Toolkit CodePlex site (<http://silverlight.codeplex.com>).

XAML

```
<phone:PhoneApplicationPage
    x:Class="TailSpin.PhoneClient.Views.AppSettingsView"
    ...
    DataContext="{Binding Source={StaticResource ViewModelLocator},
        Path=AppSettingsViewModel}"
    ...
    ...
    <TextBox Height="Auto" HorizontalAlignment="Stretch"
        Margin="0,28,0,0" Name="textBoxUsername"
        VerticalAlignment="Top" Width="Auto"
        Text="{Binding UserName, Mode=TwoWay}" Padding="0"
        BorderThickness="3">
        <i:Interaction.Behaviors>
            <prism:UpdateTextBindingOnPropertyChanged/>
        </i:Interaction.Behaviors>
    </TextBox>
    ...
    <PasswordBox Height="Auto" HorizontalAlignment="Stretch"
        Margin="0,124,0,0" Name="passwordBoxPassword"
        VerticalAlignment="Top" Width="Auto"
        Password="{Binding Password, Mode=TwoWay}">
        <i:Interaction.Behaviors>
            <prism:UpdatePasswordBindingOnPropertyChanged/>
        </i:Interaction.Behaviors>
    </PasswordBox>
    ...
    <toolkit:ToggleSwitch Header="Subscribe to Push Notifications"
        Margin="0,202,0,0"
        IsChecked="{Binding SubscribeToPushNotifications, Mode=TwoWay}" />
    ...
    <ProgressBar Height="4"
        HorizontalAlignment="Stretch"
```



The default binding mode value is **OneWay**, which is the setting used by the **ProgressBar** control. You need to change this to **TwoWay** if you want to send the changes back to the view model.

```
Name="progressBar"
VerticalAlignment="Bottom"
IsIndeterminate="{Binding IsSyncing}"/>
```

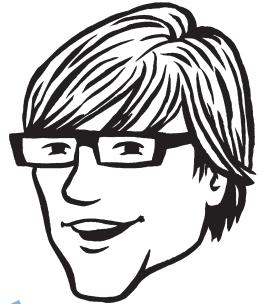
The view model class must implement the **INotifyPropertyChanged** interface for automatic updates in the view to work. In the Tailspin client application, this interface is implemented by the **ViewModel** class that all the view models inherit from. A view model notifies a view of a changed property value by invoking one of the **RaisePropertyChanged** methods. The following code example shows how the **AppSettingsViewModel** view model class notifies the view that it should display the in-progress indicator to the user.

```
C#
public bool IsSyncing
{
    get { return this.isSyncing; }
    set
    {
        this.isSyncing = value;
        this.RaisePropertyChanged(() => this.IsSyncing);
    }
}
```

The code for the **AppSettingsView** page illustrates a solution to a common issue in Silverlight for the Windows Phone 7 platform. By default, the view does not notify the view model of property value changes until the control loses focus. For example, new content in the **textBoxUserName** control is lost if the user tries to save the settings before moving to another control. The **UpdateTextBindingOnPropertyChanged** behavior from the Prism Library ensures that the view always notifies the view model of any changes in the **TextBox** control as soon as they happen. The **UpdatePasswordBindingOnPropertyChanged** behavior does the same for the **PasswordBox** control. For more information, see the section, “Handling Focus Events,” later in this chapter.

Data Binding and the Pivot Control

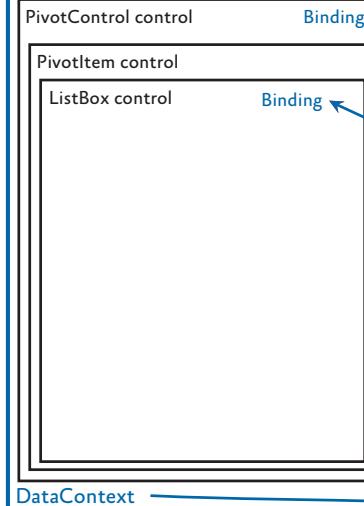
The application uses the **Pivot** control to allow the user to view different filtered lists of surveys. Figure 5 shows the components in the mobile client application that relate to the **Pivot** control as it’s used on the **SurveyListView** page.



The **RaisePropertyChanged** method uses an expression for compile-time verification.

View

SurveyListView.xaml



View Model

SurveyListViewModel class

SelectedPivotIndex property

ICollectionView properties for ListBox control bindings

NewItems property

FavoriteItems property

ByLengthItems property

LocalItems property

ViewModelLocator class

SurveyListViewModel property

FIGURE 5

Using the Pivot control on the SurveyListView page

The following code example shows how the **Pivot** control on the SurveyListView page binds to the **SelectedPivotIndex** property of the **SurveyListViewModel** instance. This two-way binding determines which **PivotItem** control, and therefore which list of surveys, is displayed on the page. Remember, the **ViewModelLocator** class is responsible for locating the correct view model for a view.

XAML

```

<phoneControls:Pivot
  Title="TAILSPIN SURVEYS"
  Name="homePivot"
  SelectedIndex="{Binding SelectedPivotIndex, Mode=TwoWay}"
  Visibility="{Binding SettingAreConfigured,
    Converter={StaticResource VisibilityConverter}}}>
  ...
</phoneControls:PivotControl>
  
```

The following code example shows the definition of the **Pivot Item** control that holds a list of new surveys; it also shows how the **ListBox** control binds to the **NewItems** property in the view model.

XAML

```
<phoneControls:PivotItem Header="new">
  <StackPanel Orientation="Vertical">
    <ContentControl Template="{StaticResource NoItemsTextBlock}"
      Visibility="{Binding NewItems.IsEmpty,
      Converter={StaticResource VisibilityConverter}}" />
    <ListBox ItemsSource="{Binding NewItems}"
      Style="{StaticResource SurveyTemplateItemStyle}"
      Visibility="{Binding NewItems.IsEmpty,
      Converter={StaticResource NegativeVisibilityConverter}}" >
    </ListBox>
  </StackPanel>
</phoneControls:PivotItem>
```

Note: In the list, the layout and formatting of each survey's information is handled by the **SurveyTemplateItemStyle** style and the *SurveyDateTemplate* data template in the *Styles.xaml* file.

The **SurveyListViewModel** class uses **CollectionViewSource** objects to hold the list of surveys to display in the list on each **Pivot Item** control. This allows the view model to notice and to react to changes in the item selected in the view, without needing to know about the view itself. The following code example shows how the **SurveyListViewModel** class defines the properties that the **ListBox** controls bind to.

```
C#
private CollectionViewSource newSurveysViewSource;
private CollectionViewSource byLengthViewSource;
private CollectionViewSource favoritesViewSource;
...
public ICollectionView NewItems { get {
  return this.newSurveysViewSource.View; } }

public ICollectionView FavoriteItems { get {
  return this.favoritesViewSource.View; } }

public ICollectionView ByLengthItems { get {
  return this.byLengthViewSource.View; } }
```

The following code example shows how the **BuildPivot Dimensions** method populates the **CollectionViewSource** objects. In this example, to save space, only the code that populates the **newSurveysViewSource** property is shown.

```
C#
private void BuildPivotDimensions()
{
    this.observableSurveyTemplates =
        new ObservableCollection<SurveyTemplateViewModel>();
    List<SurveyTemplateViewModel> surveyTemplateViewModels =
        this.surveyStoreLocator.GetStore().GetSurveyTemplates()
        .Select(t => new SurveyTemplateViewModel(t)
    {
        Completeness = this.surveyStoreLocator.GetStore()
            .GetCurrentAnswerForTemplate(t) != null
            ? this.surveyStoreLocator.GetStore()
            .GetCurrentAnswerForTemplate(t).CompletenessPercentage
            : 0
    }).ToList();
    surveyTemplateViewModels.ForEach(
        this.observableSurveyTemplates.Add);
    ...
    this.newSurveysViewSource = new CollectionViewSource
    { Source = this.observableSurveyTemplates };
    ...
    this.newSurveysViewSource.Filter +=
        (o, e) => e.Accepted =
            ((SurveyTemplateViewModel)o).Template.IsNew;
    ...
    this.newSurveysViewSource.View.CurrentChanged +=
        (o, e) => this.SelectedSurveyTemplate =
            (SurveyTemplateViewModel)this.newSurveysViewSource
            .View.CurrentItem;
    ...
    // Initialize the selected survey template.
    this.HandleCurrentSectionChanged();
}
```



The **ObservableCollection** class provides notifications when the collection is modified. This means the view automatically updates through the bindings when the data changes.

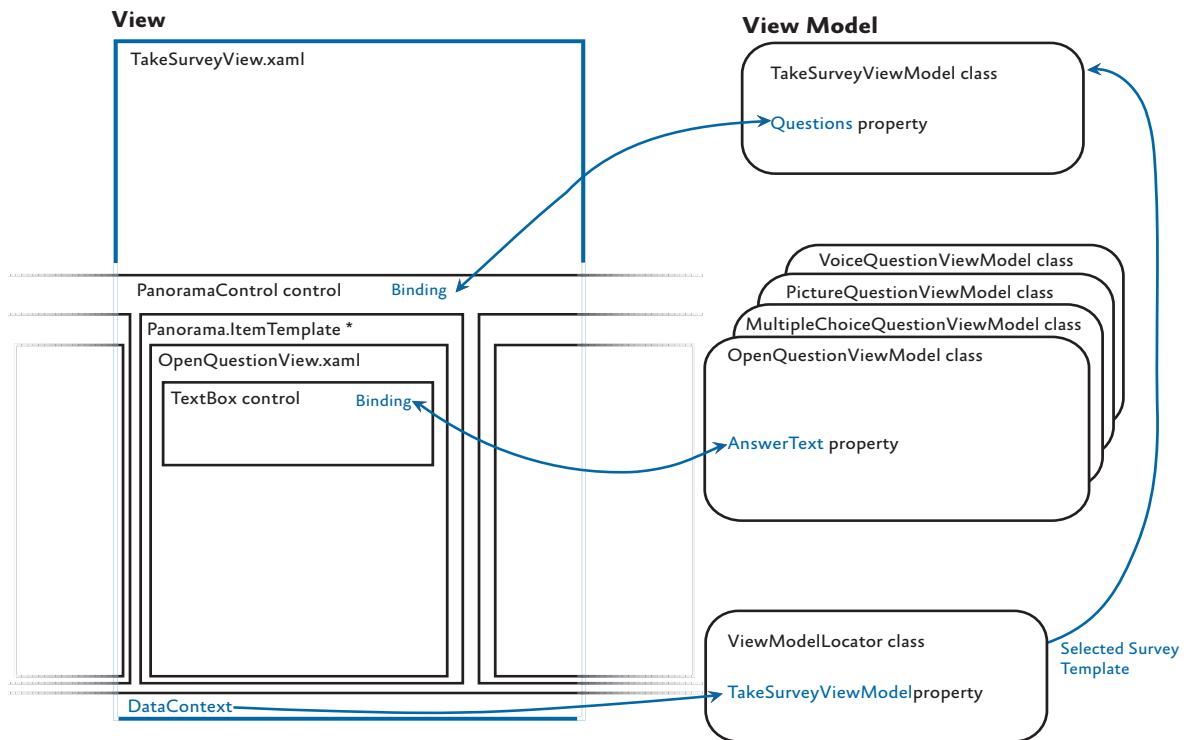
This method first creates an **ObservableCollection** collection of **SurveyTemplateViewModel** objects. Each **SurveyTemplateView Model** object holds a complete definition of a survey and its questions. The method then assigns this collection to the **Source** property of each **CollectionViewSource** object. Next, the method applies a filter or a sort to each **CollectionViewSource** object so that it displays the correct set of surveys. The method then attaches an event

handler to each **CollectionViewSource** object so that the **SelectedSurveyTemplate** property of the **SurveyListViewModel** object is updated correctly. Finally, the method calls the **HandleCurrentSectionChanged** method that causes the view to update and display the currently selected **PivotItem** control with its list of surveys.

Data Binding and the Panorama Control

The application uses the **Panorama** control to display survey questions. This enables the user to scroll left and right through the questions as if the questions are all on a single large page of which the phone's screen shows a small portion.

Figure 6 shows how the **Panorama** control's binding works on the **TakeSurveyView** page.



* The **Panorama.Item** template uses the **ContentTemplateSelector** content selector class from the Prism Library to display the correct view based on the question type. The possible views are: **FiveStarQuestionView**, **MultipleChoiceQuestionView**, **OpenQuestionView**, **PictureQuestionView**, and **VoiceQuestionView**.

FIGURE 6
Using the **Panorama** Control on the **TakeSurveyView** page

As in the previous examples, the view uses the **ViewModel Locator** class to create the view model. In this case, the **ViewModel Locator** object must also tell the view model which survey to display. The following code example shows how the **ViewModelLocator** object instantiates a **TakeSurveyViewModel** object and sets its **TemplateViewModel** property to identify which survey to display, and it creates a new **SurveyAnswer** object to hold the survey question definitions and the answers that the user enters.

C#

```
public TakeSurveyViewModel TakeSurveyViewModel
{
    get
    {
        var templateViewModel =
            this.SurveyListViewModel.SelectedSurveyTemplate;

        var vm = new TakeSurveyViewModel(this.containerLocator
            .Container.Resolve<INavigationService>())
        {
            SurveyStoreLocator = this.containerLocator.Container
                .Resolve<ISurveyStoreLocator>(),
            LocationService = this.containerLocator.Container
                .Resolve<ILocationService>(),
            TemplateViewModel = templateViewModel,
            SurveyAnswer = this.containerLocator.Container
                .Resolve<ISurveyStoreLocator>().GetStore()
                .GetCurrentAnswerForTemplate(templateViewModel.Template)
                ?? templateViewModel.Template.CreateSurveyAnswers(this
                    .containerLocator.Container.Resolve<ILocationService>())
        };
        vm.Initialize();
        return vm;
    }
}
```

The **Panorama** control binds to the **Questions** property of the **TakeSurveyViewModel** class and a **Panorama.ItemTemplate** template controls the display of each question in the survey. However, it's necessary to display different question types using different layouts. The following code example from the **TakeSurveyView.xaml** file shows how the data binding and view layout is defined for the **Panorama** control using the **DataTemplateSelector** content selector class from the Prism Library.

XAML

```
<phoneControls:Panorama
    SelectionChanged="PanoramaSelectionChanged"
    Loaded="PanoramaLoaded"
    VerticalAlignment="Top"
    Name="panorama" Margin="0,0,0,0"
    ItemsSource="{Binding Questions}">
    <phoneControls:Panorama.ItemTemplate>
        <DataTemplate>
            <ScrollViewer>
                <prism:DataTemplateSelector Content="{Binding}">
                    Grid.Row="0" VerticalAlignment="Top"
                    HorizontalContentAlignment="Left" IsTabStop="False">
                    <prism:DataTemplateSelector.Resources>
                        <DataTemplate x:Key="OpenQuestionViewModel">
                            <Views:OpenQuestionView DataContext="{Binding}" />
                        </DataTemplate>
                        <DataTemplate
                            x:Key="MultipleChoiceQuestionViewModel">
                            <Views:MultipleChoiceQuestionView
                                DataContext="{Binding}" />
                        </DataTemplate>
                        ...
                    </prism:DataTemplateSelector.Resources>
                </prism:DataTemplateSelector>
            </ScrollViewer>
        </DataTemplate>
    </phoneControls:Panorama.ItemTemplate>
    <phoneControls:Panorama.HeaderTemplate>
        <DataTemplate>
            <Canvas Width="0" Height="0" />
        </DataTemplate>
    </phoneControls:Panorama.HeaderTemplate>
</phoneControls:Panorama>
```

Note: *It is necessary to create a HeaderTemplate data template even if you’re not using it with the **Panorama** control. You’ll also notice that the XAML declares handlers for the **SelectionChanged** and **Loaded** events in the code-behind. For an explanation of why the developers at Tailspin used code-behind here, see the section, “Handling Activation and Deactivation, ” in Chapter 5, “Using Services on the Phone.” The code-behind also contains a workaround method to trim long titles that don’t always display correctly when the user scrolls in the **Panorama** control.*

Each question view on the **Panorama** control binds to a view model object in the list of questions held in the **Questions** property of the **TakeSurveyViewModel** class. For example, an **OpenQuestionView** view binds to an **OpenQuestionViewModel** object, and a **VoiceQuestionView** view binds to a **VoiceQuestionViewModel** object. The following code example shows how the **TakeSurveyViewModel** class builds this list of question view models.

```
C#
public IList<QuestionViewModel> Questions { get; set; }
...
public void Initialize(ISurveyStoreLocator surveyStoreLocator)
{
    ...
    this.Questions = this.SurveyAnswer.Answers.Select(
        a => Maps[a.QuestionType].Invoke(a)).ToArray();
}
```

The following code sample shows the definition of the **Maps** property in the **TakeSurveyViewModel**. The **Maps** property maps question types to view models.

```
C#
private static readonly
    Dictionary<QuestionType, Func<QuestionAnswer,
    QuestionViewModel>> Maps =
    new Dictionary<QuestionType, Func<QuestionAnswer,
    QuestionViewModel>>()
{
    { QuestionType.SimpleText,
        a => new OpenQuestionViewModel(a) },
    { QuestionType.MultipleChoice,
        a => new MultipleChoiceQuestionViewModel(a) },
    ...
};
```

Handling Focus Events

The file **OpenQuestionView.xaml** defines the UI for entering survey results. Tailspin found that when the user clicked the **Save** button, the last answer wasn't saved by the view model. This is because **ApplicationBarIconButton** control is not a **FrameworkElement** control, so it cannot get the focus. As a result, the lost focus event on the text box wasn't being raised; as a result, the bindings didn't tell the view model about the new field contents.

To solve this problem, the developers at Tailspin used a behavior named **UpdateTextBindingOnPropertyChanged** from the Prism

Library. This behavior ensures that the view notifies the view model whenever the user changes the text in the **TextBox** control, not just when the control loses focus. The following code example shows how this behavior is defined in OpenQuestionView.xaml.

```
C#
...
xmlns:prism=
"clr-namespace:Microsoft.Practices.Prism.Interactivity;
assembly=Microsoft.Practices.Prism.Interactivity"
...
<TextBox Text="{Binding AnswerText, Mode=TwoWay}">
  <Interactivity:Interaction.Behaviors>
    <prism:UpdateTextBindingOnPropertyChanged/>
  </Interactivity:Interaction.Behaviors>
</TextBox>
```



COMMANDS

In a Silverlight application, you can invoke some action in response to a user action (such as a button click) by creating an event handler in the code-behind class. However, in the MVVM pattern, the responsibility for implementing the action lies with the view model, and you should avoid placing code in the code-behind classes. Therefore, you should connect the control to a method in the view model using a binding.

The version of Silverlight on the Windows Phone 7 platform does not currently support binding to a command that implements the **ICommand** interface like you can do in Windows Presentation Foundation (WPF). Therefore, Tailspin uses a set of binding behaviors, including the **ApplicationBarButtonNavigation**, **AppBarButtonCommand**, and **ButtonCommand** behaviors, from the Prism Library to locate commands from the view.

For more information about the **ICommand** interface in WPF, see “Commanding Overview” on MSDN (<http://msdn.microsoft.com/en-us/library/ms752308.aspx>).

To keep the responsibilities of the view and view model separate, you should try to avoid placing code in the code-behind files of your views when you are implementing the MVVM pattern.

*Windows Phone 7 controls do not currently support binding to **ICommand** instances.*

Inside the Implementation

The developers at Tailspin use bindings to associate actions in the UI with commands in the view model. In addition to Silverlight controls on the Windows Phone 7 platform not supporting binding to **ICommand** instances, you cannot use the **InvokeCommandAction** Expression Blend behavior with the **ApplicationBarIconButton** or **ApplicationBarMenuItems** controls because they cannot have dependency properties. Tailspin uses a custom behavior to hook up commands to the view model.

The following code example from the **SurveyListView** page shows how the **Synchronize** button and the **Settings** button on the application bar are associated with commands.

XAML

```

<phoneNavigation:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar IsVisible="True">
    <shell:ApplicationBarIconButton Text="Sync" IconUri="..." />
    <shell:ApplicationBarIconButton Text="Settings"
      IconUri="..." />
    <shell:ApplicationBarIconButton Text="Filters" IconUri="..." />
  </shell:ApplicationBar>
</phoneNavigation:PhoneApplicationPage.ApplicationBar>
  ...
<Custom:Interaction.Behaviors>
  <prismInteractivity:ApplicationBarButtonCommand
    ButtonText="Sync"
    CommandBinding="{Binding StartSyncCommand}"/>
  <prismInteractivity:ApplicationBarButtonCommand
    ButtonText="Settings"
    CommandBinding="{Binding AppSettingsCommand}"/>
  <prismInteractivity:ApplicationBarButtonCommand
    ButtonText="Filters"
    CommandBinding="{Binding FiltersCommand}"/>
  ...
</Custom:Interaction.Behaviors>

```

The attributes attached to the **ApplicationBarIconButton** controls (**Text** and **IconUri**) only affect their appearance. The **ApplicationBarButtonCommand** elements handle the connection to the commands; they identify the control to associate with the command through the **ButtonText** attribute and the command binding through the **CommandBinding** attribute.

The **ApplicationBarButtonCommand** class from the Prism Library implements the custom behavior and sets up the binding that links a button click in the UI to the **StartSyncCommand**, **AppSettingsCommand**, and **FiltersCommand** properties in the **SurveyListView Model** class.

The **StartSyncCommand** property uses an instance of the **DelegateCommand** class that implements the **ICommand** interface. The following code example from the **SurveyListViewModel** class shows the definition of the **StartSyncCommand** property.

```
C#
public DelegateCommand StartSyncCommand { get; set; }
...
public SurveyListViewModel(...) : base(...)
{
    ...
    this.StartSyncCommand = new DelegateCommand(
        () => { this.StartSync(); },
        () => !this.IsSynchronizing &&
            !this.SettingsAreNotConfigured);
    ...
}
```

Note: For more information about the **DelegateCommand** class from Prism, and to learn more about Prism, see the Prism CodePlex site (<http://compositewpf.codeplex.com/>).

The following code example from the **SurveyListViewModel** class shows the implementation of the **StartSync** method and **SyncCompleted** method. These methods run the synchronization process asynchronously by invoking the **StartSynchronization** method; they also control the progress bar in the UI by setting the **IsSynchronizing** property. This method uses the **ObserveOnDispatcher** method from the Reactive Extensions (Rx) for .NET to run the synchronization process. For more information about how Tailspin uses Rx, see Chapter 5, “Using Services on the Phone.”

```
C#
public void StartSync()
{
    if (this.IsSynchronizing)
    {
        return;
    }

    this.IsSynchronizing = true;
    this.synchronizationService
        .StartSynchronization()
        .ObserveOnDispatcher()
        .Subscribe(
            taskSummaries =>
            {
                this.SyncCompleted(taskSummaries);
                this.IsSynchronizing = false;
            }
        );
}
```

```

    });
}

private void SyncCompleted(
    IEnumerable<TaskCompletedSummary> taskSummaries)
{
    ...
    this.BuildPivotDimensions();
    this.RaisePropertyChanged(string.Empty);
    this.UpdateCommandsForSync();
}

```

The **SyncCompleted** method also updates the UI to show the new list of surveys following the synchronization process. The **UpdateCommandsForSync** method disables the **Synchronize** button in the UI while the synchronization is running.

HANDLING NAVIGATION REQUESTS

In addition to invoking commands from the view, the Tailspin mobile client also triggers navigation requests from the view. These requests could be to navigate to a particular view or navigate back to the previous view. In some scenarios, for example if the application needs to navigate to a new view when a command completes, this requires the view model to send a message to the view. In other scenarios, you might want to trigger the navigation request directly from the view without involving the view model directly. When you're using the MVVM pattern, you want to be able to do all this without using any code-behind in the view, and without introducing any dependency on the view implementation in the view model classes.

Inside the Implementation

The following code example from the FilterSettingsView.xaml file shows two examples of initiating navigation in the sample application.

XAML

```

<i:Interaction.Behaviors>
    <prismInteractivity:ApplicationBarButtonCommand
        ButtonText="Save" CommandBinding="{Binding SaveCommand}"/>
    <prismInteractivity:ApplicationBarButtonNavigation
        ButtonText="Cancel" NavigateTo="#GoBack" />
    ...
</i:Interaction.Behaviors>

```

The first example invokes the **SaveCommand** command in the view model. The code that implements this command causes the application to navigate back to the previous view if the command suc-

ceeds, so in this example, the navigation is initiated from the view model. The following code example from the **FilterSettingsView Model** class illustrates this.

```
C#
public DelegateCommand SaveCommand { get; set; }

...
public FilterSettingsViewModel(...)
{
    ...
    this.SaveCommand =
        new DelegateCommand(this.Submit, () => !this.CanSubmit);
    ...
}

...
public void Submit()
{
    ...
    this.NavigationService.GoBack();
    ...
}

...
public bool CanSubmit
{
    get { return this.canSubmit; }
    set
    {
        if (!value.Equals(this.canSubmit))
        {
            this.canSubmit = value;
            this RaisePropertyChanged(() => this.CanSubmit);
        }
    }
}
```

The second example shows how the view can initiate the navigation directly by navigating back to the previous view (in this case, the SurveyListView page) without executing a command.

Both examples use the **ApplicationFrameNavigationService** class shown in the following code example to perform the navigation.

```
C#
public class ApplicationFrameNavigationService :
    INavigationService
{
    private readonly PhoneApplicationFrame frame;
```



```
public ApplicationFrameNavigationService(  
    PhoneApplicationFrame frame)  
{  
    this.frame = frame;  
}  
  
public bool Navigate(Uri source)  
{  
    return this.frame.Navigate(source);  
}  
  
public void GoBack()  
{  
    this.frame.GoBack();  
}  
  
public bool CanGoBack  
{  
    get { return this.frame.CanGoBack; }  
}  
  
public Uri CurrentSource  
{  
    get { return this.frame.CurrentSource; }  
}
```

Using the **PhoneApplicationFrame** instance ensures that the phone maintains the correct navigation stack for the application so that navigating backward works in the way that users expect.

This class, which implements Tailspin's **INavigationService** interface, uses the phone's **PhoneApplicationFrame** instance to perform the navigation request for the application.

A view model can invoke the **Navigate** method on the **ApplicationFrameNavigationService** object to cause the application to navigate to a particular view in the application or the **GoBack** method to return to the previous view.

The **ViewModel** base class maintains the **INavigationService** instance for all the view models, and the Funq dependency injection container is responsible for initially creating the **ApplicationFrameNavigationService** object that implements this interface.

To avoid any code-behind in the view when the view initiates the navigation, the developers at Tailspin use an interaction behavior from the Prism Library. The following code example shows how the **Cancel** button is declared in the FilterSettingsView.xaml file.

XAML

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar ...>
    ...
    <shell:ApplicationBarIconButton Text="Cancel"
      IconUri="..." />
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
<i:Interaction.Behaviors>
  ...
  <prismInteractivity:ApplicationBarButtonNavigation
    ButtonText="Cancel" NavigateTo="#GoBack" />
  ...
</i:Interaction.Behaviors>
```

The **ApplicationBarButtonNavigation** behavior object contains a reference to the view model, and the view model contains the reference to the **ApplicationFrameNavigationService** instance that handles the navigation request. The **ApplicationBarButtonNavigation** behavior object checks for the special value “**#GoBack**”, in which case, it uses the navigation service to navigate back to the previous view; otherwise, it uses the value of the **NavigateTo** attribute to identify the view to navigate to.



By using a behavior, Tailspin avoids having any code in the view to handle navigation requests. This general pattern is the way for the view model to handle requests from the view without using “classic” events that require handlers in code-behind.

USER INTERFACE NOTIFICATIONS

The Tailspin Surveys mobile client application performs some tasks asynchronously; one example is the potentially time-consuming synchronization with the Tailspin Surveys web service. Asynchronous tasks must often inform the user of the outcome of the task or provide status information while they’re running. It’s important to consider the usability of the application when you’re deciding on the appropriate way to notify users of significant events or to provide status information. The developers at Tailspin were concerned they would either flood the user with alerts or have the user miss an important piece of information.

For the Tailspin Surveys mobile client application, the developers identified two categories of notification, informational/warning notifications and error notifications.

Informational/Warning Notifications

Informational or warning notifications should not be disruptive, so the user should see the message but not be interrupted in their current task. The user does not need to perform any action in response to this type of message; the Tailspin mobile client application uses this type of notification to inform the user when a synchronization completes

Tailspin implemented a custom toast notification system.

successfully, for example. Tailspin uses a custom toast notification for these messages because the application does not have access to the Windows Phone 7 toast notification system.

Error Notifications

Error notifications should be disruptive because the message is informing the user that some expected action of the application will not happen. The notification should inform the user of the actions they need to take to resolve the problem; for example, to retry the synchronization operation again if it fails for some reason. Tailspin uses message boxes for this type of message.

Inside the Implementation

This example shows how Tailspin implements custom toast notifications and error notifications on the SurveyListView page to inform users when the synchronization process finishes or fails. In the sample application, many of the notifications and error messages are not intended for the imaginary user of the sample; instead, they are there to help you, the developer understand what the application is doing as you explore its functionality. You should follow the guidance published in the *UI Design and Interaction Guide for Windows Phone 7* (<http://go.microsoft.com/fwlink/?LinkId=183218>) when you design the user notification system for your application.

The following code example shows the relevant declarations in the SurveyListView.xaml file.

XAML

```
<prismInteractionRequest:MessageBoxRequestTrigger RequestBinding= " {Binding SubmitErrorInteractionRequest} "/>
<prismInteractionRequest:ToastRequestTrigger RequestBinding= " {Binding SubmitNotificationInteractionRequest} "
    PopupElementName= " SynchronizationToast " />
...
<Popup x:Name= " SynchronizationToast " >
    <Grid Background= " {StaticResource PhoneAccentBrush} " VerticalAlignment= " Bottom " Width= " 480 " >
        <TextBlock Text= " {Binding Title} " HorizontalAlignment= " Stretch " Foreground= " Black " TextWrapping= " Wrap " Margin= " 14,5,14,5 " >
            <Custom:Interaction.Behaviors>
                <pag:PopupHideOnLeftMouseUp />
            </Custom:Interaction.Behaviors>
        </TextBlock>
    </Grid>
</Popup>
```

The view model uses the **SubmitNotificationInteractionRequest** binding to trigger the toast notification and the **SubmitErrorInteractionRequest** binding to trigger the error message notification. The following code example shows how the **SurveyListViewModel** displays a toast notification when the synchronization process completes successfully and an error message when it fails.

```
C#
private readonly InteractionRequest<Microsoft.Practices.Prism.Interactivity.InteractionRequest> submitErrorInteractionRequest;
private readonly InteractionRequest<Microsoft.Practices.Prism.Interactivity.InteractionRequest> submitNotificationInteractionRequest;
...
public IInteractionRequest SubmitErrorInteractionRequest
{ get { return this.submitErrorInteractionRequest; } }

public IInteractionRequest SubmitNotificationInteractionRequest
{ get { return this.submitNotificationInteractionRequest; } }

...
private void SyncCompleted(
    IEnumerable<TaskCompletedSummary> taskSummaries)
{
    ...
    if (taskSummaries.Any(
        t => t.Result != TaskSummaryResult.Success))
    {
        this.submitErrorInteractionRequest.Raise(
            new Microsoft.Practices.Prism.Interactivity.InteractionRequest.Notification
            { Title = "Synchronization error", Content =
                stringBuilder.ToString() },
            n => { });
    }
    else
    {
        this.submitNotificationInteractionRequest.Raise(
            new Microsoft.Practices.Prism.Interactivity.InteractionRequest.Notification
            { Title = stringBuilder.ToString(), Content = null },
            n => { });
    }
    ...
}
```

Note: This solution uses the **InteractionRequest** and **Notification** classes and two custom behaviors, **Message BoxRequestTrigger** and **ToastRequestTrigger**, from the Prism Library.

ACCESSING SERVICES

The MVVM pattern identifies three major components: the view, the view model, and the model. This chapter describes the UI of the Tailspin Surveys mobile client application and the way that Tailspin uses the MVVM pattern. The Tailspin mobile client application also includes a number of services. These services can be invoked from the view, view model, or model components and include services that do the following:

- Manage the settings and surveys stores that handle data persistence for the application.
- Save and load the application's state when it's activated and deactivated.
- Synchronize survey data between the client application and the Tailspin Surveys web application.
- Notify the application when new surveys are available.
- Encode audio data, and support application navigation and other infrastructure services.

These services are discussed in following chapters. Chapter 5, "Using Services on the Phone," describes how the application uses services offered by the Windows Phone 7 platform, such as local data persistence services and geo-location services. Chapter 6, "Connecting with Services," describes how the mobile client application accesses services over the network.

Questions

1. Which of the following are good reasons to use the MVVM pattern for your Windows Phone 7 application?
 - a. It improves the testability of your application.
 - b. It facilitates porting the application to another platform, such as the desktop.
 - c. It helps to make it possible for designers and developers to work in parallel.
 - d. It may help you avoid risky changes to existing model classes.

2. Which of the following are good reasons to not use the MVVM pattern for your Windows Phone 7 application?
 - a. You have a very tight deadline to release the application.
 - b. Your application is relatively simple with only two screens and no complex logic to implement.
 - c. Windows Phone 7 controls are not ideally suited to the MVVM pattern.
 - d. It's unlikely that your application will be used for more than six months before it is completely replaced.
3. Which of the following differentiate **Pivot** and **Panorama** controls?
 - a. The **Pivot** control enables navigation in two dimensions; the **Panorama** control only supports navigation in a single dimension.
 - b. The **Panorama** control provides a large canvas that extends beyond the confines of the screen. The **Pivot** control allows the user to navigate between different views.
 - c. The **Pivot** control is ideally suited to displaying filtered lists.
 - d. The **Panorama** control is ideally suited to displaying media items.
4. Which of the following describe the role of the view model locator?
 - a. The view model locator configures bindings in the MVVM pattern.
 - b. In the Tailspin mobile client, the view model locator is responsible for instantiating view-model objects.
 - c. The view model locator hooks up views to view models.
 - d. Data template relations offer an alternative approach to a view model locator.
5. Where does the Back button take you?
 - a. To the previous view in the navigation stack.
 - b. It depends on what the code in the view model does.
 - c. If the current view is the last one in the navigation stack, you leave the application.
 - d. If your application is on the top of the phone's application stack, it takes you back to your application.

6. Why should you not use code-behind when you're using the MVVM pattern?
 - a. The view model locator always intercepts the events, so code-behind code never executes.
 - b. The MVVM pattern enforces a separation of responsibilities between the view and the view model. UI logic belongs in the view model.
 - c. If you are using the MVVM pattern, other developers will expect to see your code in the view model classes and not in the code-behind.
 - d. Code-behind has a negative effect on view performance.

More Information

For more information about designing a Windows Phone 7 UI, see “Themes for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff769554\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769554(VS.92).aspx)

For more information about the **Panorama** control, see “Panorama Control for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff941092\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941092(VS.92).aspx)

For more information about the **Pivot** control, see “Pivot Control for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff941123\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941123(VS.92).aspx)

For more information about the navigation on the Windows Phone 7 platform, see “Frame and Page Navigation for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff941091\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941091(VS.92).aspx)

For more information about Prism and MVVM see the Prism CodePlex site and “Prism (Composite Client Application Guidance)” on MSDN: <http://compositewpf.codeplex.com/> <http://msdn.microsoft.com/en-us/library/ff648465.aspx>

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

This chapter describes how the Tailspin mobile client uses services offered by the Windows Phone 7 platform. The chapter begins by describing the various model classes used in the application to represent data within the application. The view model classes described in Chapter 4, “Building the Mobile Client,” make extensive use of these model classes as they manage the data displayed and created in the user interface (UI). This chapter focuses on how the mobile client application uses the isolated storage service on the phone to persist this data, and how the application can populate these model classes with previously saved data. The chapter also discusses issues that relate to data security on the phone.

When the Windows Phone 7 operating system deactivates an application, it’s the application’s responsibility to persist enough state information to be able to restore the state of the application if and when it’s reactivated by the operating system. This chapter describes how the Tailspin mobile client uses the services offered by the phone to support this behavior.

The mobile client application can also send and receive data from the Tailspin cloud service, and this chapter describes how the application synchronizes data between the phone and the cloud. The focus of this chapter is the way that the mobile client application can use services on the phone to help it run the synchronization tasks asynchronously and in parallel.

The chapter also describes how the mobile client application uses other phone services to acquire geo-location data, image data, and audio data.

The Model Classes

Chapter 4, “Building the Mobile Client,” described the Model-View-ViewModel (MVVM) pattern adopted by the developers at Tailspin. This chapter relates to the model elements of this pattern that represent the domain entities used in the application. Figure 1 shows the key model classes and the relationships between them.

Model classes represent domain entities in the Model-View-ViewModel pattern.

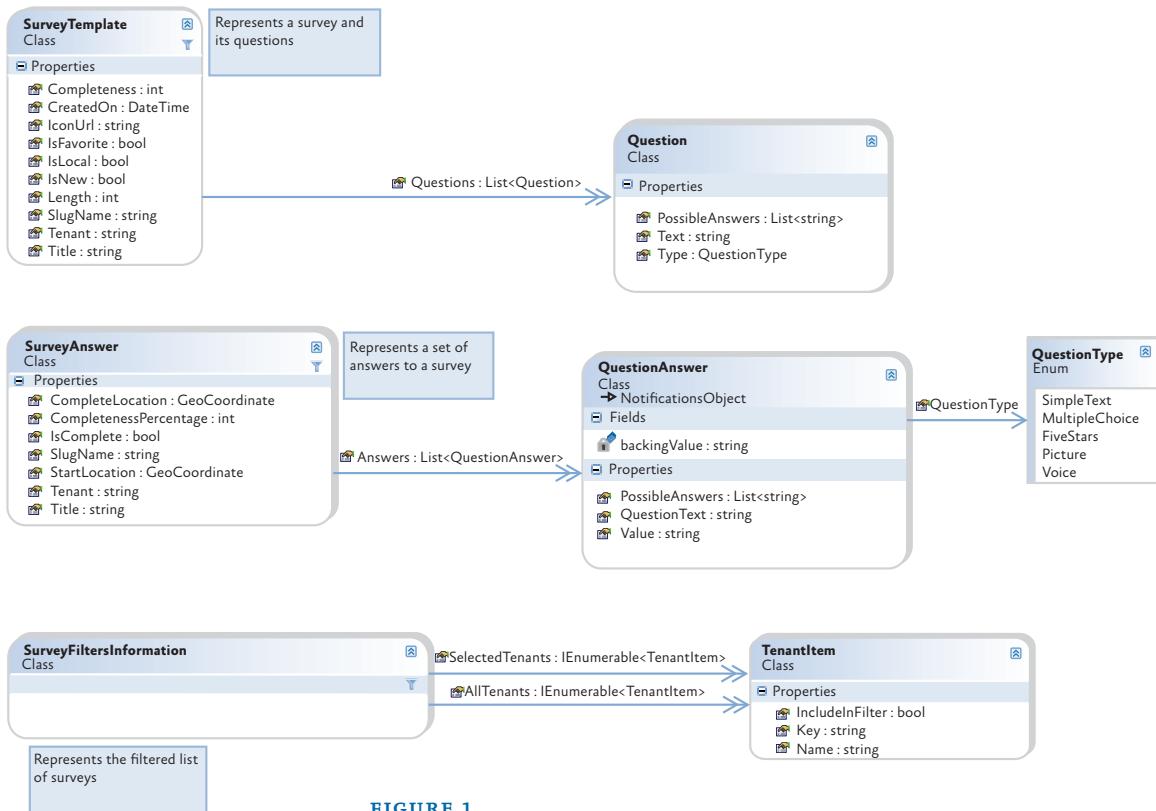


FIGURE 1
The model classes in the Tailspin mobile client application

Using Isolated Storage on the Phone

Isolated storage is the only mechanism available to persist application data on the Windows Phone 7 platform.

The Windows Phone 7 Surveys application must be able to operate when there is no available network connection. Therefore, it must be able to store survey definitions and survey answers locally on the device, together with any other data or settings that the application requires to operate.

The application must behave as a “good citizen” on the device. It is not possible to access another application’s isolated storage or access the file system directly on the Windows Phone 7 platform, so the application cannot compromise another application by reading or modifying its data. However, there is a limited amount of storage available on the phone, so Tailspin tries to minimize the amount of data that the mobile client application stores locally and be proactive about removing unused data.

The Tailspin mobile client application must adopt a robust storage solution and minimize the risk of losing any stored data.

OVERVIEW OF THE SOLUTION

The Windows Phone 7 platform offers isolated storage that works in the same way that the isolated storage APIs for the Microsoft® Silverlight® browser plug-in work in Windows. Isolated storage provides a dictionary-based model for storing application settings, and it enables the application to create folders and files that are private to the application.

Figure 2 shows how each application on a Windows Phone 7 device only has access to its own, private storage on the device, and cannot access the storage belonging to other applications.

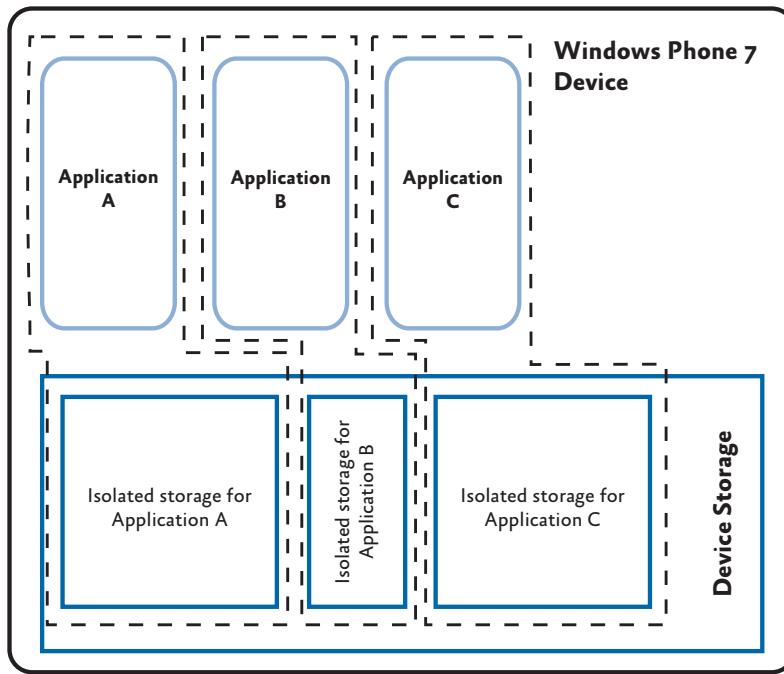


FIGURE 2
Isolated storage on the Windows Phone 7 platform

Security

Tailspin does not encrypt the data that the mobile client application stores in isolated storage because it does not consider the data that the application uses to be confidential. This may not be true of all Windows Phone 7 applications, and you must consider whether your application should take steps to protect any data that it stores on the phone.



To be a “good citizen” on the phone, your application should minimize the amount of isolated storage it uses. There’s nothing on the phone that enforces any quotas, it’s your responsibility.



It is the application’s responsibility to manage the amount of storage it uses by deleting temporary or unused data because the operating system does not enforce any quota limits. Tailspin removes completed surveys from isolated storage after they are synchronized to the Tailspin Surveys service.



Isolated storage provides one level of data security, isolation from other applications on the device, but you should consider whether this is adequate for the security needs of your application.

Note: *The sample application stores user names and passwords in plain text in isolated storage. In a real application, you should take steps to protect any stored credentials, perhaps by using a salt and hash when you store passwords in isolated storage.*

There are two scenarios to think about when you are implementing security for data stored on the phone. The first is whether other applications on the phone could potentially access your application's data and then transmit it to someone else. The isolated storage model used on Windows Phone 7 that limits applications to their own storage makes this a very unlikely scenario. However, security best practices suggest that you should guard against even unlikely scenarios so you may want to consider encrypting the data in your application's isolated storage.

The second scenario to consider is what happens if an unauthorized user gains access to the device. If you want to protect your data in this scenario, you must encrypt the data while it is stored on the device. The Windows Phone 7 API includes support for several cryptographic algorithms that can help you securely encrypt your data.

For more information, see the section, "Security Considerations," in Chapter 2, "Designing Applications for Windows Phone 7."

Storage Format

Tailspin used isolated storage and serializable model classes to store the application's setting and survey data. The initial release of the Windows Phone 7 platform does not include a version of SQL Server Compact edition, but the developers at Tailspin would like to have the option to move to this storage platform if it becomes available in the future. They have implemented the storage service in the application in a way that makes it easy to replace the storage classes with alternative implementations if they decide to use a different storage technology in the future. The current implementation also makes it easy to test the storage functionality in the application.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements isolated storage in the Tailspin mobile application in more detail. As you go through this section, you may want to download the Microsoft Visual Studio® development system solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

You can find the code that implements isolated storage access in the Tailspin mobile client application in the Services/Stores folder in the TailSpin.PhoneClient.

Application Settings

The user enters application settings data on the AppSettingsView and FilterSettingsView pages in the Surveys application. The interface **ISettingsStore** defines the data items that the application saves as settings. The following code example shows this interface.

```
C#
public interface ISettingsStore
{
    string Password { get; set; }
    string UserName { get; set; }
    bool SubscribeToPushNotifications { get; set; }
    bool LocationServiceAllowed { get; set; }
}
```

The following code example shows how the application implements this interface to save the value of the **Password** property in the **ApplicationSettings** dictionary in the application's isolated storage.

```
C#
public class SettingsStore : ISettingsStore
{
    private const string PasswordSettingDefault = "";
    private const string PasswordSettingKeyName =
        "PasswordSetting";

    ...

    private readonly IsolatedStorageSettings isolatedStore;

    public SettingsStore()
    {
        this.isolatedStore =
            IsolatedStorageSettings.ApplicationSettings;
    }

    public string Password
    {
        get { return this.GetValueOrDefault(PasswordSettingKeyName,
            PasswordSettingDefault); }
        set { this.AddOrUpdateValue(PasswordSettingKeyName, value); }
    }

    ...

    private void AddOrUpdateValue(string key, object value)
```

```
{  
    bool valueChanged = false;  
  
    try  
{  
        // If the new value is different, set the new value.  
        if (this.isolatedStore[key] != value)  
        {  
            this.isolatedStore[key] = value;  
            valueChanged = true;  
        }  
    }  
    catch (KeyNotFoundException)  
    {  
        this.isolatedStore.Add(key, value);  
        valueChanged = true;  
    }  
    catch (ArgumentException)  
    {  
        this.isolatedStore.Add(key, value);  
        valueChanged = true;  
    }  
  
    if (valueChanged)  
    {  
        this.Save();  
    }  
}  
  
private T GetValueOrDefault<T>(string key, T defaultValue)  
{  
    T value;  
  
    try  
{  
        value = (T)this.isolatedStore[key];  
    }  
    catch (KeyNotFoundException)  
    {  
        value = defaultValue;  
    }  
    catch (ArgumentException)  
    {  
        value = defaultValue;  
    }  
}
```

```

    return value;
}

private void Save()
{
    this.isolatedStore.Save();
}
}

```

Survey Data

The application saves the local survey data in isolated storage as a serialized **SurveysList** object. The following code example shows the definition of the **SurveysList** object that uses the model classes **SurveyTemplate** and **SurveyAnswer**. The **SurveyTemplate** class is a model class that defines a survey and includes the question types, question text, and survey metadata. The **SurveyAnswer** class is a model class that defines the responses collected by the surveyor. For more information about these model classes, see the section, “The Model Classes,” earlier in this chapter.

```

C#
public class SurveysList
{
    public SurveysList()
    {
        this.LastSyncDate = string.Empty;
    }

    public List<SurveyTemplate> Templates { get; set; }

    public List<SurveyAnswer> Answers { get; set; }

    public string LastSyncDate { get; set; }
}

```

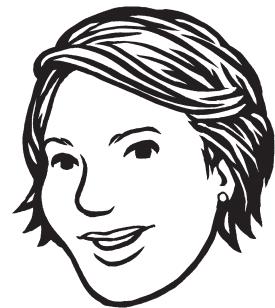
The following code example shows how the **SurveyStore** class (that implements the **ISurveyStore** interface) performs the serialization and deserialization of the **SurveyList** instance to and from isolated storage.

```

C#
private readonly string storeName;

public SurveyStore(string storeName)
{

```



Tailspin uses JSON serialization to reduce CPU usage and storage space requirements.

```
    this.storeName = storeName;
    this.Initialize();
}

public SurveysList AllSurveys { get; set; }

...

public void SaveStore()
{
    lock (this)
    {
        using (var filesystem =
            IsolatedStorageFile.GetUserStoreForApplication())
        {
            using (var fs = new IsolatedStorageFileStream(
                this.storeName, FileMode.Create, filesystem))
            {
                var serializer = new System.Runtime.Serialization
                    .JsonDataContractJsonSerializer(typeof(SurveysList));
                serializer.WriteObject(fs, this.AllSurveys);
            }
        }
    }
}

...

private void Initialize()
{
    lock (this)
    {
        using (var filesystem =
            IsolatedStorageFile.GetUserStoreForApplication())
        {
            if (!filesystem.FileExists(this.storeName))
            {
                this.AllSurveys = new SurveysList
                {
                    Templates = new List<SurveyTemplate>(),
                    Answers = new List<SurveyAnswer>()
                };
            }
            else
            {

```

```
        using (var fs = new IsolatedStorageFileStream(
            this.storeName, FileMode.Open, filesystem))
        {
            var serializer = new System.Runtime.Serialization
                .JsonDataContractJsonSerializer(typeof(SurveysList));
            this.AllSurveys =
                serializer.ReadObject(fs) as SurveysList;
        }
    }
}
```

The following code example shows the interface in the TailSpin.PhoneClient project that defines the operations that the application can perform on the survey store. The **SurveyStore** class implements this interface.

```
C#
using System.Collections.Generic;
using Models;

public interface ISurveyStore
{
    string LastSyncDate { get; set; }
    IEnumerable<SurveyTemplate> GetSurveyTemplates();
    IEnumerable<SurveyAnswer> GetCompleteSurveyAnswers();
    void SaveSurveyTemplates(IEnumerable<SurveyTemplate> surveys);
    void SaveSurveyAnswer(SurveyAnswer answer);
    SurveyAnswer GetCurrentAnswerForTemplate(
        SurveyTemplate template);
    void DeleteSurveyAnswers(
        IEnumerable<SurveyAnswer> surveyAnswers);
    void SaveStore();
}
```

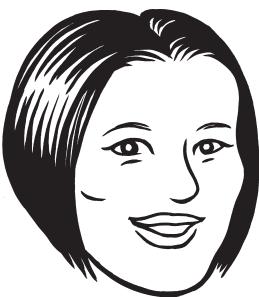
The **SurveyListViewModel** class calls the **GetSurveyTemplates** method to retrieve a list of surveys to display to the user. The **SurveyListViewModel** class creates filtered lists of surveys (new surveys, favorite surveys, and so on) after it has added the surveys to **CollectionViewSource** objects.

The **SurveysSynchronisationService** class uses the **GetCompleteSurveyAnswers**, **SaveSurveyTemplates**, **GetCurrentAnswerForTemplate**, and **DeleteSurveyAnswers** methods to manage the survey data stored on the device.

The **TakeSurveyViewModel** class uses the **GetCurrentAnswerForTemplate** and **SaveSurveyAnswer** methods to retrieve and save survey responses on the device.

Handling Activation and Deactivation

The application must restore its state if the user returns to the application after taking a call or using another application.



The application should store only enough data to be able to restore the application to same state it was in when it was deactivated. Also, remember that an application that is deactivated may never be reactivated, so you must also save any important data to permanent storage.

To comply with Windows Phone 7 certification requirements, your application must complete the deactivation and reactivation processes within 10 seconds.

Windows Phone applications must be able to restore the UI state if the application is reactivated after the user has taken a call or used another application. In this scenario, the operating system terminates the application, but it gives you the opportunity to save any data that you can use to put the application back in the same state if and when the operating system restarts it.

For more information about activation, deactivation, and tombstoning, see the section, “Application Deactivation and Tombstoning,” in Chapter 2, “Designing Applications for Windows Phone 7.”

OVERVIEW OF THE SOLUTION

The Windows Phone 7 platform provides much of the infrastructure that you need to enable your application to restore the state of the UI when the application is re-activated:

- The phone uses activation and deactivation events to notify the application when these life cycle events occur.
- The phone automatically records which screen your application is displaying, along with the current navigation stack, when it deactivates the application; then it rebuilds the navigation stack and redisplays this screen if the phone reactivates the application.
- You can use the **PhoneApplicationService** class from the Windows Phone API to save and restore your application’s state.

It’s the application’s responsibility to determine what state data it needs to save if it is to be able to restore the application to the same state when the phone reactivates it or when it restarts after being terminated by the operating system.

In the Tailspin mobile client application, when the operating system deactivates the application, the application requests that all active view models save their current state. When the operating system reactivates the application, the operating system will redisplay the view that was active when the application was deactivated. The view model locator in the Tailspin mobile application will instantiate the view model for the view, and the view model will reload its saved state and initialize any services so that it is back in the state it was in when it was originally deactivated.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that handles the activation and deactivation events in more detail. As you go through this section, you may want to download the Visual Studio solution for the Windows Phone 7 Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

The developers at Tailspin created a **Tombstoning** class that wraps the **PhoneApplicationService** class. The **PhoneApplicationService** class has methods that save and load any state that the application needs when it's activated or deactivated. The following code example shows the complete **Tombstoning** class.

Note: *There is also a **PhoneApplicationPage** class that allows you to store transient page state as you navigate away from a page and restore the state when you return to the page.*

C#

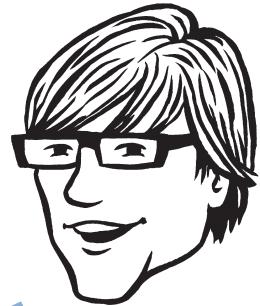
```
using Microsoft.Phone.Shell;

public static class 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);
    }

    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);
        }
    }
}
```



We often refer to the process of deactivating an application as "tombstoning."



The objects that you save in the **State** dictionary must be serializable.

```
    }  
  
    return (T)result;  
}  
}
```

Each view model is responsible for managing its own state when the phone deactivates or reactivates the application. All the view models in the application derive from the **ViewModel** class that attaches an **OnDeactivated** handler method to the **Deactivated** event and an **OnActivated** handler method to the **Activated** event in the **PhoneApplicationService** class. The following code example shows part of the abstract **ViewModel** class.

Note: The developers at Tailspin chose not to use the **Application_Activated** and **Application_Deactivated** event stubs in the `App.xaml.cs` file. Instead, each view model is responsible for persisting and reloading its own state.

```
C#
public abstract class ViewModel : NotificationObject, IDisposable
{
    ...
    protected ViewModel(INavigationService navigationService)
    {
        PhoneApplicationService.Current.Deactivated
            += this.OnDeactivated;
        PhoneApplicationService.Current.Activated
            += this.OnActivated;
        ...
    }
    ...
    public virtual void IsBeingDeactivated()
    {
    }
    public abstract void IsBeingActivated();

    private void OnDeactivated(object s, DeactivatedEventArgs e)
    {
        this.IsBeingDeactivated();
    }
}
```

```
private void OnActivated(object s, ActivatedEventArgs e)
{
    this.IsBeingActivated();
}
```

Each view model can override the **IsBeingDeactivated** and **IsBeingActivated** methods to provide its custom state saving and restoring behavior. The following code example shows how the **SurveyListViewModel** class saves and restores its state.

```
C#
public override void IsBeingDeactivated()
{
    Tombstoning.Save("SelectedTemplate",
        this.SelectedSurveyTemplate);
    Tombstoning.Save("MainPivot", this.SelectedPivotIndex);

    base.IsBeingDeactivated();
}

...
public override sealed void IsBeingActivated()
{
    if (this.SelectedSurveyTemplate == null)
    {
        var tombstoned = Tombstoning
            .Load<SurveyTemplateViewModel>("SelectedTemplate");
        if (tombstoned != null)
        {
            this.SelectedSurveyTemplate =
                new SurveyTemplateViewModel(
                    tombstoned.Template, this.NavigationService);
        }
    }

    this.SelectedPivotIndex = Tombstoning.Load<int>("MainPivot");
}
```

The **IsBeingActivated** method is called by the application's **Activated** event when the application is reactivated; it is also called from the constructor of each view model class. There are two reasons for this apparent duplication of effort. First, the constructors are not always invoked when the application is activated because the application is not always killed when it's deactivated. Second, the constructor logic sometimes requires the state data to complete its initialization logic, and the constructors run before the **Activated** event is raised.



Remember, the activated event is not raised when the application is launched, and the deactivated event is not raised when the application is closed.



If the application has been deactivated, and the user relaunches the application from the **Start** menu, the operating system will delete the saved state, so the application will behave as if it is loading for the first time.

If the application is loading for the first time, there won't be any state data, and the view model can initialize any default values. If the application is reactivating, it can use the saved state data to initialize any values.

Reactivation and the Panorama Control

When your application is reactivated by the operating system, you should restore the UI state, which in the Tailspin application includes displaying the active question if the application was tombstoned while the user was completing a survey. However, the **SelectedIndex** property of the **Panorama** control is read-only. To work around this problem and display the correct question when the application is reactivated, Tailspin changes the **DefaultItem** property of the control instead. This must be done in the code-behind file for the view. The following code example shows the event handlers in the code-behind for the **TakeSurveyView** page.

```
C#
private bool loaded;

private void PanoramaSelectionChanged(object sender,
    System.Windows.Controls.SelectionChangedEventArgs e)
{
    if (this.loaded)
    {
        ((TakeSurveyViewModel)this.DataContext)
            .SelectedPanoramaIndex = this.panorama.SelectedIndex;
    }
}

private void PanoramaLoaded(object sender,
    System.Windows.RoutedEventArgs e)
{
    this.panorama.DefaultItem = this.panorama.Items[
        ((TakeSurveyViewModel)this.DataContext)
            .SelectedPanoramaIndex];
    this.loaded = true;
    ...
}
```

The **SelectedPanoramaIndex** of the **TakeSurveyViewModel** class tracks the currently active question in the **Panorama** control.

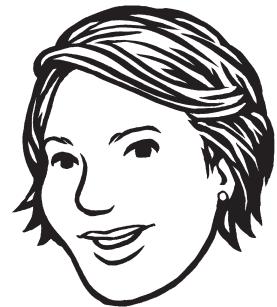
Handling Asynchronous Interactions

Chapter 4, “Building the Mobile Client,” describes how Tailspin implemented commands in the mobile client application. For some commands, Tailspin implements the command asynchronously to avoid locking the UI while a time-consuming operation is running.

For example, on the AppSettingsView page, a user can enable or disable push notifications of new surveys from the Microsoft Push Notification Service (MPNS). This requires the application to send a request to the MPNS that the application must handle asynchronously. The application displays a progress bar on the AppSettingsView page while it handles the asynchronous request.

For more information about MPNS, see Chapter 6, “Connecting with Services.”

Tailspin decided to use the Reactive Extensions (Rx) for .NET to run asynchronous tasks on the phone because it enables them to create compact, easy-to-understand code for complex asynchronous operations.



The Reactive Extensions for .NET are a great way to handle the way that an application interacts with multiple sources of data, such as user input events and web service requests.

USING REACTIVE EXTENSIONS

Rx allows you to write compact, declarative code to manage complex, asynchronous operations. Rx can be described by comparing it to the more familiar concept of enumerable collections. Figure 3 shows two alternative approaches to iterating over a sequence.

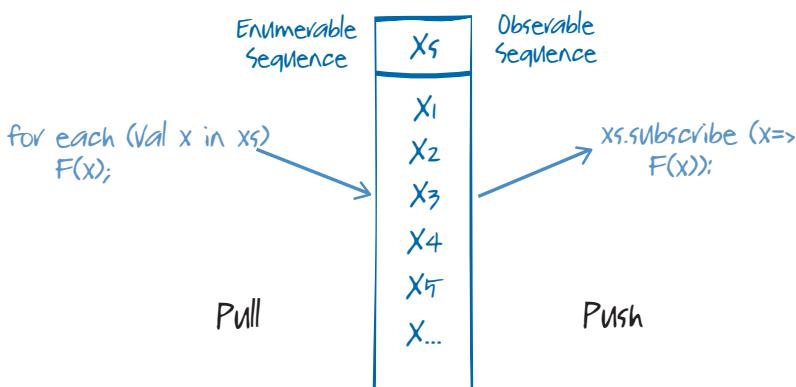


FIGURE 3
Enumerable and observable sequences

To iterate over an enumerable sequence, you can use an iterator to pull each item from the sequence in turn, which is what the C# **foreach** construct does for you. With an observable sequence, you subscribe to an observable object that pushes items from the sequence to you. For example, you can treat the events raised by a

control or the data arriving over the network as an observable sequence. Furthermore, you can use standard LINQ operators to filter the items from the observable sequence, and control which thread you use to process each item as it arrives.

For a description of Rx, see Appendix C, “Leveraging Device Capabilities.”

INSIDE THE IMPLEMENTATION

The application performs an asynchronous request to the MPNS when the user changes his or her preference for push notifications on the AppSettingsView page. The following code example from the AppSettingsView.xaml file shows the definitions of the **ToggleSwitch** control that enables the user to set his or her preference and the progress bar that is active during the asynchronous request.

XAML

```
...
xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;
assembly=Microsoft.Phone.Controls.Toolkit"
...
<toolkit:ToggleSwitch Header="Subscribe to Push Notifications"
    Margin="0,202,0,0"
    IsChecked="{Binding SubscribeToPushNotifications, Mode=TwoWay}" />
...
<ProgressBar Grid.Row="2"
    Height="4"
    Name="progressBar"
    VerticalAlignment="Bottom"
    HorizontalAlignment="Stretch"
    IsIndeterminate="{Binding IsSyncing}" />
```

The **ToggleSwitch** control binds to the **SubscribeToPushNotifications** property of the **AppSettingsViewModel**, and the **ProgressBar** control binds to the **IsSyncing** property of the **AppSettingsViewModel** class.

The following code example from the **AppSettingsViewModel** class shows what happens when the user clicks the **Save** button on the AppSettingsView page. This **Submit** method handles the asynchronous call to the **UpdateReceiveNotifications** method in the **RegistrationServiceClient** class by using Rx. Before it calls the **UpdateReceiveNotifications** method, it first sets the **IsSyncing** property to **true** so that the UI can display an active progress meter. The method handles the asynchronous call in three steps:

1. The **UpdateReceiveNotifications** method in the **RegistrationServiceClient** class returns an observable **TaskSummaryResult** object that contains information about the task.

2. The **Submit** method uses the **ObserveOnDispatcher** method to handle the **TaskSummaryResult** object on the dispatcher thread.
3. The **Subscribe** method specifies how to handle the **TaskSummaryResult** object and starts the execution of the source observable sequence by asking for the next item.

```
C#
public void Submit()
{
    this.IsSyncing = true;
    this.settingsStore.UserName = this.UserName;
    this.settingsStore.Password = this.Password;
    this.settingsStore.LocationServiceAllowed =
        this.LocationServiceAllowed;

    if (this.SubscribeToPushNotifications ==
        this.settingsStore.SubscribeToPushNotifications)
    {
        this.IsSyncing = false;
        this.NavigationService.GoBack();
        return;
    }

    this.registrationServiceClient
        .UpdateReceiveNotifications(
            this.SubscribeToPushNotifications)
        .ObserveOnDispatcher()
        .Subscribe(
            taskSummary =>
            ...
            exception =>
            ...
        );
}
```

The following code example shows the definition of the action that the **Subscribe** method performs when it receives a **TaskSummaryResult** object. If the **TaskSummaryResult** object indicates that the change was successful, it updates the setting in local, isolated storage, sets the **IsSyncing** property to false, and navigates back to the previous view. If the **TaskSummaryResult** object indicates that the change failed, it reports the error to the user.



It's not good practice to catch all exception types; you should rethrow any unexpected exceptions and not simply swallow them.

```
C#
taskSummary =>
{
    if (taskSummary == TaskSummaryResult.Success)
    {
        this.settingsStore.SubscribeToPushNotifications =
            this.SubscribeToPushNotifications;
        this.IsSyncing = false;
        this.NavigationService.GoBack();
    }
    else
    {
        var errorText = TaskCompletedSummaryStrings
            .GetDescriptionForResult(taskSummary);
        this.IsSyncing = false;
        this.submitErrorInteractionRequest.Raise(
            new Notification
            { Title = "Server error", Content = errorText }, n => { });
    }
}
```

The **Subscribe** method can also handle an exception returned from the asynchronous task. The following code example shows how it handles the scenario in the Tailspin mobile client where the asynchronous action throws a **WebException** exception.

```
C#
exception =>
{
    if (exception is WebException)
    {
        var webException = exception as WebException;
        var summary = ExceptionHandling.GetSummaryFromWebException(
            "Update notifications", webException);
        var errorText = TaskCompletedSummaryStrings
            .GetDescriptionForResult(summary.Result);
        this.IsSyncing = false;
        this.submitErrorInteractionRequest.Raise(
            new Notification
            { Title = "Server error", Content = errorText }, n => { });
    }
    else
    {
        throw exception;
    }
}
```

Synchronizing Data between the Phone and the Cloud

The Tailspin mobile client must be able to download new surveys from the Tailspin Surveys service and to upload survey answers to the service. This section describes how Tailspin designed and implemented this functionality. It focuses on the details of the synchronization logic instead of on the technologies the application uses to store data locally and to interact with the cloud. Details of the local storage implementation are described earlier in this chapter, and Chapter 6, “Connecting with Services,” describes how the mobile client application interacts with Tailspin’s cloud services.

There are two separate synchronization tasks that the mobile client must perform:

- The mobile client must download from the cloud service any new surveys that match the user’s subscription criteria.
- The mobile client must send completed survey answers to the cloud service for analysis.

These two tasks are independent of each other; therefore, the mobile client can perform these operations in parallel. Furthermore, for the Tailspin application, the synchronization logic is very simple. At the time of this writing, the Tailspin cloud application does not allow subscribers to modify or delete their survey definitions, so the mobile client only needs to look for new survey definitions. On the client, a surveyor cannot modify survey answers after the survey is complete, so the mobile client can send all of its completed survey answers to the cloud service and then remove them from the mobile client’s local store.

In the Tailspin mobile client application, the user manually initiates the synchronization process by tapping a button. Tailspin considered automating the synchronization process, but the developers at Tailspin decided not to adopt this approach in the first version of the application because it could result in increased resource usage, especially of the battery. A more sophisticated approach for the application would be to automate the synchronization process, but first check the battery level and available network connectivity. Tailspin plans to add this feature in a later version of the application.

Because synchronization can be a time-consuming process, the mobile client should run it asynchronously and notify the user of the outcome when the synchronization completes.

Note: *How often you should run a synchronization process in your application involves some trade-offs. More frequent synchronizations mean that the data on both the client and in the service is more up to date. It can also help to free up valuable storage*

The Tailspin mobile client synchronizes survey definitions and answers between the phone and the Tailspin cloud service.



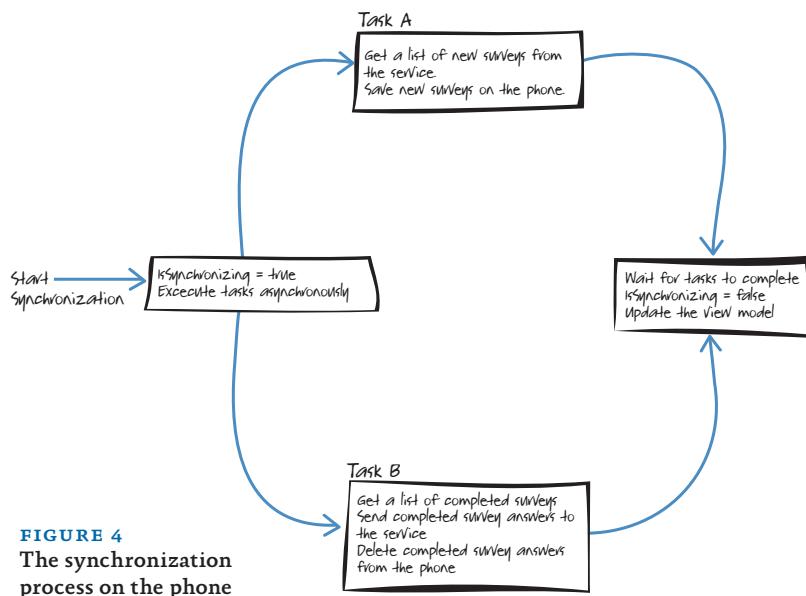
Tailspin’s synchronization logic is relatively simple. A more complex client application may have to deal with modified and deleted data during the synchronization process.

space on the client if the client no longer needs a local copy of the data after it has been transferred to the service. Data stored in the service is also less vulnerable to loss or unauthorized access. However, synchronization is often a resource-intensive process itself, consuming battery power and CPU cycles on the mobile client and using potentially expensive bandwidth to transfer the data. You should design your synchronization logic to transfer as little data as possible.

OVERVIEW OF THE SOLUTION

Tailspin considered using the Microsoft Sync Framework, but they decided to implement the synchronization logic themselves. (For more information about the Microsoft Sync Framework, see Appendix E, "Microsoft Sync Framework and Windows Phone 7.") The reason for this decision was that when Tailspin started developing the mobile client application, the necessary providers were only available as pre-release versions. In addition, the fact that the synchronization requirements for the application are relatively simple reduces the risks associated with developing this functionality themselves. The developers at Tailspin have designed the synchronization service so that they can easily replace the synchronization functionality with an alternative implementation in the future.

Tailspin decided to use the Rx to run the two synchronization tasks asynchronously and in parallel on the phone. Figure 4 summarizes the synchronization process and the tasks that it performs.



The user starts the synchronization process by tapping a button in the UI. A progress bar in the UI is bound to the **IsSynchronizing** property in the view model to provide a visual cue that the synchronization process is in flight. Rx runs the two tasks in parallel, and after both tasks complete, it updates the view model with the new survey data.

In Figure 4, Task A is responsible for downloading a list of new surveys for the user and saving them locally in isolated storage. The service creates the list of new surveys to download based on information sent by the mobile client application. The client sends the date of the last synchronization so that the service can find surveys created since that date, and the service uses the user name sent by the client to filter for surveys that the user is interested in. For more information, see the section, “Filtering Data,” in Chapter 6, “Connecting with Services.”

Task B sends all completed survey answer data to the cloud service, and then it deletes the local copy to free up storage space on the phone.

When both tasks are complete, the application updates the data in the view model, the UI updates based on the bindings between the view and the view model, and the application displays a toast notification if the synchronization was successful or an error pop-up window otherwise. For more information about how the mobile client application handles UI notifications, see the section, “User Interface Notifications,” in Chapter 4, “Building the Mobile Client.”

Limitations of the Current Approach

As discussed earlier, Tailspin’s requirements for the synchronization service are relatively simple because the online Tailspin Surveys service does not allow tenants to modify a survey after they have published it. However, it is possible for tenants to delete surveys in the online application. The current synchronization process in the sample application does not take this into account, so the number of survey definitions stored on the client never decreases. Furthermore, the client will continue to be able to submit answers to surveys that no longer exist in the online service. A real implementation should extend the synchronization logic to accommodate this scenario. One possible solution would be to give every survey an expiration date and make it the mobile client’s responsibility to remove out-of-date surveys. Another solution would be to adopt a full-blown synchronization service, such as the Microsoft Sync Framework.

In addition, the current approach does not address the use case where a user removes a tenant from their list of preferred tenants. The mobile client application will not receive any new surveys from the



It’s important to let the user know that an operation is running asynchronously. When you don’t know how long it will take, use the indeterminate progress bar.



These two limitations highlight the fact that synchronization logic can be complicated, even in relatively simple applications.

deselected tenants, but the application does not remove any previously downloaded surveys from tenants who are no longer on the list. A complete synchronization solution for Tailspin should also address this use case.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements the data synchronization in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

The user can initiate the synchronization process by tapping the **Sync** button on the SurveyListView page. This sends a command to the **SurveyListViewModel** view model, which, in turn, starts the synchronization process. While the synchronization process is running, the application displays an indeterminate progress bar because it has no way of telling how long the synchronization process will take to complete. If the synchronization process is successful, the **SurveyListViewModel** class rebuilds the lists of surveys that are displayed by the SurveyListView page. If the synchronization process fails with a network error or a credentials error, the **SurveyListViewModel** class does not rebuild the lists of surveys that are displayed by the SurveyListView page.

Note: For information about how the user initiates the synchronization process from the user interface, see the section “Commands” in Chapter 4, “Building the Mobile Client.”

The **SurveyListViewModel** class uses Rx to run the synchronization process asynchronously by invoking the **StartSynchronization** method in the **SurveysSynchronizationService** class. When the synchronization process is complete, the **SurveysSynchronizationService** class returns a summary of the synchronization task as a collection of **TaskCompletedSummary** objects. The view model updates the UI by using the **ObserveOnDispatcher** method to run the code on the dispatcher thread. The following code example shows the **StartSync** method in the **SurveyListViewModel** class that interacts with the **SurveysSynchronizationService** class.

```
C#
private readonly
    ISurveysSynchronizationService synchronizationService;
...
public void StartSync()
{
    if (this.IsSynchronizing)
```

```

{
    return;
}

this.IsSynchronizing = true;
this.synchronizationService
    .StartSynchronization()
    .ObserveOnDispatcher()
    .Subscribe(
        taskSummaries => this.SyncCompleted(taskSummaries));
}

```

The **SurveysSynchronizationService** class uses Rx to handle the parallel and asynchronous behavior in the synchronization process. Figure 5 shows the overall structure of the **StartSync** and **StartSynchronization** methods and how they use Rx to run the synchronization tasks in parallel.

The **StartSync** method (**SurveyListViewModel** class) subscribes to:

The **StartSynchronization** method (**SurveysSynchronization** class) that declares:

The **getNewSurveys** task to get a list of new surveys from the Tailspin Surveys service and return:
IObservable<TaskSummary>

The **saveSurveyAnswers** task to save completed surveys to the Tailspin Surveys service and return:
IObservable<TaskSummary>

...and uses the **ForkJoin** method to run these two tasks in parallel and return: **IObservable<TaskSummary[]>**.

...and uses **ObserveOnDispatcher** method to update the UI with the information in the **TaskSummary[]** array.

FIGURE 5
The synchronization methods

The **StartSynchronization** method in the **SurveysSynchronizationService** class uses the **Observable.ForkJoin** method to define the set of parallel operations that make up the synchronization process. The **ForkJoin** method blocks until all the parallel operations are complete.

The following code example shows the **SurveysSynchronizationService** class and includes an outline of the **StartSynchronization** method that the **SurveyListViewModel** class calls. This code implements the set of tasks shown in Figure 5.



Using Rx can make code that handles asynchronous operations simpler to understand and more compact.

```
C#
...
using Microsoft.Phone.Reactive;
...

public class SurveysSynchronizationService :
    ISurveysSynchronizationService
{
    ...
    public IObservable<TaskCompletedSummary[]>
        StartSynchronization()
    {
        var surveyStore = this.surveyStoreLocator.GetStore();

        var getNewSurveys =
            this.surveysServiceClientFactory()
                .GetNewSurveys...
        // Get the list of new surveys from Tailspin Surveys.
        ...
        saveSurveyAnswers = this.surveysServiceClientFactory()
            .SaveSurveyAnswers...
        // Save completed surveys to the Tailspin Surveys service.
        ...
        return Observable.ForkJoin(getNewSurveys, saveSurveyAnswers);
    }
}
```

Note: The application uses the `Funq` dependency injection container to create the **SurveysSynchronizationService** instance. For more information, see the **ViewModelLocator** class.

The **StartSynchronization** method uses Rx to run the two synchronization tasks asynchronously and in parallel. When each task completes, it returns a summary of what happened in a **TaskCompletedSummary** object, and when both tasks are complete, the method returns an array of **TaskCompletedSummary** objects from the **ForkJoin** method.

The `getNewSurveys` Task

The **getNewSurveys** task retrieves a list of new surveys from the Tailspin Surveys service and saves them in isolated storage. When the task is complete, it creates a **TaskCompletedSummary** object with information about the task. The following code example shows the complete definition of this task that breaks down to the following subtasks:

- The **GetNewSurveys** method returns an observable list of **SurveyTemplate** objects from the Tailspin Surveys service.
- The **Select** method saves these surveys to isolated storage on the phone, updates the last synchronization date, and then returns an observable **TaskCompletedSummary** object.
- The **Catch** method traps any errors and returns a **TaskCompletedSummary** object with details of the error.

C#

```
var getNewSurveys =
    this.surveysServiceClientFactory()
    .GetNewSurveys(surveyStore.LastSyncDate)
    .Select(
        surveys =>
    {
        surveyStore.SaveSurveyTemplates(surveys);

        if (surveys.Count() > 0)
        {
            surveyStore.LastSyncDate =
                surveys.Max(s => s.CreatedOn).ToString("s");
        }
    }

    return new TaskCompletedSummary
    {
        Task = GetSurveysTask,
        Result = TaskSummaryResult.Success,
        Context = surveys.Count().ToString()
    };
})
.Catch(
    (Exception exception) =>
{
    if (exception is WebException)
    {
        var webException = exception as WebException;
        var summary = ExceptionHandling.GetSummaryFromWebException(
            GetSurveysTask, webException);
        return Observable.Return(summary);
    }

    if (exception is UnauthorizedAccessException)
    {
        return Observable.Return(new TaskCompletedSummary
```

```

    {
        Task = GetSurveysTask,
        Result = TaskSummaryResult.AccessDenied
    });
}

throw exception;
});

```

The **saveSurveyAnswersTask**

The **saveSurveyAnswersTask** saves completed survey answers to the Tailspin Surveys service and then removes them from isolated storage to free up storage space on the phone. It returns an observable **TaskCompletedSummary** object with information about the task. The following code example shows the complete definition of this task that breaks down to the following subtasks:

1. The **GetCompleteSurveyAnswers** method gets a list of completed surveys from isolated storage.
2. The first call to **Observable.Return** creates an observable **TaskCompletedSummary** object so that the task returns at least one observable object (otherwise, the **ForkJoin** method may never complete). This also provides a default value to return if there are no survey answers to send to the Tailspin Surveys service.
3. The **SaveSurveyAnswers** method saves the completed surveys to the Tailspin Surveys service and returns an observable list of saved **SurveyModel** objects.
4. The **Select** method deletes all the saved surveys from isolated storage and then returns an observable **TaskCompletedSummary** object.
5. The **Catch** method traps any errors and returns a **TaskCompletedSummary** object with details of the error.

C#

```

var surveyAnswers = surveyStore.GetCompleteSurveyAnswers();
var saveSurveyAnswers =
    Observable.Return(new TaskCompletedSummary
    {
        Task = SaveSurveyAnswersTask,
        Result = TaskSummaryResult.Success,
        Context = 0.ToString()
    });
if (surveyAnswers.Count() > 0)

```

```
{  
    saveSurveyAnswers =  
        this.surveysServiceClientFactory()  
            .SaveSurveyAnswers(surveyAnswers)  
            .Select(  
                unit =>  
                {  
                    var sentAnswersCount = surveyAnswers.Count();  
                    surveyStore.DeleteSurveyAnswers(surveyAnswers);  
                    return new TaskCompletedSummary  
                    {  
                        Task = SaveSurveyAnswersTask,  
                        Result = TaskSummaryResult.Success,  
                        Context = sentAnswersCount.ToString()  
                    };  
                })  
                .Catch(  
                    (Exception exception) =>  
                    {  
                        ...  
                    });  
            }  
}
```

Using Location Services on the Phone

Tailspin would like to capture the user's location when they are answering a survey and include this location information as part of the survey data that's sent to the Tailspin service when the synchronization process runs. Tenants can use the location information when they analyze the survey results.

OVERVIEW OF THE SOLUTION

The Windows Phone 7 API includes a Location Service that wraps the available hardware on the phone and enables your application to easily access location data. However, there is a trade-off between the accuracy of the data you can obtain from the Location Service and your application's power consumption. Tailspin does not require highly accurate location data for their surveys, so they have optimized the Surveys mobile client application to minimize power consumption.

The developers at Tailspin also decided that it is more important to save the survey data quickly and reliably, and not to wait if the location data is not currently available. It can take up to 120 seconds to get location data back from the Location Service: therefore, they use

Higher accuracy in location data leads to higher power consumption.



The Tailspin Surveys website displays survey location data using a Bing™ maps control. For more information, see the SurveyLocation.ascx file in the TailSpin.Web project. There is also a Bing maps control available for the phone; for information about it, see Appendix C, "Leveraging Device Capabilities."

You could also create a dummy implementation of the **ILocationService** interface that returns a fixed location to use in the emulator.



It's down to the phone to determine the optimal way to obtain the phone's location using the available data from the GPS receiver, cellular triangulation, or Wi-Fi data.

the latest available location data to save along with the survey answers instead of waiting for new data. Furthermore, the application only asks the Location Service for location data when the Surveys application needs it to save with a survey, although in some applications, you might consider caching the available location data at fixed intervals.

Note: *The sample application asks the user's permission to collect location data in the AppSettingsView page. In your own application, you must obtain the user's consent before collecting and using location data, typically on an initial settings screen when the application first runs. You should also make sure that your application can continue to function if the user disables the Location Service or doesn't give their consent for your application to use the phone's location data.*

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that acquires location data from the phone in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

The **ILocationService** interface defines a single method, **TryToGetCurrentLocation**, which returns a **GeoCoordinate** object that holds the location data. The **LocationService** class implements this interface.

The **LocationService** class uses the **GeoCoordinateWatcher** class from the Windows Phone 7 API to retrieve the current location from the phone. Tailspin does not require highly accurate location data in the Surveys application, so the **LocationService** class initializes the **GeoCoordinateWatcher** class using the default accuracy. This gives the phone the opportunity to reduce its power consumption and to return location data more quickly.

The **GeoCoordinateWatcher** class uses the **StatusChanged** event to notify the application when it has an available fix on the phone's location.

The following code example shows how the application initializes the **GeoCoordinateWatcher** object, handles the **StatusChanged** event, collects the phone's location, and then disposes of the **GeoCoordinateWatcher** object. To minimize your application's power consumption, you should dispose of the **GeoCoordinateWatcher** instance as soon as you've got the location data you need.

```
C#  
private readonly TimeSpan maximumAge = TimeSpan.FromMinutes(15);  
private GeoCoordinate lastCoordinate = GeoCoordinate.Unknown;  
private DateTime lastCoordinateTime;  
private GeoCoordinateWatcher watcher;  
...  
private void InitializeWatcher()  
{  
    this.watcher =  
        new GeoCoordinateWatcher(GeoPositionAccuracy.Default);  
    this.watcher.Start(true);  
    if (this.watcher.Status == GeoPositionStatus.Initializing ||  
        this.watcher.Status == GeoPositionStatus.NoData)  
    {  
        this.watcher.StatusChanged += this.WatcherStatusChanged;  
    }  
    else  
    {  
        this.GetNewLocation();  
    }  
}  
  
private void WatcherStatusChanged(  
    object sender, GeoPositionStatusChangedEventArgs e)  
{  
    this.GetNewLocation();  
}  
  
private void GetNewLocation()  
{  
    if (this.watcher != null)  
    {  
        var newCoordinate = this.watcher.Position.Location;  
        if (newCoordinate != GeoCoordinate.Unknown)  
        {  
            this.lastCoordinate = this.watcher.Position.Location;  
            this.lastCoordinateTime = DateTime.Now;  
        }  
  
        this.DisposeWatcher();  
    }  
}  
  
private void DisposeWatcher()  
{
```

```
if (this.watcher != null)
{
    var oldWatcher = this.watcher;
    this.watcher = null;
    oldWatcher.Stop();
    oldWatcher.Dispose();
}
```

To minimize how often the application requests location data from the phone, the **TryToGetCurrentLocation** method only asks the phone for new location data if at least 15 minutes have passed since the previous request.

The following code example shows the **TryToGetCurrentLocation** method that returns location data if the user has given consent for Tailspin to use location data obtained from the phone.

```
C#
public GeoCoordinate TryToGetCurrentLocation()
{
    if (!this.settingsStore.LocationServiceAllowed)
    {
        return GeoCoordinate.Unknown;
    }

    if (this.watcher == null)
    {
        if (this.maximumAge <
            (DateTime.Now - this.lastCoordinateTime) ||
            this.lastCoordinate == GeoCoordinate.Unknown)
        {
            this.InitializeWatcher();
        }
    }

    return this.lastCoordinate;
}
```

For more information about using the Location Service on the Windows Phone 7 platform, see Appendix C, “Leveraging Device Capabilities.”

Acquiring Image and Audio Data on the Phone

The Tailspin Surveys mobile client application allows users to capture images from the device's camera and record audio from the device's microphone as answers to survey questions. The application saves the captured data as part of the survey answer, and this data is sent to the Tailspin Surveys web service when the user synchronizes the mobile client application.

The mobile client application can capture image and audio data.

OVERVIEW OF THE SOLUTION

The techniques you use to capture audio and image data are different. To capture image data from the camera, you use a "chooser," and to capture audio data from the microphone, you must use interop with the XNA® framework on the phone.

For more information about these techniques, see Appendix C, "Leveraging Device Capabilities."

Capturing Image Data

The chooser for capturing image data is the **CameraCaptureTask**. When you use a chooser, the operating system deactivates your application and runs the chooser as a new process. When the chooser has completed its task, the operating system reactivates your application and delivers any data to your application using a callback method. The developers at Tailspin chose to implement this using the **Observable** class from the Rx library on the phone. The Tailspin Surveys mobile client application saves the captured picture to isolated storage along with the other survey answers and also displays the picture in the view.

Recording Audio Data

To access the microphone on the Windows Phone 7 device, you have to use XNA; it's not possible to access it directly from Silverlight. To interoperate with XNA, you must use an XNA asynchronous event dispatcher to connect to the XNA events from Silverlight. Your application can then handle the microphone events that deliver the raw audio data to your application. Your application must then convert or encode the audio data to a valid sound format before saving the audio data in isolated storage.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that captures image data and records audio in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).



You can use the EXIF data in the picture to determine the correct orientation for displaying the picture.

Using a Chooser to Capture Image Data

The following code example shows how the **CameraCaptureCommand** delegate command is defined in the constructor for **PictureQuestionViewModel** class. This command uses the **Capturing** property to check whether the application is already in the process of capturing a picture and to control whether the command can be invoked from the UI. The method displays a picture if there is already one saved for this question.

The method then creates a **CameraCaptureTask**, which will launch the chooser for taking the photograph and return the captured picture.

C#

```
private readonly CameraCaptureTask task;
public DelegateCommand CameraCaptureCommand { get; set; }

public PictureQuestionViewModel(QuestionAnswer questionAnswer)
    : base(questionAnswer)
{
    this.CameraCaptureCommand =
        new DelegateCommand(this.CapturePicture,
            () => !this.Capturing);
    if (questionAnswer.Value != null)
    {
        this.CreatePictureBitmap(questionAnswer.Value);
    }
    this.task = new CameraCaptureTask();
    ...
}
...
private void CapturePicture()
{
    if (!this.Capturing)
    {
        this.task.Show();
        this.Capturing = true;
        this.CameraCaptureCommand.RaiseCanExecuteChanged();
    }
}
...
public bool Capturing
{
    get { return this.capturing; }
    set
```

```
    {
        if (this.capturing != value)
        {
            this.capturing = value;
            this.RaisePropertyChanged(() => this.Capturing);
        }
    }
}
```

The following code examples show how the constructor uses the **Observable.FromEvent** method to specify how to handle the **Completed** event raised by the **CameraCaptureTask** chooser object when the user has finished with the chooser. The first example shows how the application saves the picture using a thread from the thread pool, and uses the dispatcher thread to display the picture and re-enable the **CameraCaptureCommand** command if the user has taken a photograph.

```
C#
Observable.FromEvent<PhotoResult>(
    h => this.task.Completed += h,
    h => this.task.Completed -= h)
    .Where(e => e.EventArgs.ChosenPhoto != null)
    .ObserveOn(Scheduler.ThreadPool)
    .Select(a => CreatePictureFile(a.EventArgs.ChosenPhoto))
    .ObserveOn(Scheduler.Dispatcher)
    .Subscribe(p =>
{
    this.Answer.Value = p;
    this.CreatePictureBitmap(p);
    this.Capturing = false;
    this.RaisePropertyChanged(string.Empty);
    this.CameraCaptureCommand.RaiseCanExecuteChanged();
});
```

The second example shows how the **CameraCaptureCommand** command is re-enabled if the user didn't take a picture.

```
C#  
Observable.FromEvent<PhotoResult>(  
    h => this.task.Completed += h,  
    h => this.task.Completed -= h)  
.Where(e => e.EventArgs.ChosenPhoto == null)  
.ObserveOn(Scheduler.Dispatcher)  
.Subscribe(p =>
```



Using Rx means we can filter on just the events and event parameter values that we're interested in by using simple LINQ expressions, and we can make sure we use the right thread to update the UI—all in compact and easy-to-read code.

```
{
    this.Capturing = false;
    this.CameraCaptureCommand.RaiseCanExecuteChanged();
});
```

Using XNA Interop to Record Audio

Before the Tailspin mobile client application can handle events raised by XNA objects, such as a **Microphone** object, it must create an XNA asynchronous dispatcher service. The following code example from the App.xaml.cs file shows how this is done.

```
C#
public App()
{
    ...
    this.ApplicationLifetimeObjects.Add(
        new XnaAsyncDispatcher(TimeSpan.FromMilliseconds(50)));
}
```

For more information about the **XnaAsyncDispatcher** class, see Appendix B, “Silverlight and XNA in Windows Phone 7.”

The VoiceQuestionView.xaml file defines two buttons, one toggles recording on and off, and the other plays back any saved audio. The recording toggle button is bound to the **DefaultAction Command** command in the view model, and the play button is bound to the **PlayCommand** command in the view-model.

The **DefaultAction** command uses the **StartRecording** and **StopRecording** methods in the **VoiceQuestionViewModel** class to start and stop audio recording. The following code example shows the **StartRecording** method.

```
C#
private void StartRecording()
{
    var mic = Microphone.Default;
    if (mic.State == MicrophoneState.Started)
    {
        mic.Stop();
    }

    this.formatter = new WaveFormatter(
        this.wavFileName, (ushort)mic.SampleRate, 16, 1);

    this.observableMic = Observable.FromEvent<EventArgs>(
        new EventHandler<EventArgs>(mic_Recording),
        new EventHandler<EventArgs>(mic_Stopped));
}
```

```

h => mic.BufferReady += h, h => mic.BufferReady -= h)
.Subscribe(p =>
{
    var content =
        new byte[mic.GetSampleSizeInBytes(mic.BufferDuration)];
    mic.GetData(content);
    if (this.formatter != null)
    {
        this.formatter.WriteDataChunk(content);
    }
});
mic.Start();
}

```

This method gets a reference to the default microphone on the device and creates a **WaveFormatter** instance to convert the raw audio data to the WAV format.

Note: You can find the **Microphone** class in the **Microsoft.Xna.Framework.Audio** namespace.

The method uses the **Observable.FromEvent** method to subscribe to the microphone's **BufferReady** event, and whenever the event is raised, the application uses the **WaveFormatter** instance to write the audio data to isolated storage. Finally, the method starts the microphone.

The following code example shows the **StopRecording** method that disposes of the **Microphone** and **WaveFormatter** instances and attaches the name of the saved audio file to the question.

```

C#
private void StopRecording()
{
    Microphone.Default.Stop();

    this.observableMic.Dispose();
    this.formatter.Dispose();
    this.formatter = null;
    this.Answer.Value = this.wavFileName;
}

```

The play button in the **VoiceQuestionView** view plays the recorded audio by using the **SoundEffect** class from the **Microsoft.Xna.Framework.Audio** namespace. The following code example shows the **Play** method from the **VoiceQuestionViewModel** class that loads audio data from isolated storage and plays it back.



The Windows Phone 7 API does not include any methods to convert audio formats. You can find the **WaveFormatter** class in the **TailsSpin.Phone Client.Infrastructure** namespace.

```
C#
private void Play()
{
    this.isPlaying = true;
    using (var fileSystem =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (var dat = fileSystem.OpenFile(
            this.wavFileName, FileMode.Open, FileAccess.Read))
        {
            try
            {
                using (var effect = SoundEffect.FromStream(dat))
                {
                    var instance = effect.CreateInstance();
                    instance.Play();

                    while (instance.State == SoundState.Playing)
                    {
                        System.Threading.Thread.Sleep(100);
                    }
                }
            }
            catch (ArgumentException)
            {
            }
        }
    }
    this.isPlaying = false;
}
```

Logging Errors and Diagnostic Information on the Phone

The sample application does not log any diagnostic information or details of error conditions from the Tailspin mobile client application. Tailspin plans to add this functionality to a future version of the mobile application after they evaluate a number of tradeoffs.

For example, Tailspin must decide whether to keep a running log on the phone or simply report errors to a service as and when they occur. Keeping a log on the phone will use storage, so Tailspin would have to implement a mechanism to manage the amount of isolated storage used for log data, perhaps by keeping rolling logs of recent

activity or implementing a purge policy. Sending error data as it occurs minimizes the storage requirements, but it makes it harder to access data about the state of the application before the error occurred. Tailspin would also need to develop a robust way to send the error information to a service, while transferring log files could take place during the application's standard synchronization process. Collecting logs also makes it easier to correlate activities on the phone with activities in the various Tailspin Surveys services.

Tailspin must also consider the user experience. A privacy policy would require a user to opt into collecting diagnostic information, and Tailspin might want to include options that would enable users to set the level of logging detail to collect.

Conclusion

This chapter described how the developers at Tailspin implemented the model elements from the MVVM pattern in the Tailspin Surveys mobile client, and how the application leverages services offered by the Windows Phone 7 platform, such as isolated storage and location services.

The developers at Tailspin also created some services themselves; for example, they created the synchronization service that runs a set of asynchronous parallel tasks on the phone to manage the data used by the application. This synchronization service needs to access data held remotely by the Tailspin Surveys application that runs on Windows Azure™. The next chapter will describe how the mobile client application can access remote services like the one that provides access to the data held in the Windows Azure application.

Questions

1. Why is it difficult to encrypt data in isolated storage?
 - a. The Windows Phone 7 platform does not include any useful encryption APIs.
 - b. There is no way to securely encrypt data in isolated storage without requiring the user to enter a password or PIN at least once every time he or she uses the application.
 - c. Any encryption mechanism will increase CPU usage and reduce battery life.
 - d. You cannot control the format of the data in isolated storage.

2. What happens when your application is reactivated?
 - a. You return to the first screen in your application.
 - b. The operating system makes sure that the screen is displayed as it was when the application was deactivated.
 - c. The operating system recreates the navigation stack within your application.
 - d. The **Launching** event is raised.
3. What data should you save when you handle the deactivation request?
 - a. State data required to rebuild the state of the last screen that was active before the application was deactivated
 - b. State data required to rebuild the state of previous screens that user had navigated through before the application was deactivated
 - c. Data that is normally persisted to isolated storage by the application at some point
 - d. The currently active screen
4. Why does Tailspin use the Reactive Extensions (Rx) for .NET?
 - a. To handle notifications from the Microsoft Push Notification Service
 - b. To handle UI events
 - c. To manage asynchronous tasks
 - d. To make the code that implements the asynchronous and parallel operations more compact and easier to understand
5. What factors should you consider when you use location services on the phone?
 - a. What level of accuracy your application requires for its geo-location data
 - b. Whether the device has a built-in Global Positioning System (GPS)
 - c. How quickly you need to obtain the current location
 - d. Whether the user has consented to allowing your application to use the phone's GPS data

6. What special component do you need to handle XNA interop when you're recording audio?
 - a. None
 - b. You must state that your application will use XNA interop in the WMAppManifest.xml file
 - c. An XNA Asynchronous Dispatcher class
 - d. An **Observable** instance

More Information

For more information about isolated storage, see “Isolated Storage for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff626522\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff626522(VS.92).aspx)

For more information about handling deactivation, reactivation, and tombstoning, see “Execution Model for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff769557\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769557(VS.92).aspx)

For a series of great posts that discuss the Windows Phone 7 execution model, see The Windows Phone Developer Blog: http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/07/15/understanding-the-windows-phone-application-execution-model-tombstoning-launcher-and-choosers-and-few-more-things-that-are-on-the-way-part-1.aspx

For more information about launchers and choosers, see “Launchers and Choosers for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff769556\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769556(VS.92).aspx)

For more information about Reactive Extensions, see “Reactive Extensions for .NET Overview for Windows Phone” and “DevLabs: Reactive Extensions for .NET (Rx)” on MSDN: [http://msdn.microsoft.com/en-us/library/ff431792\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431792(VS.92).aspx). <http://msdn.microsoft.com/en-us/devlabs/ee794896.aspx>

For more information about location services, see “Location for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff431803\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431803(VS.92).aspx)

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”



6

Connecting with Services

This chapter describes the different ways that the Tailspin Surveys mobile client interacts with external services, both custom services created by Tailspin, and services offered by third-party companies. Connecting to external services from a mobile client introduced a set of challenges for the development team at Tailspin to meet in the design and implementation of the mobile client, and in the services hosted in Windows Azure™ technology platform. The mobile client application must do the following:

- It must operate reliably with variable network connectivity.
 - It must minimize the use of network bandwidth (which may be costly).
 - It must minimize its impact on the phone's battery life.
- The online service components must do the following:
- They must offer an appropriate level of security.
 - They must be easy to develop client applications against.
 - They must support a range of client platforms.

The key areas of functionality in the Tailspin Surveys application that this chapter describes include hosting web content for Windows® Phone 7 devices, authenticating with a web service from an application on the phone, pushing notifications to Windows Phone 7 devices, and transferring data between a Windows Phone 7 device and a web service.

Installing the Mobile Client Application

This section describes how Tailspin arranges for users to install the mobile client application on their Windows Phone 7 devices. Users can only install applications on their devices from the Windows Marketplace, so Tailspin must first make sure that the application is available there. For more information about distributing and selling your applications, see Chapter 7, "Interacting with Windows Marketplace."

Users can only install applications onto their phones through Windows Marketplace.

OVERVIEW OF THE SOLUTION

To make it easy for users to find, download, and install the mobile client application, Tailspin wanted to provide a link to the mobile client installer from the public Tailspin website with which users may already be familiar. Tailspin provides a Windows Phone 7-friendly page at the same address as the public Tailspin website. Accessing the site with Microsoft® Internet Explorer® from a desktop device shows a list of available surveys; alternatively, accessing the site with Internet Explorer from a Windows Phone 7 device shows a link to the installer for the mobile client application.

The developers at Tailspin used a Model-View-Controller (MVC) view engine to display a different page based on the type of device making the request.

For more information about MVC, see “ASP.NET MVC 2” on MSDN® (<http://msdn.microsoft.com/en-us/library/dd394709.aspx>).

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements the Windows Phone 7 web page in more detail. As you go through this section, you may want to download the Microsoft® Visual Studio® development system solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

To render different pages at the same address based on the type of device, the Tailspin web application uses the **WebFormViewEngine** class in the **MVC** namespace. The application creates a new view engine of type **MobileCapableWebFormViewEngine** in the Global.asax.cs file. The following code example shows the **MobileCapableWebFormViewEngine** class in the TailSpin.Web.Survey.Public project.

```
C#
namespace TailSpin.Web.Survey.Public.Extensions
{
    using System;
    using System.Web.Mvc;

    public class MobileCapableWebFormViewEngine : WebFormViewEngine
    {
        public override ViewEngineResult FindView(
            ControllerContext controllerContext, string viewName,
            string masterName, bool useCache)
        {
            ViewEngineResult result = null;
```

```
if (this.UserAgentIs(controllerContext, "IEMobile/7"))
{
    result = new ViewEngineResult(new WebFormView(
        "~/Views/MobileIE7/Index.aspx", string.Empty), this);
}

if (result == null || result.View == null)
{
    result = base.FindView(controllerContext, viewName,
        masterName, useCache);
}

return result;
}

public bool UserAgentIs(ControllerContext controllerContext,
    string userAgentToTest)
{
    return controllerContext.HttpContext.Request
        .UserAgent.IndexOf(userAgentToTest,
        StringComparison.OrdinalIgnoreCase) > 0;
}
}
```

The **FindView** method checks the user agent to determine the browser type, and then it returns an appropriate **ViewEngineResult** instance.

For more information about creating websites for mobile devices, see the post, “Mix: Mobile Web Sites with ASP.NET MVC and the Mobile Browser Definition File,” on the blog, Scott Hanselman’s ComputerZen.com (<http://www.hanselman.com/blog/MixMobileWebSitesWithASPMVCAndTheMobileBrowserDefinitionFile.aspx>).

Authenticating with the Surveys Service

The Surveys service running in Windows Azure needs to know the identity of the user who is using the mobile client application on the Windows Phone 7 device for two reasons. First, when the mobile client requests a list of surveys, the service determines the contents of the list based on the preferences stored in the user’s profile. Second, when the mobile client submits a set of survey answers, the Surveys service needs to record who submitted the survey in order to be able to calculate any rewards due to the user.

The Windows Azure-based Surveys service needs to identify the user using the mobile client application.

GOALS AND REQUIREMENTS

Tailspin wants to be able to change the way it authenticates users without requiring major changes to the Surveys application.

Tailspin wants to externalize as much as possible of the authentication and authorization functionality from the main Surveys application. This will give Tailspin the flexibility to make changes to the way they handle authentication and authorization in the future without affecting the Surveys application itself. For example, Tailspin may want to enable users to identify themselves by using a Windows Live® ID.

It's also important to ensure that the mechanism the mobile client uses to authenticate is easy to implement on the Windows Phone 7 platform and any other mobile platforms that Tailspin may support in the future.

OVERVIEW OF THE SOLUTION

Figure 1 shows a high-level view of the approach adopted by Tailspin.

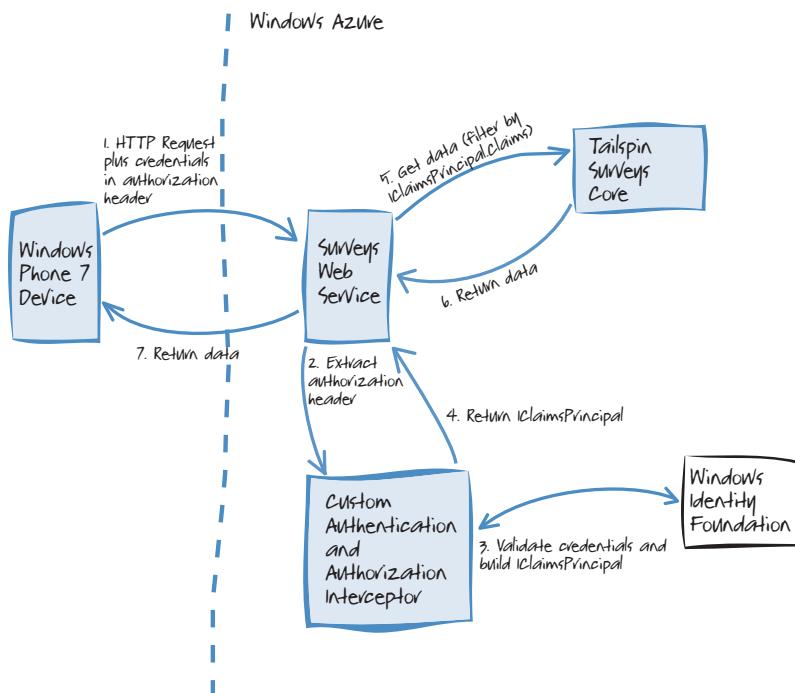


FIGURE 1
Authentication and authorization for the Surveys web services

The approach that Tailspin adopted assumes that the Windows Phone 7 client application can send credentials to the Surveys web service in an HTTP header. The credentials could be a user name and password or a token. Tailspin could easily change the type of credentials in a future version of the application.

Note: In the sample application, the phone sends a user name and password to demonstrate this approach. The mobile client does not perform any validation on the credentials that the user enters on the AppSettingsView page or encrypt the credentials before it saves them in isolated storage. In a real application, you may decide to enforce a password policy that requires strong passwords and regular password renewals.

In the Surveys web service, the custom authentication and authorization interceptor extracts the header that contains the user's credentials and identifies an authentication module to perform the authentication. It's possible that different client platforms use different authentication schemes, so the interceptor must be able to identify from the HTTP headers which type of authentication is being used, and then pass the credentials to the correct custom authentication module.

Note: The sample application uses a mock authentication module that simply checks for one of several hard-coded user names; it does not verify the password. If you use password-based credentials in your application, you should send a hashed version of the password over the network to compare with a hashed version stored in Windows Azure storage in your authentication module.

After the custom authentication module validates the credentials, it uses Windows Identity Foundation (WIF) to construct an **IClaimsPrincipal** object that contains the user's identity. In the future, this **IClaimsPrincipal** object might contain additional claims that the Surveys application could use to perform any authorization that it requires.

The surveys web service includes the **IClaimsPrincipal.CClaims.Name** value when it invokes any methods in the Tailspin Surveys core application. The Tailspin Surveys core application returns data for the user. In the future, the web service could also perform any necessary authorization before it invokes a method in the core application.

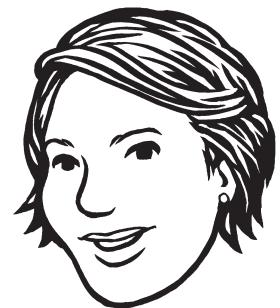
A Future Claims-Based Approach

In the future, Tailspin is considering replacing the simple user name and password authentication scheme with a claims-based approach. One option is to use Simple Web Token (SWT) and the Open Authentication (OAuth) 2.0 protocol. This approach offers the following benefits:

- The authentication process is managed externally from the Tailspin Surveys application.
- The authentication process uses established standards.
- The Surveys application can use a claims-based approach to handle any future authorization requirements.



Tailspin must be able to handle authentication from other mobile platforms in the future, so choosing a flexible, standards-based approach to authentication is crucial.

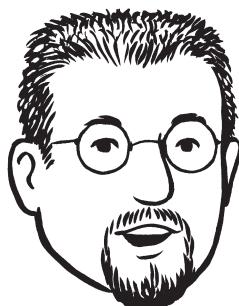


To change the authentication method for the mobile client application, Tailspin must make two changes to the application:

1. Modify the mobile client to send the credentials in a custom HTTP header.
2. Add a new custom authentication module to validate the credentials and create an **IClaimsPrincipal** object.



When Tailspin were developing the mobile client, new versions of the OAuth and SWT standards were anticipated, so they decided to wait for these new releases.



You should consider keeping the Client ID and Client Secret secure on the phone. If someone discovers them, they could create an application that impersonates the Tailspin mobile client application.

Figure 2 illustrates this approach, showing the external token issuer.

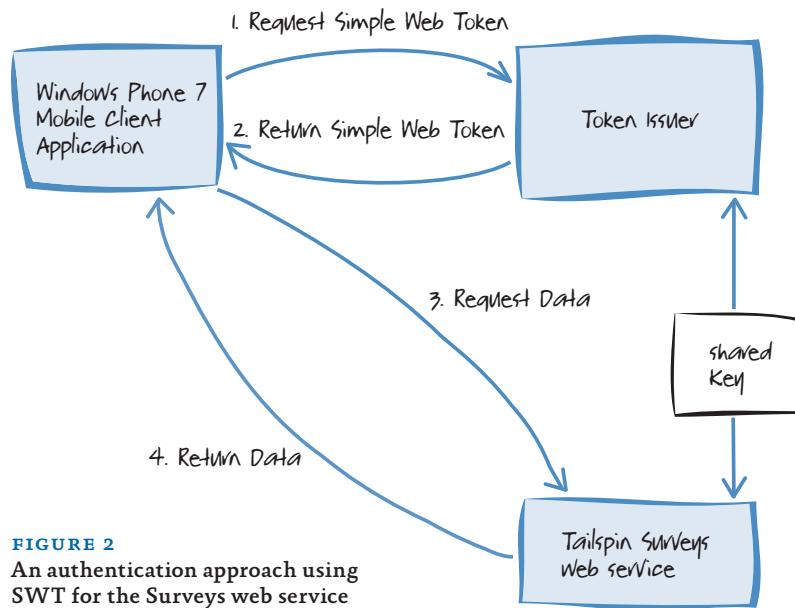


FIGURE 2
An authentication approach using SWT for the Surveys web service

In this scenario, before the mobile client application invokes a Surveys web service, it must obtain an SWT. It does this by sending a request to a token issuer that can issue SWTs; for example, Windows Azure Access Control Services (ACS). The request includes the items of information described in the following table.

Field	Description
Client ID	The client ID is an identifier for the consumer application, which is the Surveys service in this case.
Client Secret	A piece of information that proves that it is your application.
User Name	The user name of the person who wants to authenticate with the Surveys service. The application will prompt the user to enter this name in the user interface (UI).
Password	The user's password. The application will prompt the user to enter this password in the UI.

The client ID and client secret enable the issuer to determine which application is requesting an SWT. The issuer uses the user name and password to authenticate the user.

The token issuer then constructs an SWT containing the user's identity and any other claims that the consumer application (Tailspin Surveys) might require. The issuer also attaches a hash value generated using a secret key shared with the Tailspin Surveys service.

When the client application requests data from the Surveys service, it attaches the SWT to the request in the request's authorization header.

When the Surveys service receives the request, a custom authentication module extracts the SWT from the authorization header, validates the SWT, and then extracts the claims from the SWT. The Surveys service can then use the claims with its authorization rules to determine what data, if any, it should return to the user.

The validation of the SWT in the custom authentication module performs the following steps.

- It verifies the hash of the SWT by using the shared secret key. This enables the Surveys service to verify the data integrity and the authenticity of the message.
- It verifies that the SWT has not expired. The token issuer sets the expiration time when it creates the SWT.
- It checks that the issuer that created the SWT is an issuer that the service is configured to trust.
- It checks that the client application that is making the request is a client that the service is configured to trust.



The OAuth protocol uses a shared key to generate a hash of the SWT. This shared secret key must be known by the issuer and the Surveys service.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements the authentication process in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

The **CustomServiceHostFactory** class in the TailSpin.Services.Surveys project initializes the Surveys service. The following code example shows how this factory class creates the authorization manager.

```
C#
public class CustomServiceHostFactory : WebServiceHostFactory
{
    private readonly IUnityContainer container;

    public CustomServiceHostFactory(IUnityContainer container)
    {
        this.container = container;
    }

    protected override ServiceHost CreateServiceHost(
        Type serviceType, Uri[] baseAddresses)
    {
```

```

        var host = new CustomServiceHost(
            serviceType, baseAddresses, this.container);

        host.Authorization.ServiceAuthorizationManager =
            new SimulatedWebServiceAuthorizationManager();
        host.Authorization.PrincipalPermissionMode =
            PrincipalPermissionMode.Custom;

        return host;
    }
}

```

Note: The sample *Surveys* application uses a simulated authorization manager. You must replace this with a real authorization manager in a production application.

The following code example from the **SimulatedWebServiceAuthorizationManager** class shows how to override the **CheckAccessCore** method in the **ServiceAuthorizationManager** class to provide a custom authorization decision.

```

C#
protected override bool CheckAccessCore(
    OperationContext operationContext)
{
    try
    {
        if (WebOperationContext.Current != null)
        {
            var headers =
                WebOperationContext.Current.IncomingRequest.Headers;
            if (headers != null)
            {
                var authorizationHeader =
                    headers[HttpRequestHeader.Authorization];
                if (!string.IsNullOrEmpty(authorizationHeader))
                {
                    if (authorizationHeader.StartsWith("user"))
                    {
                        var userRegex = new Regex(@"(\w+):([^\s]+)", 
                            RegexOptions.Singleline);
                        var username = userRegex.Match(authorizationHeader)
                            .Groups[1].Value;
                        var password = userRegex.Match(authorizationHeader)
                            .Groups[2].Value;
                    }
                }
            }
        }
    }
}

```



In this simulated authorization manager class, the **CheckAccessCore** method extracts the user name and password from the authorization header, calls a validation routine, and if the validation routine succeeds, it attaches a **ClaimsPrincipal** object to the web service context.

In the sample application, the validation routine does nothing more than check that the user name is one of several hard-coded values.

The following code example shows how the **RequestTo** method in the **HttpClient** class adds the authorization header with the user name and password credentials to the HTTP request that the mobile client sends to the various Tailspin web services.

```
C#
public static HttpWebRequest RequestTo(
    Uri surveysUri, string userName, string password)
{
    var request = (HttpWebRequest)WebRequest.Create(surveysUri);
    var authHeader = string.Format(CultureInfo.InvariantCulture,
        "user {0}:{1}", userName, password);
    request.Headers[HttpRequestHeader.Authorization] = authHeader;
    return request;
}
```

Notifying the Mobile Client of New Surveys

Tailspin wants a way to notify users of new surveys from the user's list of preferred tenants. Tenants are subscribers to the cloud-based Tailspin Surveys application who publish surveys. Users will then be able to synchronize the mobile client application and start using the new surveys.

OVERVIEW OF THE SOLUTION

Tailspin chose to use the Microsoft Push Notifications Service (MPNS) for Windows Phone to deliver information about relevant new surveys to users with Windows Phone 7 devices. This feature allows the cloud-based Surveys application to notify a user about a relevant new survey even when the mobile client application isn't running. This is possible even though you can't write a background task to run on the Windows Phone 7 device when your application isn't running.

Figure 3 shows, at a high level, how this notification process works for the Tailspin Surveys application.



You can't create applications that run in the background in Windows Phone 7 because constantly running applications will drain the device's battery.

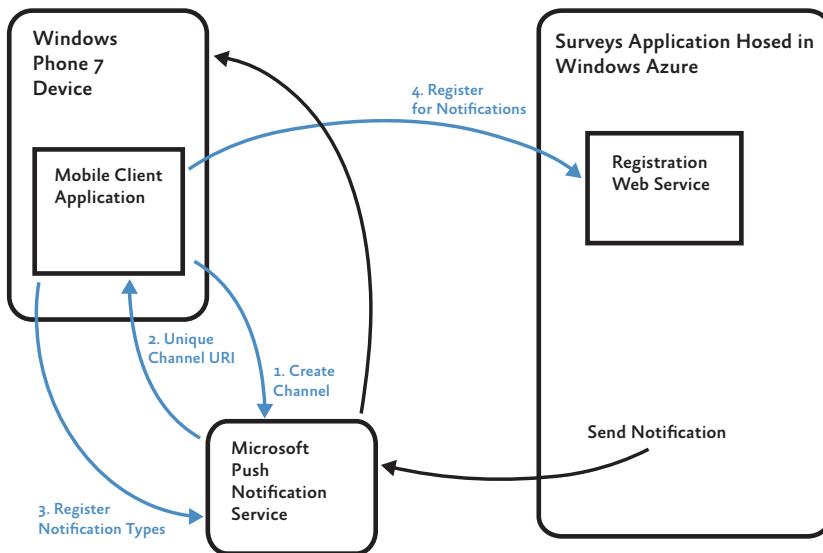


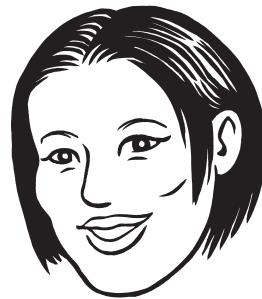
FIGURE 3
Push notifications for Windows Phone

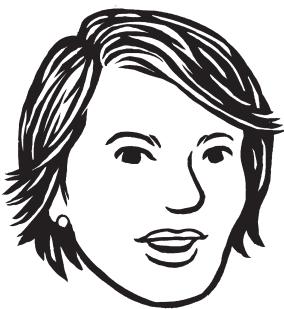
Figure 3 shows how an application on the Windows Phone 7 device can register for push notifications from another application—a service running in Windows Azure in this case. After the registration process is complete, the service in Windows Azure can push notifications to the Windows Phone 7 device. The following describes the process illustrated in Figure 3:

1. The registration process starts when the client application establishes a channel by sending a message to the MPNS.
2. The MPNS returns a URI that is unique to the instance of the client application on a particular Windows Phone 7 device.
3. The mobile client application can also specify which types of notification it will receive; this part of the registration process sets up a binding that enables the phone to associate a notification with the application and enables the user to launch the mobile application in response to receiving a message. The Windows Phone 7 Application Certification Requirements specify that you must provide the user with the ability to disable toast and tile notifications. You must run the application at least once to execute the code that establishes the channel before your phone can receive notifications.

There are two types of notification that the phone can receive when the application isn't running. The first is a "toast" notification that the phone displays in an overlay on the user's current screen. The user can click the message to launch the application. The second type of notification is a "tile" notification that changes the appearance of application's tile in the Quick Launch area of the phone's Start experience. If you change the appearance of a tile by sending a tile notification, and if you want to change the tile back to its original state, you'll have to send another message.

You can also use "raw" notifications to send data directly to the application, but this type of notification requires the application to be running in the foreground. If the application is not running, MPNS discards the notification and notifies the sender.





Remember that notifications are sent to the phone, not the application. This is because there is no guarantee that the application will be running when the Surveys service sends a message.

4. Establishing the channel simply enables the phone to receive messages. The client application must also register with the service that will send the notification messages by sending its unique Uniform Resource Identifier (URI) to the service. In the Surveys application, there is a Registration web service hosted in Windows Azure that the mobile client application can use to register its URI for notifications of new surveys.

The service can use the unique URI to send messages to the client application. The Surveys service sends a message to a mobile client by sending a message to the endpoint specified by the URI that the client sent when it registered. The MPNS hosts this endpoint and forwards messages from the service on to the correct device.

For more information about the certification requirements that relate to push notifications, see Section 6.2, “Push Notification Application,” of *Windows Phone 7 Application Certification Requirements*. You can download a PDF copy of this document from the Microsoft Download Center (<http://go.microsoft.com/?LinkId=9730558>).

Note: *The sample application uses the free, unauthenticated MPNS that limits you to sending 500 notification requests per channel per day. If you use the free version of MPNS, it also means that your application is vulnerable to spoofing and denial of service attacks if someone impersonates the worker role that is sending notifications.*

The authenticated MPNS has no restrictions on the number of notification messages you can send, and it requires the communication between your server component and MPNS to use Secure Sockets Layer (SSL) for sending notification messages.

For more information about the Microsoft Push Notification Service, see “Push Notifications Overview for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402558\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402558(VS.92).aspx)).

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements push notifications in more detail. As you go through this section, you may want to download the Visual Studio solution for the Windows Phone 7 Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

Registering for Notifications

Before a phone can receive notifications from the Surveys service of new surveys, it must obtain its unique URI from the MPNS. This registration process takes place when the user taps the Save button on the AppSettingsView page.

The following code example shows the **IRegistrationService** interface in the TailSpin.PhoneClient project that defines the registration operations that the mobile client can perform.

```
C#
public interface IRegistrationServiceClient
{
    IObservable<TaskSummaryResult> UpdateReceiveNotifications(
        bool receiveNotifications);
    IObservable<Unit> UpdateTenants(
        IEnumerable<TenantItem> tenants);
    IObservable<SurveyFiltersInformation>
        GetSurveysFilterInformation();
}
```

For details of the **UpdateTenants** and **GetSurveysFilterInformation** methods, see the section, “Filtering Data,” later in this chapter.

The **RegistrationServiceClient** class implements the **UpdateReceiveNotifications** method to handle the registration process for the mobile client. The following code example shows how the **UpdateReceiveNotifications** method manages the **HttpNotificationChannel** object in the mobile client application:

- The method first tries to retrieve previously cached channel information by calling the **Find** method.
- If the user is enabling notifications from MPNS, the method removes any cached channel data before binding to the MPNS channel and registering the unique URI with the Tailspin Surveys service.
- If the user is disabling notifications from MPNS, the method unregisters from the Tailspin Surveys service and removes the channel.

*The **HttpNotificationChannel** object stores the unique URI allocated by the MPNS.*

```
C#
private const string ChannelName = "tailspindemo.cloudapp.net";
private const string ServiceName = "TailSpinRegistrationService";
...
public IObservable<TaskSummaryResult>
UpdateReceiveNotifications(bool receiveNotifications)
{
    this.httpChannel = HttpNotificationChannel.Find(ChannelName);

    if (receiveNotifications)
    {
        if (this.httpChannel != null)
        {
```

```
        this.httpChannel.Close();
        this.httpChannel.Dispose();
        this.httpChannel = null;
    }

    this.httpChannel =
        new HttpNotificationChannel(ChannelName, ServiceName);

    ...

    // Bind to the MPNS channel and register the URI with the
    // Tailspin Surveys service.

}

if (this.httpChannel != null && this.httpChannel.ChannelUri
    != null)
{
    return
        this.CallUpdateReceiveNotifications(
            false, this.httpChannel.ChannelUri)
        .Select(
            s =>
            {
                this.httpChannel.Close();
                this.httpChannel.Dispose();
                this.httpChannel = null;

                return s;
            });
}

return Observable.Return(TaskSummaryResult.Success);
}
```

When the user is enabling notifications, the code in the following example from the **UpdateReceiveNotifications** method binds to the MPNS channel and registers the URI with the Tailspin Surveys service. It does this by using the **Observable.FromEvent** method to set up event handlers for the **ChannelUriUpdated** and **ErrorOccurred** events on the **HttpChannel** object before it opens the channel. In the **ChannelUriUpdated** event handler, it invokes the **BindChannelAndUpdateDeviceUri** method. The method also creates a timer task to test for timeouts. Finally, it returns the observable **TaskSummaryResult** object from whichever of the three asynchronous tasks completes first.

```
C#
var channelUriUpdated = from o in
    Observable.FromEvent<NotificationChannelUriEventArgs>(
        h => this.httpChannel.ChannelUriUpdated += h,
        h => this.httpChannel.ChannelUriUpdated -= h)
    from summary in this.BindChannelAndUpdateDeviceUriInService()
    select summary;

// If this line is not present, the preceding subscription is
// never executed.
this.httpChannel.ChannelUriUpdated += (s, a) => { };

var channelUriUpdateFail = from o in
    Observable.FromEvent<NotificationChannelErrorEventArgs>(
        h => this.httpChannel.ErrorOccurred += h,
        h => this.httpChannel.ErrorOccurred -= h)
    select TaskSummaryResult.UnknownError;

this.httpChannel.Open();

// If the notification service does not respond in time, it
// is assumed that the server is unreachable.
var timeout =
    Observable
        .Interval(TimeSpan.FromSeconds(30))
        .Select(i => TaskSummaryResult.UnreachableServer);

return channelUriUpdated.Amb(channelUriUpdateFail).Amb(timeout)
    .Take(1);
```

Note: The **Take(1)** method does not force execution of the observables; it simply makes sure that the code executes only once.

The following code example shows the **BindChannelAndUpdateDeviceUriInService** method in the **RegistrationServiceClient** class that configures the phone to respond to toast and tile notifications and then calls the **CallUpdateReceiveNotifications** method to register the unique URI with the Tailspin Surveys web service.

```
C#
private IObservable<TaskSummaryResult>
    BindChannelAndUpdateDeviceUriInService()
{
    this.httpChannel = HttpNotificationChannel.Find(ChannelName);
```

```

if (!this.httpChannel.IsShellToastBound)
{
    this.httpChannel.BindToShellToast();
}

if (!this.httpChannel.IsShellTileBound)
{
    this.httpChannel.BindToShellTile();
}

return this.CallUpdateReceiveNotifications(
    true, this.httpChannel.ChannelUri);
}

```

The following code example shows how the **CallUpdateReceiveNotifications** method in the **RegistrationServiceClient** class registers the clients unique URI with the Tailspin Surveys web service by asynchronously invoking a web method and passing it a **DeviceDto** object that contains the phone's unique URI.

```

C#
private readonly Uri serviceUri;
...
private IObservable<TaskSummaryResult>
    CallUpdateReceiveNotifications(
        bool receiveNotifications, Uri channelUri)
{
    var device = new DeviceDto
    {
        Uri = channelUri.ToString(),
        RecieveNotifications = receiveNotifications
    };

    var uri = new Uri(this.serviceUri, "Notifications");
    return
        HttpClient
            .RequestTo(uri, this.settingsStore.UserName,
            this.settingsStore.Password)
            .PostJson(device)
            .Select(u => TaskSummaryResult.Success);
}

```

This method uses the **HttpClient** and **HttpWebRequestExtensions** classes to post the data transfer object to the web service. Tailspin developed these classes to simplify sending asynchronous HTTP requests from the mo-

bile client application. The following code example shows the **RequestTo** method from the **HttpClient** class that creates an **HttpWebRequest** object.

```
C#
public static HttpWebRequest RequestTo(
    Uri surveysUri, string userName, string password)
{
    var request = (HttpWebRequest)WebRequest.Create(surveysUri);
    var authHeader = string.Format(CultureInfo.InvariantCulture,
        "user {0}:{1}", userName, password);
    request.Headers[HttpRequestHeader.Authorization] = authHeader;
    return request;
}
```

For an explanation of the authorization header that Tailspin Surveys uses to authenticate the client, see the section, “Authenticating with the Surveys Service,” earlier in this chapter.

The following code example shows the **PostJson** method from the **HttpWebRequestExtensions** class that uses the **Observable** object from the Reactive Extensions (Rx) framework to call the web method asynchronously. This code example shows four steps:

- It uses the **FromAsyncPattern** method to create an asynchronous function that returns an observable **Stream** object from the **HttpWebRequest** object.
- It uses the **SelectMany** method to asynchronously attach the payload to the request stream.
- It then returns the **WebResponse** object from an asynchronous call to the **HttpWebRequest** object.
- The method returns an **IObservable<Unit>** instance, which is equivalent to a **null** in Rx, when it has a complete HTTP response message.

```
C#
public static IObservable<Unit> PostJson<T>(
    this HttpWebRequest request, T obj)
{
    request.Method = "POST";
    request.ContentType = "application/json";

    return
        Observable
            .FromAsyncPattern<Stream>(
                request.BeginGetRequestStream, request.EndGetRequestStream())
            .SelectMany(
```

```

requestStream =>
{
    using (requestStream)
    {
        var serializer =
            new DataContractJsonSerializer(typeof(T));
        serializer.WriteObject(requestStream, obj);
        requestStream.Close();
    }

    return
        Observable.FromAsyncPattern<WebResponse>(
            request.BeginGetResponse,
            request.EndGetResponse)();
},
(requestStream, webResponse) => new Unit());
}

```

So far, this section has described Tailspin's implementation of the client portion of the registration process. The next part of this section describes how the Surveys service stores the unique URI that the client obtained from the MPNS in Windows Azure storage whenever a Windows Phone 7 device registers for notifications of new surveys.

The following code example shows the implementation of the registration web service that runs in Windows Azure. You can find this class is in the Tailspin.Surveys.Services project.

```

C#
[AspNetCompatibilityRequirements(RequirementsMode =
    AspNetCompatibilityRequirementsMode.Allowed)]
[ServiceBehavior(InstanceContextMode =
    InstanceContextMode.PerCall)]
public class RegistrationService : IRegistrationService
{
    ...
    private readonly IUserDeviceStore userDeviceStore;
    ...
    public void Notifications(DeviceDto device)
    {
        var username = Thread.CurrentPrincipal.Identity.Name;

        bool isWellFormedUriString = Uri.IsWellFormedUriString(
            device.Uri, UriKind.Absolute);
        if (isWellFormedUriString)

```

```
        {
            if (device.RecieveNotifications)
            {
                this.userDeviceStore.SetUserDevice(username,
                    new Uri(device.Uri));
            }
            else
            {
                this.userDeviceStore.RemoveUserDevice(
                    new Uri(device.Uri));
            }
        }
    }
}
```

The **Notifications** method receives a **DeviceDto** parameter object that includes the unique URI allocated to the phone by the MPNS and a Boolean flag to specify whether the user is subscribing or unsubscribing to notifications. The service saves the details in the **DeviceDto** object in the device store in Windows Azure storage.

Note: In the sample application, the service does not protect the data as it reads and writes to Windows Azure storage. When you deploy your application to Windows Azure, you should secure your table, binary large object (BLOB), and queue endpoints using SSL.

Although phones can explicitly unsubscribe from notifications, the application also removes devices from the device store if it detects that they are no longer registered with the MPNS when it sends a notification.

Sending Notifications

When a survey creator saves a new survey, the Surveys service in Windows Azure retrieves all the URLs for the subscribed Windows Phone 7 devices and then sends the notifications to all the devices that subscribe to notifications for surveys created by that particular survey creator.

The Tailspin Surveys service sends tile notifications of new surveys to subscribed Windows Phone 7 devices.

The following code example from the **NewSurveyNotificationCommand** class in the TailSpin.Workers.Notifications project shows how the Windows Azure worker role retrieves the list of subscribed phones and sends the notifications. This method uses the filtering service to retrieve the list of devices that should receive notifications about a particular survey. For more information, see the section, “Filtering Data,” later in this chapter.

```

C#
public void Run(NewSurveyMessage message)
{
    var survey = this.surveyStore.GetSurveyByTenantAndSlugName(
        message.Tenant, message.SlugName, false);

    if (survey != null)
    {
        var deviceUris =
            from user in this.filteringService.GetUsersForSurvey(survey)
            from deviceUri in this.userDeviceStore.GetDevices(user)
            select deviceUri;

        foreach (var deviceUri in deviceUris)
        {
            this.pushNotification.PushTileNotification(
                deviceUri.ToString(),
                "New Surveys",
                null,
                0,
                uri => this.userDeviceStore.RemoveUserDevice(new
                    Uri(uri)));
        }
    }
}

```

The **Run** method also passes a reference to a callback method that removes from the device store phones that are no longer registered with the MPNS.

Note: For more information about how the **Run** method is triggered, and about retry policies if the **Run** method throws an exception, see the section “The Worker Role ‘Plumbing’ Code” in Chapter 4, “Building a Scalable, Multi-Tenant Application for Windows Azure,” of the book, *Developing Applications for the Cloud on the Microsoft Windows Azure™ Platform*. This is available on MSDN (<http://msdn.microsoft.com/en-us/library/ff966499.aspx>).

The following code example shows the **PushTileNotification** method in the **PushNotification** class that’s invoked by the worker role command to send the notification. This method creates the tile notification message to send to the MPNS before it calls the **SendMessage** method.

```
C#
public void PushTileNotification(string channelUri,
    string message, string backgroundImage, int count,
    DeviceNotFoundInMpns callback)
{
    byte[] payload = TileNotificationPayloadBuilder.Create(
        message, backgroundImage, count);
    string messageId = Guid.NewGuid().ToString();
    this.SendMessage(NotificationType.Tile, channelUri, messageId,
        payload, callback);
}
```

The **SendMessage** method sends messages to the MPNS for forwarding on to the subscribed devices. The following code example shows how the **SendMessage** method sends the message to the MPNS; the next code example shows how the **SendMessage** method receives a response from the MPNS.

```
C#
private void SendMessage(NotificationType notificationType,
    string channelUri, string messageId, byte[] payload,
    DeviceNotFoundInMpns callback)
{
    try
    {
        WebRequest request = WebRequestFactory.CreatePhoneRequest(
            channelUri, payload.Length, notificationType, messageId);
        request.BeginGetRequestStream(
            ar =>
        {
            // After the async call returns, get the Stream object.
            Stream requestStream = request.EndGetRequestStream(ar);

            // Start to write the payload to the stream
            // asynchronously.
            requestStream.BeginWrite(
                payload,
                0,
                payload.Length,
                iar =>
            {
                // After the writing completes, close the stream.
                requestStream.EndWrite(iar);
                requestStream.Close();
            });
        });
    }
}
```



The **SendMessage** method acquires the stream and writes to it asynchronously because the service may be sending messages to hundreds or even thousands of devices, and for every device, it needs to open and write to a stream.

```
// Switch to receiving the response from MPNS.

...

},
null);
},
null);
}
catch (WebException ex)
{
    if (ex.Status == WebExceptionStatus.ProtocolError)
    {
        this.OnNotified(notificationType,
        (HttpWebResponse)ex.Response);
    }
    Trace.TraceError(ex.TraceInformation());
}
}
```

After the **SendMessage** method sends the message to the MPNS, it waits for a response. It must receive the message asynchronously because the MPNS does not return a response until it, in turn, receives a response from the Windows Phone 7 device. The **SendMessage** method notifies its caller of the response through the **OnNotified** method call.

C#

```
...
// Switch to receiving the response from MPNS.
request.BeginGetResponse(
    iar =>
{
    try
    {
        using (WebResponse response =
            request.EndGetResponse(iar))
        {
            // Notify the caller with the MPNS results.
            this.OnNotified(notificationType,
            (HttpWebResponse)response);
        }
    }
    catch (WebException ex)
    {
```

```

        if (ex.Status ==
            WebExceptionStatus.ProtocolError)
        {
            this.OnNotified(notificationType,
                (HttpWebResponse)ex.Response);
        }
        if (((HttpWebResponse)ex.Response).StatusCode
            == HttpStatusCode.NotFound)
        {
            callback(channelUri);
        }
        Trace.TraceError(ex.TraceInformation());
    }
},
null);
...
protected void OnNotified(NotificationType notificationType,
    HttpWebResponse response)
{
    var args = new NotificationArgs(notificationType, response);
    ...
}

```

If the **SendMessage** method receives a “404 Not Found” response code from the MPNS, it removes the stored subscription details from the store because this response indicates that the device is no longer registered with the MPNS.

The following table summarizes the information available from the MPNS in the response.

Item	Description
Message ID	This is a unique identifier for the response message.
Notification Status	This is the status of the notification message. Possible values are Received , Dropped , or QueueFull . You could use this value to determine whether you need to resend the message to this device.
Device Connection Status	This is the connection status of the device. Possible values are Connected , InActive , Disconnected , and TempDisconnected . If it is important that a device receive the message, you can use this value to determine whether you need to resend the message later.
Subscription Status	This is the device's subscription status. Possible values are Active and Expired . If a device's subscription has expired, you can remove its URI from your list of subscribed devices.

For more information about sending push notifications, see “How to: Send a Push Notification from a Web Service for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402545\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402545(VS.92).aspx)).

Notification Payloads

Tile notifications include three items of data:

- A background PNG or JPG image for the tile that should be 173 pixels by 173 pixels in size
- A text label that overlays the background image
- A count value that also overlays the background image; for example, the number of new surveys available

A toast notification includes two strings that appear in the toast: the first string is a header, and the second string is optional body text.

You must make sure that the strings you send on both tile and toast notifications fit in the available space.

The following code example from the **TileNotificationPayloadBuilder** class in the TailSpin.Web.Survey.Shared project shows how the Surveys service constructs a tile notification message.

C#

```
public static byte[] Create(string title,
    string backgroundImage = null, int count = 0)
{
    using (var stream = new MemoryStream())
    {
        var settings = new XmlWriterSettings
        { Indent = true, Encoding = Encoding.UTF8 };
        using (var writer = XmlWriter.Create(stream, settings))
        {
            if (writer != null)
            {
                writer.WriteStartDocument();
                writer.WriteStartElement("wp", "Notification",
                    "WPNotification");
                writer.WriteStartElement("wp", "Tile", "WPNotification");
                writer.WriteStartElement("wp", "BackgroundImage",
                    "WPNotification");
                writer.WriteValue(backgroundImage ?? string.Empty);
                writer.WriteEndElement();
                writer.WriteStartElement("wp", "Count",
                    "WPNotification");
                writer.WriteValue(count == 0 ? string.Empty :
                    count.ToString(CultureInfo.InvariantCulture));
                writer.WriteEndElement();
            }
        }
    }
}
```

```
        writer.WriteStartElement("wp", "Title",
            "WPNotification");
        writer.WriteValue(title);
        writer.WriteEndElement();
        writer.WriteEndDocument();
        writer.Close();
    }

    byte[] payload = stream.ToArray();
    return payload;
}
}
}
```

The *backgroundImage* parameter is the URL of the image, which must be no more than 80 KB in size.

The following code example from the **ToastNotificationPayloadBuilder** class in the TailSpin.Workers.Notifications project shows how the Surveys service constructs a toast notification message.

```
C#
public static byte[] Create(string text1, string text2 = null)
{
    using (var stream = new MemoryStream())
    {
        var settings = new XmlWriterSettings { Indent = true,
            Encoding = Encoding.UTF8 };
        XmlWriter writer = XmlWriter.Create(stream, settings);
        writer.WriteStartDocument();
        writer.WriteStartElement("wp", "Notification",
            "WPNotification");
        writer.WriteStartElement("wp", "Toast", "WPNotification");
        writer.WriteStartElement("wp", "Text1", "WPNotification");
        writer.WriteValue(text1);
        writer.WriteEndElement();
        writer.WriteStartElement("wp", "Text2", "WPNotification");
        writer.WriteValue(text2);
        writer.WriteEndElement();
        writer.WriteEndElement();
        writer.WriteEndDocument();
        writer.Close();

        byte[] payload = stream.ToArray();
        return payload;
    }
}
```

Accessing Data in the Cloud

The Surveys application stores its data in the cloud using a combination of Windows Azure tables and BLOBs. The mobile client application needs to access this data. For example, it must be able to retrieve survey definitions before it can display the questions to the phone user, and it must be able to save completed survey responses back to the cloud where they will be available for analysis by the survey creator.

GOALS AND REQUIREMENTS

The mobile client application must be able to reliably download survey definitions and reliably upload survey responses. Making sure that surveys download reliably is important because survey creators want to be sure that surveys are delivered to all potential surveyors in order to maximize the number of responses. Making sure that surveys upload reliably is important because survey creators want to receive the maximum number of completed surveys, with no duplication of results.

The developers at Tailspin are aware that some surveyors may have limited bandwidth available, so they wanted to control the amount of bandwidth used to transfer data between the phone and the service. In addition to this, the developers at Tailspin want to make sure that the application performs well when it transfers data to and from the cloud.

The developers also wanted a solution that was as simple as possible to implement, and that they could easily customize in the future if, for example, authentication requirements changed.

Finally, again with a view to the future, the developers wanted a solution that could potentially work with platforms other than Windows Phone 7.

OVERVIEW OF THE SOLUTION

Tailspin considered three separate aspects of the solution: how to implement the server, how to implement the client, and the format of the data that the application moves between the phone and the cloud.

Exposing the Data in the Cloud

The developers at Tailspin decided to use the Windows Communication Foundation (WCF) Representational State Transfer (REST) programming model to expose the data in the Tailspin Surveys service. Tailspin also considered using WCF Data Services, but when Tailspin began developing the mobile client application, the WCF Data Services support for Windows Azure table storage was still a pre-release version.

Tailspin is using the WCF REST programming model to exchange data between the mobile client and the cloud-based service.

Data Formats

Because Tailspin is using a custom WCF service, they must use custom data transfer objects to exchange data between the mobile client application and the cloud-based service. Tailspin chose to use a Java Script Object Notation (JSON) format for moving the data over the wire because it produces a compact payload that reduces bandwidth requirements, is relatively easy to use, and will be usable on platforms other than the Windows Phone 7 platform.

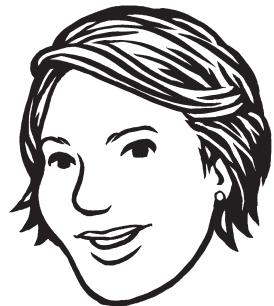
Note: *If, in the future, Tailspin moves to WCF Data Services, it will no longer require the custom data transfer objects because WCF Data Services use the OData protocol to move data over the wire.*

Tailspin also considered compressing the data before transferring it over the network to reduce bandwidth utilization, but the developers at Tailspin decided that the additional CPU and battery usage on the phone that this would require outweighed the benefits in this particular case. You should evaluate this trade-off between the cost of bandwidth and battery consumption in your own application before you decide whether to compress data that you need to move over the network.

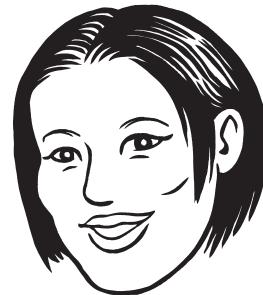
Consuming the Data

The developers at Tailspin implemented a set of custom classes in the mobile client application to handle the data transfer with the Tailspin web service. The classes on the mobile client interact with the WCF REST service and parse the data received from the service. Tailspin's analysis of the data transfer requirements for the Windows Phone 7 application identified only two types of interaction with the service: a "Get Surveys" operation and a "Send Survey Result" operation, so the implementation of custom client should be quite simple. Furthermore, the "Send Survey Result" operation always appends the result to the store on the server so there are no concurrency issues, and survey creators cannot modify a survey design after they publish it so there are no versioning issues.

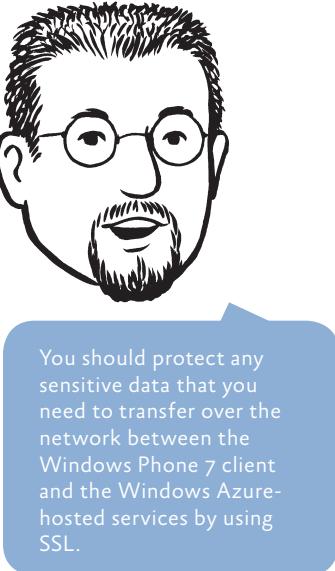
Note: *In the future, Tailspin may decide to use WCF Data Services and the OData protocol. OData client libraries, including a version for Windows Phone 7, are available for download on the Open Data Protocol website (<http://www.odata.org/developers/odata-sdks>). At the time of this writing, the version of the OData Client Library for Windows Phone 7 was a pre-release version. You should check to see whether a later release is available.*



In the future, Tailspin would like to use WCF Data Services because of its support for the OData standard. For more information about OData, see the Open Data Protocol website (<http://www.odata.org>).



On the phone, additional CPU usage affects both the responsiveness of the device and its battery life.



Using the OData Client Library would minimize the amount of code that the developers would have to write because the library and the code generated by using the DataSvcUtil utility fully encapsulate the calls to the WCF Data Service endpoint. Furthermore, using the client library offers advanced features such as batching, client-side state management, and conflict resolution.

For a walkthrough that shows how to use the OData Client Library on the Windows Phone 7 platform, see the post, “Developing a Windows Phone 7 Application that consumes OData,” on Phani Raj’s blog (<http://blogs.msdn.com/b/phaniraj/>).

Tailspin also evaluated the Microsoft Sync Framework to handle synchronizing data between the phone and Windows Azure, but the Microsoft Sync Framework 3.0 was also a pre-release version when they began their implementation.

Using SSL

When Tailspin developed the mobile client application, it wasn’t possible to install certificates on the Windows Phone 7 device, so to implement SSL, it was necessary to use a server certificate from a trusted third-party company, such as VeriSign, instead of using a self-signed certificate. Therefore, the sample application does not secure the WCF REST service with SSL, so a malicious client can impersonate the phone client and send malicious data.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that enables the mobile client application to access data in the cloud in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

Creating a WCF REST Service in the Cloud

The TailSpin.Services.Surveys project includes a standard WCF REST Service named SurveysService hosted in Windows Azure that exposes the survey data to the Windows Phone 7 client application. The Windows Azure web role defined in the TailSpin.Services.Surveys.Host.Azure project populates a routing table that includes a route to the Surveys service in its Global.asax.cs file. The following code example shows how the **RegisterRoutes** method creates the **Route Table** object.

```
C#
public class Global : HttpApplication
{
    protected void Application_Start(object sender, EventArgs e)
    {
        RegisterRoutes();
    }

    private static void RegisterRoutes()
    {
        var customServiceHostFactory = new
            CustomServiceHostFactory(ContainerLocator.Container);
        ...
        RouteTable.Routes.Add(new ServiceRoute("Survey",
            customServiceHostFactory, typeof(SurveysService)));
        ...
    }
}
```

This **SurveysService** class implements the WCF REST service endpoints. The following code example shows the **GetSurveys** method in the **SurveysService** class that exposes the surveys data stored in Windows Azure storage.

```
C#
public SurveyDto[] GetSurveys(string lastSyncUtcDate)
{
    DateTime fromDate;
    if (!string.IsNullOrEmpty(lastSyncUtcDate))
    {
        if (DateTime.TryParse(lastSyncUtcDate, out fromDate))
        {
            fromDate = DateTime.SpecifyKind(
                fromDate, DateTimeKind.Utc);
        }
        else
        {
            throw new FormatException("lastSyncUtcDate is in an incorrect
                format. The format should be: yyyy-MM-ddTHH:mm:ss");
        }
    }
    else
    {
        fromDate = new DateTime(
            1900, 1, 1, 0, 0, 0, DateTimeKind.Utc);
    }
}
```



You should secure all your Windows Azure BLOB, table, and queue endpoints using SSL.

```

var username = Thread.CurrentPrincipal.Identity.Name;

return this.filteringService
    .GetSurveysForUser(username, fromDate)
    .Select(s => new SurveyDto
    {
        SlugName = s.SlugName,
        Title = s.Title,
        Tenant = s.Tenant,
        Length = 5 * s.Questions.Count,
        IconUrl = this.GetIconUrlForTenant(s.Tenant),
        CreatedOn = s.CreatedOn,
        Questions = s.Questions.Select(q =>
            new QuestionDto
            {
                PossibleAnswers = q.PossibleAnswers,
                Text = q.Text,
                Type = Enum.GetName(typeof(QuestionType), q.Type)
            }).ToList()
        }).ToArray();
}

```

This example shows how the Surveys service returns survey definitions to the mobile client in an array of **SurveyDto** objects that represent surveys added to the service after a specified date. It also demonstrates applying a filter to the request. In the current version of the application, the filter returns a list of surveys from the phone user's preferred list of tenants. For more information about how Tailspin implemented the filtering behavior, see the section, "Filtering Data," later in this chapter.

To enable the mobile client to upload survey answers, the **Surveys Service** class provides two methods: one for uploading individual images and sound clips, and one for uploading complete survey answers. The following code example shows how the **AddMediaAnswer** method saves an image or a sound clip to Windows Azure BLOB storage and returns a URL that points to the BLOB.

```

C#
public string AddMediaAnswer(Stream media, string type)
{
    var questionType =
        (QuestionTypeEnum.Parse(typeof(QuestionType), type));
    return this.mediaAnswerStore.SaveMediaAnswer(media,
        questionType);
}

```

The following code example shows the **AddSurveyAnswers** method that receives an array of **SurveyAnswerDto** objects that it unpacks and saves in the survey answer store in Windows Azure storage.

```
C#
public void AddSurveyAnswers(SurveyAnswerDto[] surveyAnswers)
{
    foreach (var surveyAnswerDto in surveyAnswers)
    {
        this.surveyAnswerStore.SaveSurveyAnswer(
            new SurveyAnswer
            {
                Title = surveyAnswerDto.Title,
                SlugName = surveyAnswerDto.SlugName,
                Tenant = surveyAnswerDto.Tenant,
                QuestionAnswers = surveyAnswerDto.QuestionAnswers
                    .Select(qa => new QuestionAnswer
                    {
                        QuestionText = qa.QuestionText,
                        PossibleAnswers = qa.PossibleAnswers,
                        QuestionType = (QuestionType)Enum.Parse(
                            typeof(QuestionType), qa.QuestionType),
                        Answer = qa.Answer
                    }).ToList()
            });
    }
}
```

Consuming the Data in the Windows Phone 7 Client Application

The mobile client application uses the methods exposed by the Surveys service to send and receive survey data.

The following code example shows the **ISurveysServiceClient** interface that defines the set of asynchronous WCF REST calls that the mobile client application can make.

```
C#
public interface ISurveysServiceClient
{
    IObservable<IEnumerable<SurveyTemplate>>
        GetNewSurveys(string lastSyncDate);
    IObservable<Unit>
        SaveSurveyAnswers(IEnumerable<SurveyAnswer> surveyAnswers);
}
```

The **SurveysServiceClient** class implements this interface, and the following code example shows the **GetNewSurveys** method that sends the request to the service and returns the observable list of surveys to the application. This method makes the asynchronous web request by using the **GetJson** extension method from the **HttpWebRequestExtensions** class, converts the returned data transfer objects to **SurveyTemplate** objects, and then returns an observable sequence of **SurveyTemplate** objects.

```
C#
public IObservable<IEnumerable<SurveyTemplate>>
    GetNewSurveys(string lastSyncDate)
{
    var surveysPath = string.Format(CultureInfo.InvariantCulture,
        "Surveys?lastSyncUtcDate={0}", lastSyncDate);
    var uri = new Uri(this.serviceUri, surveysPath);

    return
        HttpClient
            .RequestTo(uri, this.settingsStore.UserName,
            this.settingsStore.Password)
            .GetJson<IEnumerable<SurveyDto>>()
            .Select(ToSurveyTemplate);
}
```



Tailspin decided to support clients that upload individual media items instead of using multi-part messages in order to support the widest set of possible client platforms.

Note: For more information about the **HttpClient** class that makes the asynchronous web request, see the section, “Registering for Notifications,” earlier in this chapter; it describes the **HttpClient** and **HttpWebRequestExtensions** classes.

To save completed survey answers to the Surveys service, the client must first save any image and sound clip answers. It uploads each media answer in a separate request, and then it includes the URL that points to the BLOB that contains the media when the client uploads the complete set of answers for a survey. It would be possible to upload the survey answers and the media in a single request, but this may require a large request that exceeds the maximum upload size configured in the service. By uploading all the media items first, the worst that can happen if there is a failure is that there are orphaned media items in Windows Azure storage.

The following code example shows the first part of the **SaveAndUpdateMediaAnswers** method that creates a list of answers that contain media.

C#

```
var mediaAnswers =
  from surveyAnswer in surveyAnswersDto
  from answer in surveyAnswer.QuestionAnswers
  where
    answer.Answer != null &&
    (answer.QuestionType ==
      Enum.GetName(typeof(QuestionType), QuestionType.Picture) ||
    answer.QuestionType ==
      Enum.GetName(typeof(QuestionType), QuestionType.Voice))
  select answer;
```

The method then iterates over this list and asynchronously creates HTTP requests to post each media item to the Tailspin Surveys service.

C#

```
foreach (var answer in mediaAnswers)
{
  var mediaAnswerPath = string.Format(
    CultureInfo.InvariantCulture,
    "MediaAnswer?type={0}",
    answer.QuestionType);
  var mediaAnswerUri = new Uri(this.serviceUri, mediaAnswerPath);
  byte[] mediaFile = GetFile(answer.Answer);

  var request = HttpClient.RequestTo(mediaAnswerUri,
    this.settingsStore.UserName, this.settingsStore.Password);
  request.Method = "POST";
  request.Accept = "application/json";
  ...
}
```

The method then, asynchronously, needs to send each of these requests and retrieve the response to each request in order to extract the URL that points to the BLOB where the service saved the media item. It must then assign the returned URLs to the **Answer** property of the correct **QuestionAnswerDto** data transfer object. The following code example shows how the **SaveAndUpdateMediaAnswers** method sends the media answers to the Tailspin Surveys service asynchronously. The code example shows how this operation is broken down into the following steps:

1. The method first makes an asynchronous call to access the HTTP request stream as an observable object.
2. The method writes the media item to the request stream and then makes an asynchronous call to access the HTTP response stream.
3. It reads the URL of the saved media item from the response stream and sets the **Answer** property of the **QuestionAnswerDto** object.
4. The **ForkJoin** method makes sure that the media answers are uploaded in parallel, and after all the media answers are uploaded, the method returns an **IObservable<Unit>** object.

C#

```
var mediaAnswerObservables = new List<IObservable<Unit>>();
foreach (var answer in mediaAnswers)
{
    ...
    QuestionAnswerDto answerCopy = answer;
    var saveFileAndUpdateAnswerObservable =
        Observable
            .FromAsyncPattern<Stream>(request.BeginGetRequestStream,
            request.EndGetRequestStream())
            .SelectMany(
                requestStream =>
            {
                using (requestStream)
                {
                    requestStream.Write(mediaFile, 0, mediaFile.Length);
                    requestStream.Close();
                }
                return
                    Observable.FromAsyncPattern<WebResponse>(
                        request.BeginGetResponse, request.EndGetResponse());
            },
            (requestStream, webResponse) =>
            {
                using (var responseStream =
                    webResponse.GetResponseStream())
                {
                    var responseSerializer = new
                        DataContractJsonSerializer(typeof(string));
                    ...
                    responseStream.Close();
                }
            }
        );
    mediaAnswerObservables.Add(saveFileAndUpdateAnswerObservable);
}
return Observable
    .FromAsyncPattern<List<QuestionAnswerDto>>(ForkJoin);
}
```

```

        answerCopy.Answer =
            (string)responseSerializer.ReadObject(responseStream);
    }

    return new Unit();
});

mediaAnswerObservables.Add(saveFileAndUpdateAnswerObservable);
}
return mediaAnswerObservables.ForkJoin().Select(u => new Unit());
}

```

The following code example shows how the mobile client uploads completed survey answers. It first creates the data transfer objects, then it uploads any media answers using the **SaveAndUpdateMediaAnswers** method described earlier, and finally, it uploads the **SurveyAnswerDto** object using the **HttpClient** class. The **SelectMany** method here makes sure that all the media answers are uploaded before the **SurveyAnswerDto** object.

```

C#
public IObservable<Unit>
    SaveSurveyAnswers(IEnumerable<SurveyAnswer> surveyAnswers)
{
    var surveyAnswersDto = ToSurveyAnswersDto(surveyAnswers);
    var saveAndUpdateMediaAnswersObservable =
        this.SaveAndUpdateMediaAnswers(surveyAnswersDto);

    var uri = new Uri(this.serviceUri, "SurveyAnswers");
    var saveSurveyAnswerObservable =
        HttpClient
            .RequestTo(uri, this.settingsStore.UserName,
            this.settingsStore.Password)
            .PostJson(surveyAnswersDto);

    return saveAndUpdateMediaAnswersObservable.SelectMany(
        u => saveSurveyAnswerObservable);
}

```

Both the **GetNewSurveys** and **SaveSurveyAnswer** methods are called from the **SurveysSynchronizationService** class that is responsible for coordinating which surveys should be downloaded and which survey answers should be uploaded. For a description of the synchronization process, see Chapter 5, “Using Services on the Phone.”

Filtering Data

The Surveys service filters lists of surveys for notifications and for synchronization.

Users of the mobile client application can express preferences for which surveys they'd like to see on their devices. In the initial version of the application, the criteria will be a list of preferred tenants (survey creators such as Adatum and Fabrikam in the sample). There are two scenarios where the Tailspin Surveys service must filter data based on filter criteria from the client:

- The cloud service can send notifications to mobile clients to tell them when new surveys are available. The notification service should only send a notification to a user if one of the user's preferred tenants created the new survey.
- When the mobile client synchronizes its data with the Tailspin Surveys service, the phone should only receive surveys created by the user's preferred tenants.

Note: *For more details about the notification process, see the section, "Notifying the Mobile Client of New Surveys," earlier in this chapter. For more details about the synchronization process, see the section, "Synchronizing Data between the Client and the Cloud," in Chapter 5, "Using Services on the Phone."*

OVERVIEW OF THE SOLUTION

There are three tasks that make up the filtering functionality in the Tailspin Surveys application: registering the user's preferences, identifying which devices to notify when a tenant adds a new survey, and identifying which surveys to include in the synchronization process.

Registering User Preferences

The user creates a list of preferred tenants on the FilterSettingsView page in the mobile client application. When the user saves the settings, the Tailspin Surveys mobile client application sends the preferences to the Tailspin Surveys registration service that then stores the list of preferred tenants in Windows Azure table storage.

Figure 4 outlines the way that the mobile client application saves the user's settings in the Tailspin Surveys web service.

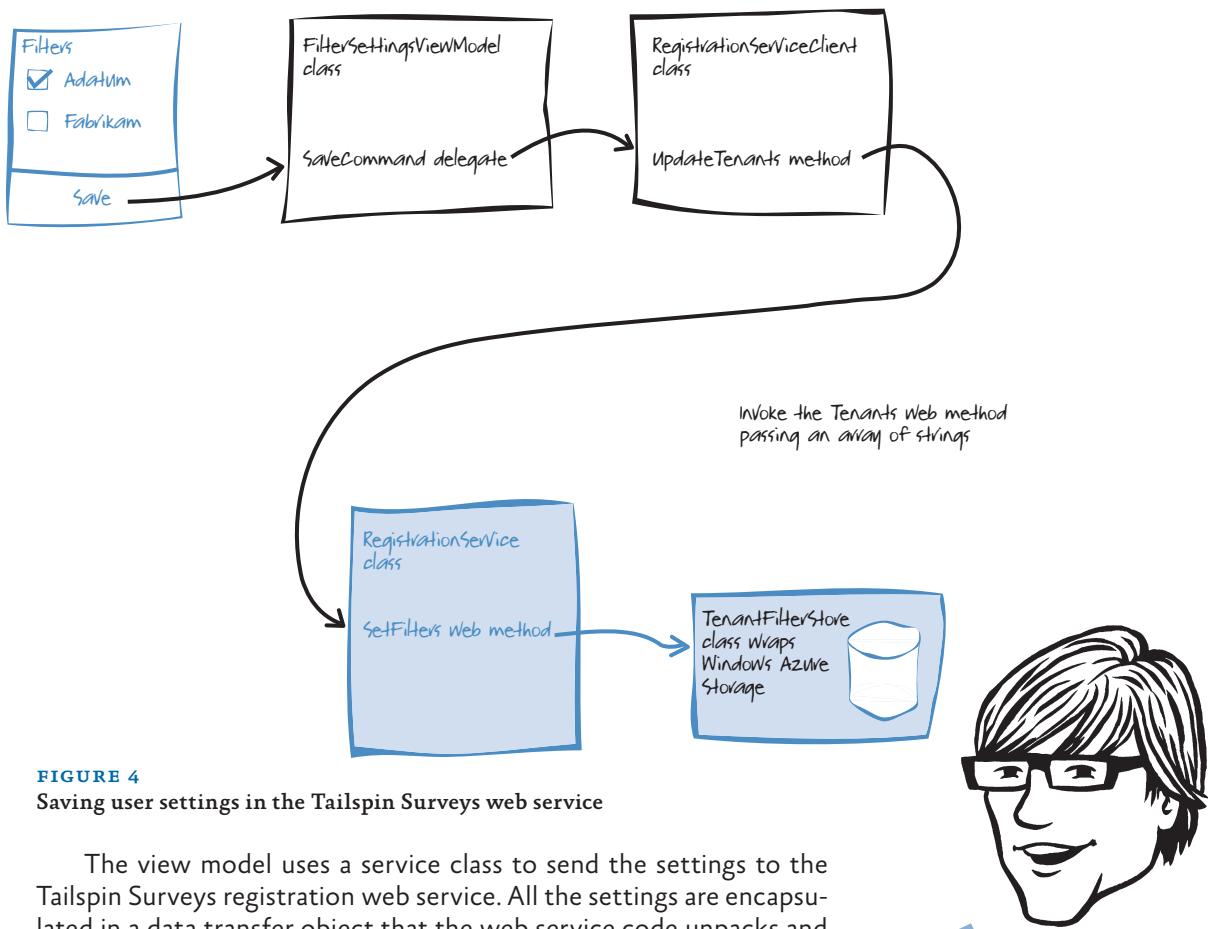


FIGURE 4
Saving user settings in the Tailspin Surveys web service

The view model uses a service class to send the settings to the Tailspin Surveys registration web service. All the settings are encapsulated in a data transfer object that the web service code unpacks and saves in Windows Azure table storage.

The following table shows the structure of the tenant filter table in Windows Azure table storage that stores the list of preferred tenants for each user.

Column	Notes
Tenant name	A string holding the tenant name. This column is the table's partition key.
User name	A string holding the user name. This column is the table's row key.

The application can use this table to retrieve a list of tenants that a particular user is interested in or a list of users who are interested in a particular tenant.

If you query this table, including the tenant name in the **where** clause, the query will run efficiently because it only needs to access a single partition.

For example, finding all the users who are interested in surveys created by Adatum is an efficient query. However, asking for a list of all the tenants that a particular user is interested in is an inefficient query because it must scan all the partitions that make up the table.

Note: For more information about Windows Azure table storage, including partition keys and row keys, see Chapter 5, “Phase 2: Automating Deployment and Using Windows Azure Storage,” of the book, *Moving Applications to the Cloud*. It is available on MSDN (<http://msdn.microsoft.com/en-us/library/ff728592.aspx>).

The following table shows the structure of the user device table in Windows Azure table storage that records which physical devices, identified by a unique URI allocated by the MPNS, are associated with each user.

Column	Notes
User name	A string holding the user name. This column is the table’s partition key.
Device URI	A string holding the device’s unique URI that the MPNS uses to push notifications to the device. This column is the table’s row key.

The structure of this table reflects the fact that a user can have more than one device, each with a unique URI for the MPNS to use. Tailspin optimized this table to support queries that look for the list of devices owned by a user.

Note: With the current implementation, it’s possible for a user who owns multiple Windows Phone 7 devices to enter different lists of preferred tenants on the different devices. The data model in the Tailspin Surveys service only supports a single list of preferred tenants for each user, so the user might see unexpected results on some of their devices. Tailspin plans to address this issue in the next version of the application.

Identifying Which Devices to Notify

Figure 5 outlines the process that the Tailspin Surveys service uses to identify which devices to notify when a subscriber adds a new survey.

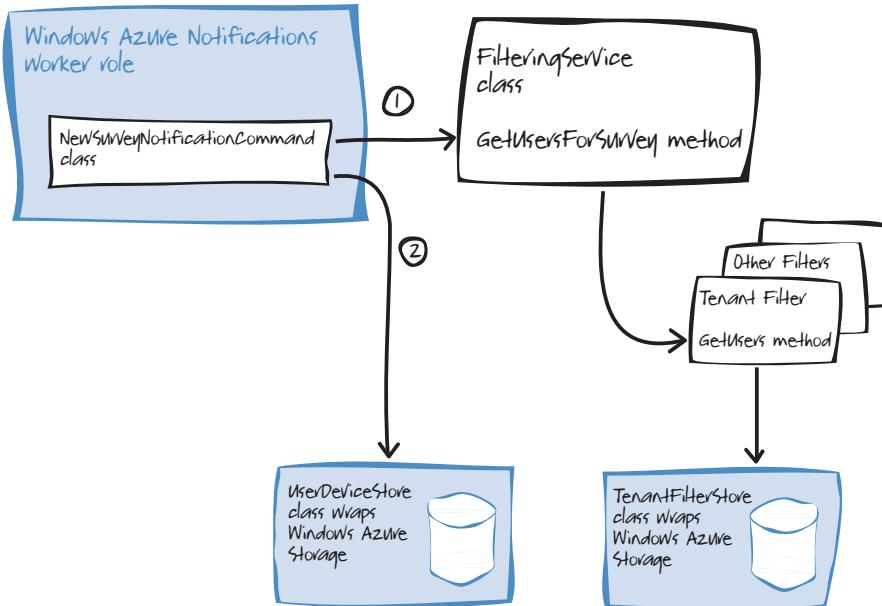


FIGURE 5
Building a list of devices to notify about a new survey

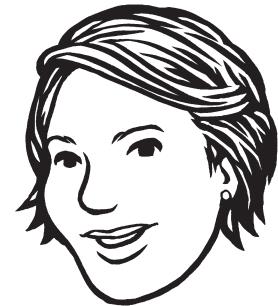
Identifying which devices to notify when a tenant adds a new survey requires two queries. The first query gets the list of users who are interested in that particular survey and this query uses a filtering service to build this list of users based on user preferences stored in Windows Azure table storage.

The second query uses the list of users to find a list of devices to notify, again using data stored in Windows Azure table storage that maps users to devices.

Selecting Surveys to Synchronize

Figure 6 outlines the way that the Tailspin application delivers new, relevant surveys to the mobile client application when it synchronizes with the Tailspin Surveys service. The user initiates the synchronization process from the `SurveyListView` page, and the view model invokes a method in a service class to retrieve the list of new surveys. In the Tailspin Surveys service, the application gets the list of surveys to return to the client from a filtering service class.

In the current version of the application, obtaining the list of users who are interested in a particular survey uses just a single criterion: the user's list of preferred tenants. In the future, the criteria may become more complex if, for example, users can express preferences for location, survey length, or some other attribute of the survey. The developers at Tailspin have implemented an extensible filtering mechanism to accommodate any future additions to the list of supported criteria.



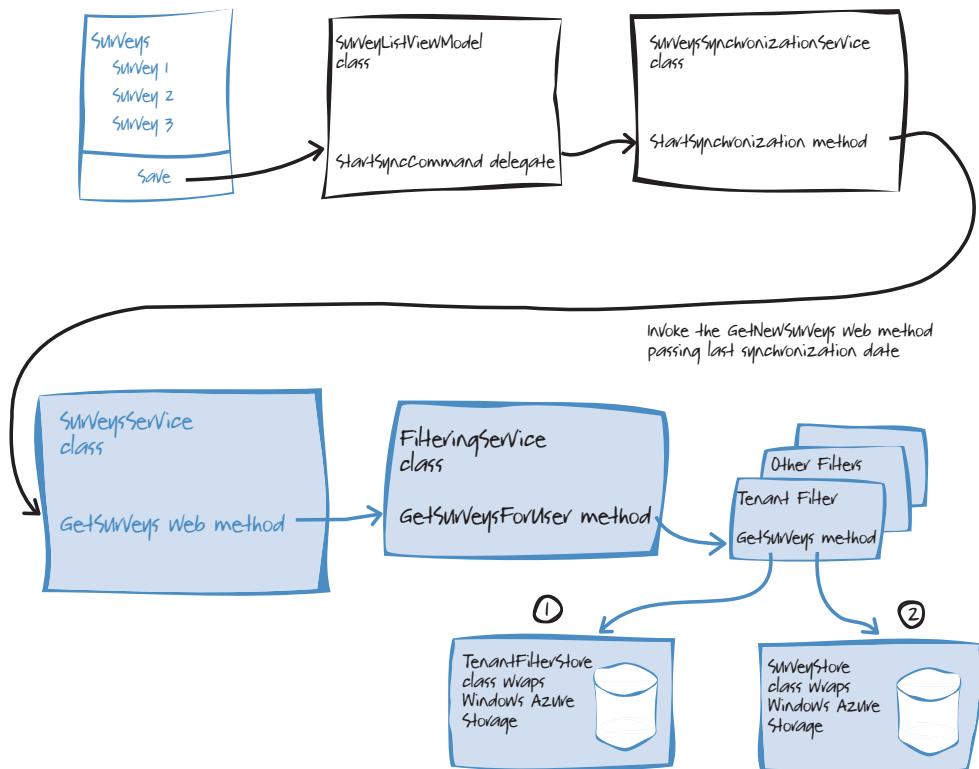


FIGURE 6
Building a list of new surveys for a user

Identifying which surveys to download when a user synchronizes from the mobile client requires two queries. The first query returns a list of tenants based on user preferences stored in Windows Azure table storage. In the current version of the application, these preferences are lists of tenants that the user is interested in. The second query, which is inside the filter, uses the list of tenants to return a list of new surveys from those tenants.

INSIDE THE IMPLEMENTATION

Now is a good time to walk through the code that implements the filtering of the list of surveys in more detail. As you go through this section, you may want to download the Visual Studio solution for the Tailspin Surveys application from CodePlex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

Storing Filter Data

The Tailspin Surveys service stores filter data in Windows Azure table storage. It uses one table to store the tenant filter data, and one table to store the device filter data. The following code example shows how the **TenantFilterStore** class saves tenant filter data in the Windows Azure table storage.

```
C#
public class TenantFilterStore : ITenantFilterStore
{
    private readonly IAzureTable<TenantFilterRow>
        tenantFilterTable;
    ...
    public void SetTenants(string username,
        IEnumerable<string> tenants)
    {
        var rowsToDelete =
            (from r in this.tenantFilterTable.Query
             where r.RowKey == username
             select r).ToList();

        this.tenantFilterTable.Delete(rowsToDelete);

        var rowsToAdd = tenants.Select(t => new TenantFilterRow
        { PartitionKey = t, RowKey = username });
        this.tenantFilterTable.Add(rowsToAdd);
    }
    ...
}
```

The following code example shows how the **UserDeviceStore** class saves device data.

```
C#
public class UserDeviceStore : IUserDeviceStore
{
    private readonly IAzureTable<UserDeviceRow> userDeviceTable;
    ...
    public void SetUserDevice(string username, Uri deviceUri)
    {
        this.RemoveUserDevice(deviceUri);

        var encodedUri = Convert.ToBase64String(
            Encoding.UTF8.GetBytes(deviceUri.ToString()));
        this.userDeviceTable.Add(
            new UserDeviceRow
```



The application removes any existing data with the same key before adding the new rows.

```

    {
        PartitionKey = username,
        RowKey = encodedUri
    });
}

public void RemoveUserDevice(Uri deviceUri)
{
    var encodedUri = Convert.ToBase64String(
        Encoding.UTF8.GetBytes(deviceUri.ToString()));
    var row =
        (from r in this.userDeviceTable.Query
         where r.RowKey == encodedUri
         select r).SingleOrDefault();

    if (row != null)
    {
        this.userDeviceTable.Delete(row);
    }
}
...
}

```

The following code example shows how the **SetFilters** web method in the **RegistrationService** class stores the list of preferred tenants sent from the mobile client.

```

C#
public void SetFilters(SurveyFiltersDto surveyFiltersDto)
{
    var username = Thread.CurrentPrincipal.Identity.Name;

    this.tenantFilterStore.SetTenants(
        username, surveyFiltersDto.Tenants.Select(t => t.Key));
}

```

Note: *There is also a **GetFilters** web method to return the current list of preferred tenants to the client. The phone does not save the filter information locally; instead, it retrieves it from the web service using the **GetFilters** method when it needs it.*

Building a List of Devices to Receive Notifications

Whenever a tenant creates a new survey, the Tailspin Surveys service must send notifications to all the devices that are interested in that survey. The criteria that the service uses to select the devices may change in future versions of the application, so the developers at Tailspin implemented an extensible filtering method to build the list.

The following code example from the **Run** method in the **NewSurveyNotificationCommand** class retrieves a list of device URLs to which it will send notifications.

The **NewSurveyNotificationCommand** class implements a command that a Windows Azure worker role executes in response to receiving a message on a Windows Azure queue.

C#

```
var deviceUrIs =
    from user in this.filteringService.GetUsersForSurvey(survey)
    from deviceUri in this.userDeviceStore.GetDevices(user)
    select deviceUri;
```

This code uses the **FilteringService** class to get a list of users who are interested in the new survey, and then it uses the **GetDevices** method in the **UserDeviceStore** class to get a list of device URLs for those users. The application populates the user device store when the mobile client registers for notifications with the MPNS. For more information, see the section, “Notifying the Mobile Client of New Surveys,” earlier in this chapter.

The following code example shows how the **FilteringService** class uses one or more **ISurveyFilter** instances in the **GetUsersForSurvey** method to filter the list of users.

C#

```
private readonly ISurveyFilter[] filters;
...
public IEnumerable<string> GetUsersForSurvey(Survey survey)
{
    return
        (from filter in this.filters
        from user in filter.GetUsers(survey)
        select user).Distinct();
}
```

The current version of the application uses a single filter class named **TenantFilter** to retrieve a list of users from the tenant filter store based on the tenant name. The following code example shows part of this **TenantFilter** class.

```
C#
public class TenantFilter : ISurveyFilter
{
    ...
    private readonly ITenantFilterStore tenantFilterStore;
    ...
    public IEnumerable<string> GetUsers(Survey survey)
    {
        return this.tenantFilterStore.GetUsers(survey.Tenant);
    }
}
```

Building a List of Surveys to Synchronize with the Mobile Client

Whenever a user of the mobile client application initiates the synchronization process, the Tailspin Surveys service must build a list of new surveys that the user is interested in. The current version of the application uses the user's list of preferred tenants to select the list of surveys, but Tailspin may expand the set of selection criteria in the future. The mobile client invokes the **GetSurveys** web method in **SurveysService** class to get a list of new surveys. The **GetSurveys** method uses the **GetSurveysForUser** method of the **FilteringService** class to get the list of surveys that it will return to the client.

The following code example shows how the **FilteringService** class gets the list of surveys for a user.

```
C#
private readonly ISurveyFilter[] filters;
...
public IEnumerable<Survey> GetSurveysForUser(
    string username, DateTime fromDate)
{
    return
        (from filter in this.filters
         from survey in filter.GetSurveys(username, fromDate)
         select survey).Distinct(new SurveysComparer());
}
```

This method can use one or more **ISurveyFilter** objects to filter the list of surveys. The current version of the application uses a single filter named **TenantFilter**. The following code example shows the **GetSurveys** method of the **TenantFilter** class that first retrieves a list of tenants for the user, and then it retrieves a list of surveys for those tenants.

```
C#
public IEnumerable<Survey> GetSurveys(
    string username, DateTime fromDate)
{
    var tenants = this.tenantFilterStore.GetTenants(username);
    if (tenants.Count() == 0)
    {
        tenants = this.tenantStore.GetTenants().Select(t =>
            t.Name.ToLowerInvariant());
    }

    return
        (from tenant in this.tenantFilterStore.GetTenants(username)
         from survey in this.surveyStore.GetSurveys(
             tenant, fromDate)
         select survey).Distinct(new SurveysComparer());
}
```

Summary

This chapter described the ways in which the Tailspin Surveys mobile client application accesses remote services; it also discussed designing services that the Windows Phone 7 can access. You have now seen a complete picture of the Tailspin Surveys mobile client, including the application's UI, Tailspin's use of the MVVM pattern, how the application leverages services offered by the Windows Phone 7 platform, and how it consumes services over the network.

Questions

1. What do you need to do to make a website mobile-friendly?
 - a. Configure your site to identify requests from mobile browsers and to return pages optimized for mobile web browsers.
 - b. Create web pages using the Wireless Markup Language (WML).
 - c. Minimize the page size as far as possible.
 - d. Don't use JavaScript.

2. How does Tailspin pass authentication requests to the web service?
 - a. Tailspin uses basic authentication with the credentials in an authorization header.
 - b. Tailspin uses Window Live® ID.
 - c. Tailspin uses OAuth.
 - d. Tailspin uses the Windows Identity Framework (WIF).
3. What notification methods does the Microsoft Push Notification Service support?
 - a. Toast notifications.
 - b. Tile notifications.
 - c. SMS notifications.
 - d. Raw notifications.
4. Why does the client need to register with MPNS before it can receive notifications?
 - a. Because MPNS requires clients to authenticate before it will send notifications.
 - b. Because MPNS can then notify your service that the client is ready to receive notifications.
 - c. Because the client must obtain a unique URI to send to your service.
 - d. Because the free version of MPNS has a limit on the number of clients that can receive notifications from your service.
5. How does Tailspin transport data between the client and the web service?
 - a. Tailspin uses the Microsoft Sync Framework to handle the data transport.
 - b. Tailspin uses the WCF Data Services framework.
 - c. Tailspin uses data transfer objects with a WCF REST endpoint.
 - d. The mobile client application uploads directly to Windows Azure BLOB storage.

6. Why does Tailspin filter data on the server and not on the client?
 - a. To minimize the amount of data moved over the network.
 - b. To simplify the application.
 - c. For security reasons.
 - d. To minimize storage requirements on the phone.

More Information

For more information about push notifications, see “Push Notifications for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff402537\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402537(VS.92).aspx)

For more information about developing websites for Windows Phone 7, see “Web Development for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff462082\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff462082(VS.92).aspx)

For more information about security for Windows Phone 7, see “Security for Windows Phone” on MSDN: [http://msdn.microsoft.com/en-us/library/ff402533\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402533(VS.92).aspx)

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

To distribute your applications, you must package them and upload them to the Microsoft® Windows® Marketplace. For your application to be accepted, it must meet the Windows Marketplace certification requirements that are discussed later in this chapter. You can specify the price if you want to sell your application, or you can offer it as a free download. It will appear in the Windows Marketplace hub on users' phones when they search for new applications. You can also use the Marketplace Trial API if you want to enable a reduced feature set and allow users to download and try the application before they purchase it. The applications can be installed on the phone using a wired or Wi-Fi connection, or over the air.

You can also incorporate targeted advertising into your applications by using the Microsoft Advertising Exchange for Mobile hub. Applications can interact with Microsoft Advertising service through a special API, providing a simple way to earn additional income by including advertising within your applications.

This chapter describes the typical application development and publishing life cycle, the application certification requirements, how you package the application, the Windows Marketplace submission and validation process, and how you can access Windows Marketplace and the Microsoft Advertising Exchange for Mobile hub from within your applications.

Note: Windows Marketplace will also allow you to deploy applications to a limited audience, such as a defined set of testers or reviewers, or to employees or associates of your company.

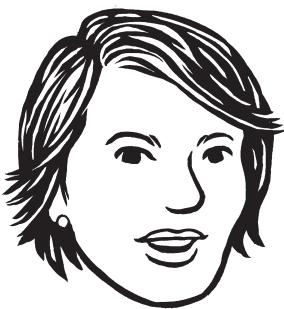
The Application Development and Publishing Life Cycle

When you create an application for Windows Phone 7, you can publish it on Windows Marketplace to make it available to users. The overall process involves interaction with both the Windows Phone 7 developer portal and the Window Marketplace portal. Figure 1 shows a



Windows Marketplace provides the central hub for users to find and install applications for their phone.

high-level view of the stages of building and publishing an application for Windows Phone 7.



The App Hub (<http://create.msdn.com/>) provides all the tools and publishing facilities required for your applications.

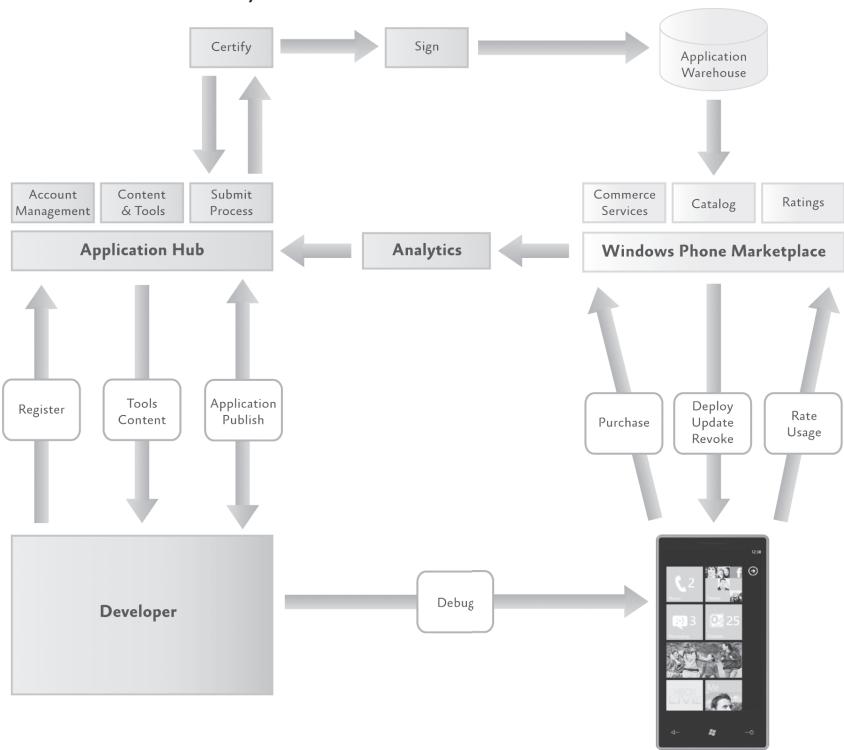


FIGURE 1
The development and publishing life cycle for Windows Phone 7 applications

After you create, test, and debug your application using the tools available from the App Hub, you can register as a vendor and upload your application to the App Hub. As part of this process, you must create an account and verify your identity to obtain a publisher certificate.

After you complete the registration process, you can submit your application. After it passes the validation and certification stages, it is signed with your publisher certificate and published to the Windows Marketplace application warehouse.

The Marketplace hub on the phone allows users to purchase your application, download it, and install it on their phones. As mentioned earlier, you can also create and publish trial versions of your application. The App Hub provides analysis of downloads, sales information, and user ratings. In addition, you can manage your application and update it (by deploying a new version) as required.

Note: Users can also find and buy your Windows Phone 7 application by using Zune software on their computers. For more information about finding and buying Windows Phone 7 applications, see "Find and buy apps in Marketplace" (<http://www.microsoft.com/windowsphone/en-us/howto/wp7/apps/find-and-buy-apps-in-marketplace.aspx>).

The following sections of this chapter describe the requirements and processes for publishing your applications on Windows Phone Marketplace. For more information about the development platform and application life cycle, see the previous chapters of this book and "Application Platform Overview for Windows Phone" on MSDN® ([http://msdn.microsoft.com/en-us/library/ff402531\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402531(VS.92).aspx)).

For more information about creating trial versions of your applications, see Appendix A, "Tools, Frameworks, and Processes," in this book and "Creating Trial Applications for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff967554\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff967554(VS.92).aspx)).

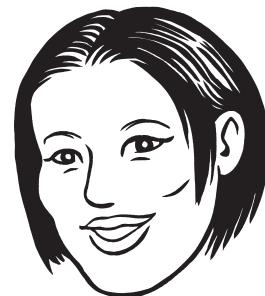
The home page for App Hub is <http://create.msdn.com/>.

Application Certification Requirements

To ensure the quality and usability of applications distributed through Windows Marketplace, every application must pass through a validation process after submission and before it is made available for download. The following are the core tenets of this validation process:

- **Applications must be reliable.** This is reflected by the implementation of published best practices for creating reliable applications.
- **Applications must use resources efficiently.** They must execute efficiently on the phone and not have a negative impact on performance.
- **Applications must not interfere with the functionality of the phone.** They must not modify settings, or they must prompt the user before changing any user preferences or the behavior of other applications.
- **Applications are free from viruses and malicious software.** They must be safe for users to install and run.

In addition, applications must meet several other requirements. These include complying with the Microsoft policies for acceptable content and ensuring that localized metadata and content are accurately represented for the application's supported geographic regions. The full set of requirements falls into four main categories: application code restrictions; run-time behavior, performance and metrics; user



Your Windows Phone 7 applications must conform to a series of requirements before they can be made available on Windows Marketplace.



opt-in and privacy; and media and visual content. The following sections provide an overview of the requirements within each of these categories.

For full documentation of all the requirements, download *Windows Phone 7 Application Certification Requirements* from the Microsoft Download Center (<http://go.microsoft.com/?linkid=9730556>).

APPLICATION CODE RESTRICTIONS

This section describes the requirements for your Windows Phone 7 applications in terms of the code you write, the way the application works, and the way that you compile it.

Your application must run on any Windows Phone 7 device, regardless of model, screen size, keyboard hardware, or manufacturer.

The application must use only the Windows Phone Application Platform documented APIs, and it must not use **PIInvoke** or COM interoperability. If you use controls from the **System.Windows.Controls** namespace, you must not call any APIs in the **Microsoft.Xna.Framework.Game** or **Microsoft.Xna.Framework.Graphics** assemblies.

The application runs in a sandbox on the device. Application code must be type-safe when compiled to Microsoft intermediate language (MSIL) code. You cannot use unsafe code or security critical code. You also cannot redistribute the Windows Phone assemblies; however, you can redistribute the **Panorama**, **Pivot**, and **Map** control assemblies if these are used in your application.

The application must also handle all exceptions raised by the .NET Framework, must not terminate unexpectedly or become unresponsive to user input, and it must display user-friendly error messages. A visual progress indicator must be displayed when executing time-consuming activities, such as downloading data over network connections, and you must provide an option to cancel this type of activity.

When you compile your application for distribution, you must do so using the Release or Retail configuration. It must not be compiled using the debug configuration, and it must not contain debugging symbols or output. However, you can submit an application with obfuscated code if you want.

Finally, the application must not use reflection to discover and invoke types that require security permissions. The repackaging process that takes place before distribution does not detect invoking types in this way, so the generated manifest will not request the required permissions. This will cause the application to fail at run time.

RUN-TIME BEHAVIOR, PERFORMANCE, AND METRICS

To maintain a consistent user experience, the Back button must only be used for backward navigation in the application. Pressing the Back button must return the application to the previous page, or—if the user is already on the initial page—must close the application. If the current page displays a context menu or a dialog box, pressing the Back button must close the menu or dialog box and cancel the backward navigation to the previous page. The only exceptions to this are in games, where the Back button may display a context menu or dialog box, navigate to the previous menu screen, or close the menu.

The certification requirements for Windows Phone 7 applications specify both the startup performance of applications and some limitations on the use of resources. For example, the application must render the main user interface (UI) startup screen (not the splash screen) within five seconds on a typical physical device, and it must be responsive to user input within twenty seconds. The application must also complete execution in response to both the **Activated** and **Deactivated** events within ten seconds or it will be terminated by the operating system.

In addition, the application must not use more than 90 MB of RAM memory on the device unless it first queries the properties of the **DeviceExtendedProperties** class, and this class reports that more than 256 MB of memory is installed. The application must be able to run whether or not this extended memory is available, and it should vary its behavior only at run time to take advantage of additional memory, if required.

Your application should, as far as is appropriate, minimize its use of hardware resources, services, and device capabilities to preserve battery power. For example, it should use the location service and the push notification service only when required (which also minimizes the load on these Microsoft services).

An application in the foreground can continue to run when the phone screen is locked by setting the **PhoneApplicationService.ApplicationIdleDetectionMode** property (you must prompt the user to allow this). When an application is running under a locked screen, it must minimize power usage as much as possible.

USER OPT-IN AND PRIVACY

Your application must maintain user privacy. In particular, it must protect data provided by the user and must not pass any user-identifiable or personal information to another service or application unless you provide full details of how the information will be used and the user agrees to this (the opt-in approach).



Your application must start up and be responsive within 20 seconds, and use no more than 90 MB of memory unless the device has extended memory installed.



Many types of applications will use features that require user opt-in.

You must not rely on the push notification service for delivering critical messages of any kind because timeliness of delivery—or delivery at all—cannot be guaranteed.

If the application includes person-to-person communication such as chat features, and these features can be set up from the device, you must take steps to ascertain that the user is at least 13 years old before enabling this feature.

You must also obtain the user's consent before your application uses the location service to identify the user's location or if it uses the Microsoft Push Notification Service (MPNS) to send notifications to the user's phone. For example, the application must ask the user for explicit permission to receive a toast notification before it first calls the **BindtoShellToast** method for the first time.

For information about the different types of notifications, see "Types of Push Notifications for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff941124\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941124(VS.92).aspx)).

If your application relies on the location feature and passes the location information to another application or service (such as a social community site), you must periodically remind users by displaying a message or including a visual indication on screen that location data is being sent to another service.

Users must also be able to turn off the location feature and push notifications within the application, and the application must either continue to work or present the user with a suitable message if it cannot work with the feature turned off. It must not simply fail when the feature is disabled.

An application in the foreground can continue to run when the phone screen is locked by setting the **ApplicationIdleDetection Mode** property. If you design your application with this capability, you must first obtain consent from the user to run in this mode and provide them with an option to disable it.

Your application must also ask users for permission before performing large downloads. Typically, this means asking for permission for data downloads greater than 50 MB that the application requires or for the use of services or communication protocols that may incur excessive communication costs.

APPLICATION MEDIA AND VISUAL CONTENT

The visual appearance and content of an application must conform to a series of requirements. For example, in terms of styling, the UI must be readable and usable if the user changes the color scheme from the default white on black.

Windows Phone 7 applications and Windows Marketplace initially support localization into only English, French, Italian, German, and Spanish. However, it is likely that other languages will be added over time. All Windows Marketplace materials and application content must be correctly localized into the chosen language. For exam-

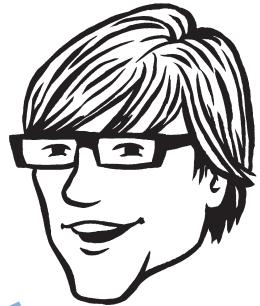
ple, an application that is submitted in French must have a product description, UI text, screen shots, and icons in French. You can submit an application for multiple languages as a single package. If you submit separate language versions, each is validated and certified separately.

For information about the localization features in Windows Phone 7, see “Globalization and Localization Overview for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff462083\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff462083(VS.92).aspx)).

Your applications must also abide by a series of restrictions on the content and links to content located elsewhere. These restrictions include the appropriate use of licensed content and trademarks; they also include limitations based on preventing illegal content such as libel and slander, threatening behavior, violence, pornography, and discrimination. There are very strict limits that prevent an application from containing content related to hate speech and defamation; the use or promotion of alcohol, tobacco, weapons, and drugs; certain adult-related content; and excessive profanity.

Finally, there are a series of restrictions that apply specifically to music-related or photo-related applications, and to advertising. These specify how users must be able to launch your application, and the types of advertising content that is acceptable. Although your applications can contain most types of advertising, they cannot, for example, promote other phone service plans or phone application marketplaces.

However, your application can play music in the background, even when its primary function is not related to music or video experiences.



Localization of an application includes all the resources you upload to Windows Marketplace.



Packaging the Application

You must package your application as a XAP file to be able to upload it to Windows Marketplace. The Windows Phone Tools for Visual Studio and the special versions of Microsoft Visual Studio® Express and Microsoft Expression Blend® design software included in the Windows Phone Tools automatically create a XAP file that you can upload.

A XAP file contains a series of artifacts that are required to install the application on a Windows Phone. These include the following:

- A valid Windows Phone application manifest file named WMAppManifest.xml, with the application title contained in the **<Title>** element, and including a list of permission demands for device capabilities
- A valid .NET Framework application manifest file named AppManifest.xaml that lists the assemblies used by the application

Applications for Windows Phone 7 are distributed as XAP files that you can create using the Windows Phone Tools.



XAP packages larger than 20 MB cannot be downloaded over the air using a phone connection.

- The assembly (DLL) files specified in the AppManifest.xaml file
- Any user controls or XAML files required by the application
- The resources, icons, images, media, and other content files that the application requires; the application icon must be a 62-by-62 pixel PNG file, and the application tile image used on the Start screen must be a 173-by-173 pixel PNG file.

The maximum size of the XAP package file you can upload is 400 MB, including all the media or other content that is included in the XAP file but not compiled into the application as resources.

If the package to download to the phone exceeds 20 MB in size, users will not be able to download it from Windows Marketplace when using a GPRS or 3G phone connection. Packages over 20 MB in size must be downloaded and installed over a Wi-Fi or physical wired connection.

If your application requires a separate download in order to work, such as an additional data package, and if the separate download exceeds 50 MB, you must inform the user and obtain permission to download it.

THE WINDOWS MARKETPLACE REPACKAGING PROCESS

The validation process, which the Windows Marketplace applies to all submitted applications, automatically unpacks and then repackages the application. It unpacks all the contents of the submitted XAP file and performs a range of validation checks on these. Afterward, it creates a new WMAppManifest.xml file based on the results of the validation process and repackages all the original content with the new WMAppManifest.xml into the XAP package that users will download.

The updated WMAppManifest.xml file contains a product identifier that is used in Windows Marketplace to uniquely identify the application download package; it also includes an indicator of the application type (such as a music and videos hub application). However, the main feature of the validation process that it is important for developers to understand is the way that the security permission demands are applied in the new WMAppManifest.xml file.

By default, the WMAppManifest.xml file created by the Windows Phone Tools templates in Visual Studio and Expression Blend contains permission demands for all the device capabilities. These include networking, location, microphone, push notifications, and more. However, applications distributed through Windows Marketplace advertise the capabilities they require, and users can be sure that an application that does not indicate it uses (for example) the location service, actually will not use it internally without the user being aware.

To ensure that this is the case, the validation process searches the application code for calls to any of the device capabilities that require permission. If the way that these are used in the application meets the certification requirements, the validation process includes that permission demand in the WMAppManifest.xml file and the product metadata displayed by Windows Marketplace. If a feature is not used, the permission demand is removed from the WMAppManifest.xml file. Therefore, code can use only the features for which it is certified, and any attempt to use other features (such as using reflection to invoke a feature) will fail because the application does not have the relevant permission.

You can perform the same detection process as is carried out by the submission validation process on your own computer by using the Windows Phone Capability Detection Tool to check that the correct capabilities are detected.

The Windows Phone Capability Detection Tool (CapabilityDetection.exe) is installed by default in the %ProgramFiles%\Microsoft SDKs\Windows Phone\v7.0\Tools\CapDetect folder. For information on using this tool, see "How to: Use the Windows Phone Capability Detection Tool" on MSDN ([http://msdn.microsoft.com/en-gb/library/gg180730\(v=VS.92\).aspx](http://msdn.microsoft.com/en-gb/library/gg180730(v=VS.92).aspx)).

The Windows Phone Capability Detection Tool is part of the October 2010 update to the Windows Phone Tools, which is available from the Microsoft Download Center (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=49B9DoC5-6597-4313-912A-F0CCA9C7D277&displaylang=en>).

After the validation process completes, the XAP file is rebuilt, including all the original content plus a file named WMAppPRHeader.xml that contains the Digital Rights Management (DRM) information. The DRM information controls how and where the application can be used, and it manages the trial period where applicable.

Summary of the Submission and Validation Process

The following steps provide an overview of how you can submit an application to Windows Marketplace and the validation and certification processes that occur:

1. Go to the App Hub website (<http://create.msdn.com/>) and click **Sign In** to log on to App Hub. You must have a Windows Live® ID to log on.
2. If you do not already have an App Hub account, you must register, sign the application provider agreement, select your account type (different accounts are available for businesses,



Using reflection to invoke device capabilities is likely to cause your application to fail at run time because it will not have the security permissions required to run properly. Therefore, it will fail certification.

students, Microsoft Partner Program members, and individuals) and provide the details required for your account.

Information, such as a tax identification number, is required to establish a vendor account. For companies, this information should come from an authorized officer of your company. Non-U.S. residents and corporations must obtain a U.S. non-resident tax payer ID (ITIN) to submit to Microsoft, together with a local country tax registration number.

3. Finally, you must pay the annual subscription fee, and then wait until your account is validated and approved.
4. The validation and approval process has three stages. You will receive an email to validate your email address. You will also receive an email from the company, GeoTrust that validates applicants. The response to this email must come from an officer of the company if you are registering a company. Individuals will need to validate themselves to GeoTrust. The third stage is to complete the vetting process by submitting the required documents to GeoTrust before you are issued a publication certificate.
5. After the validation process is complete, your account is active, and you have a publication certificate, you can go back to the portal and upload your application XAP file. You must pay a fee for each application you publish. You must enter the required metadata for the application, such as the description, category, support email address, and iconography. You must also specify the distribution countries and pricing information for your application. You can upload screen shots of your application for use in the Windows Marketplace catalog.
6. The application XAP file is validated and, providing that validation succeeds, you can specify that it is published as soon as the certification process is complete, or it can be held unpublished until you decide to publish it at a later time. If it fails validation, you can view the test results to see why it failed.
7. The certification process for the application begins. The XAP file is unpacked and the content is checked to ensure it complies with all publishing requirements. These include both automated and manual technical testing, policy checking, and market validation. If the application meets all the requirements, it is repackaged and signed with a Windows Marketplace certificate. You can visit the Dashboard page to see a

list of your applications, together with their current validation and publication status.

8. You cannot update existing applications on Windows Marketplace. If you want to update a published application, you must resubmit it as a new product.

For a more detailed description of the registration process, see "App Hub Registration Walkthrough" on App Hub (http://create.msdn.com/en-US/home/about/developer_registration_walkthrough).

For a more detailed description of the submission process, see "App Hub Application Submission Walkthrough" on App Hub (http://create.msdn.com/en-US/home/about/app_submission_walkthrough).

For useful frequently asked questions (FAQ) for App Hub that include prices and estimates of how long the certification process takes, see the App Hub website (<http://create.msdn.com/en-us/home/faq>).



Displaying Advertisements in an Application

One way that you may consider increasing revenue from your Windows Phone 7 applications is to display advertisements within some or all of the pages. An easy way to accomplish this is to download and install the Microsoft Advertising SDK for Mobile on your development computer and insert the Ad Control (provided as a separate assembly within the download) into your pages.

The Ad Control calls the Microsoft Ad Exchange to fetch an advertisement each time it is displayed. Typically, this is a small image that displays above the application bar in the phone and does not dramatically decrease the space available to your application UI. When a user clicks the advertisement, either a page on the advertiser's web site is displayed in a browser within the context of your application, or the Ad Control can initiate a phone call to the advertiser. You will receive payment from Microsoft based on the number of impressions displayed.

Microsoft Ad Exchange is an exchange where advertisers can bid for advertising slots on Windows Phone 7 devices. It automatically selects a suitable advertisement to display based on the properties you specify for the Ad Control and the advertiser's profile and other parameters within Microsoft Ad Exchange. You can specify values that influence the way that the exchange selects an advertisement. These properties include the type of application, keywords you provide, demographic information, if available (for example, if you collect this in your application through a registration process), and current location data if you collect this as the application executes and update it in the Ad Control properties.

You can earn money by displaying advertisements within your applications.



Your applications can interact with the Windows Marketplace hub on the phone.

For more information about the Mobile Advertising SDK for Windows Phone 7 and Microsoft Advertising Exchange for Mobile, see “Monetize your Windows Phone 7 Apps” on the Microsoft Advertising website (<http://advertising.microsoft.com/mobile-apps>).

Accessing Windows Marketplace within an Application

It is possible to access Windows Marketplace using code within a Windows Phone 7 application. This is a useful way to offer users of your application other applications that you publish, or to help them find upgrades and new versions of the application.

The Windows Phone 7 operating system includes an API that allows your application to open the Windows Marketplace hub on the phone to show specific types of content, such as applications, music, or podcasts. You can also open the hub showing a filtered list of items from one of these categories using a search string, or it can show just a specific item if you specify its CUID content identifier. Finally, you can open the Reviews screen. You can also include a direct link to a specific product on Windows Marketplace in non-Windows Phone 7 applications (such as websites and desktop applications).

For information about using these features, see the section, “Windows Marketplace,” in Appendix C, “Leveraging Device Capabilities.”

All links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Questions

1. Which of the following are valid ways that you can distribute your Windows Phone 7 applications?
 - a. Publish it as a full product for sale through Windows Marketplace.
 - b. Create an installer and sell it on a CD-ROM.
 - c. Create a trial version and offer it free to users through Windows Marketplace.
 - d. Plug in each phone to the USB port and copy it directly to the phone.
2. Which of the following services are available through the Windows Phone Developer Portal?
 - a. Developer registration service.
 - b. Tools for building Windows Phone 7 applications.

- c. Windows Marketplace account management.
 - d. Application submission service for your applications.
3. Which of the following are **not** requirements for obtaining certification for an application?
- a. Applications must be free from viruses and malicious software.
 - b. Applications must be reliable.
 - c. Applications must allow users to select music files on their phone.
 - d. Applications must minimize use of resources.
4. Which of the following programming techniques are permitted in an application?
- a. The application can terminate if the phone does not have extended memory installed.
 - b. The code can use PInvoke or COM interop to execute native code.
 - c. The application can use code that is not type safe if this is absolutely necessary.
 - d. The application can use reflection to discover and invoke types that require security permissions.
5. For which of the following functions must the application prompt the user to obtain consent before using that feature?
- a. The location service.
 - b. Notifications.
 - c. Person-to-person communication, such as chat features.
 - d. Downloading additional content or data.
6. Which of the following tasks are executed during the application submission process?
- a. Unpacking and validating the content of the XAP file.
 - b. Recreating the permissions demands in the manifest file.
 - c. Removing any content that does not conform to content-restriction policies.
 - d. Adding a Digital Rights Management (DRM) header.

Appendix A

Tools, Frameworks, and Processes

This appendix describes the development and testing environment, tools, and techniques for building Windows® Phone 7 applications. It covers the following topics:

- Setting up a development environment for Windows Phone 7
- Using a hardware device during development
- Developing Windows Phone 7 applications
- Debugging Windows Phone 7 applications
- Unit testing Windows Phone 7 applications
- Additional tools and frameworks

Setting Up a Development Environment for Windows Phone 7

You can use Microsoft® Visual Studio® development system and the Windows Phone Developer Tools to simplify application development for Windows Phone 7. The Windows Phone Developer Tools include the following:

- Visual Studio 2010 Express for Windows Phone
- Integration Components and Templates for Visual Studio 2010
- Windows Phone Emulator
- Microsoft Silverlight® 4 Tools for Visual Studio
- Microsoft Expression Blend® design software for Windows Phone
- XNA® Game Studio 4.0

The tools install Silverlight, and you may see an error message during installation if you have a patched version of Silverlight already installed. You can uninstall Silverlight before you install the tools to avoid this, or you can just ignore this message.

If you are installing the tools on the Windows Vista® operating system with Service Pack 2, you must ensure that the following Windows updates that are a part of Knowledge Base article 971644,

"Description of the Platform Update for Windows Server 2008 and the Platform Update for Windows Vista," have been applied. The following are the updates:

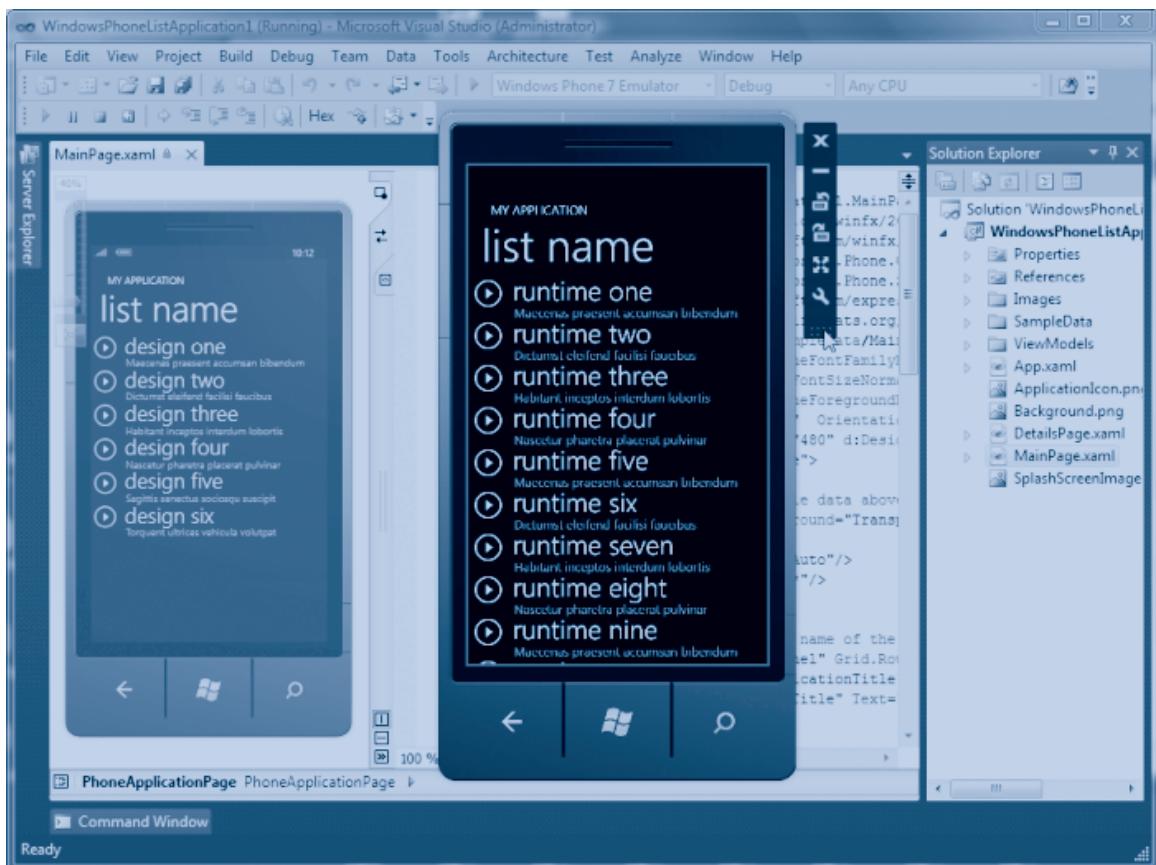
- Update for Windows Vista (Knowledge Base article 971512)
- Update for Windows Vista (Knowledge Base article 971513)
- Update for Windows Vista (Knowledge Base article 971514)
- Update for Windows Vista (Knowledge Base article 960362)

Note: *When running the emulator on Windows Vista, you may experience some audio glitches due to lost audio packets when playing audio, such as a media file or alarm.*

You can obtain the Windows Phone Developer Tools from the App Hub website (<http://create.msdn.com/en-us/home/getting-started>).

Note: *You will also need to install the Windows Phone Developer Tools October 2010 Update from the Microsoft Download Center (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=49B9DoC5-6597-4313-912A-F0CCA9C7D277&displaylang=en>).*

Figure 1 illustrates Visual Studio 2010 Solution Explorer with the Windows Phone Developer Tools installed.

**FIGURE 1**

Visual Studio 2010 with the Windows Phone Developer Tools and emulator

If you have a touch-sensitive screen on your development computer, you can use this to operate the emulator. You can also install additional input devices, such as a tablet or an extra mouse, to test functions such as zoom that require “two finger” operations when you do not have access to a development computer that supports touch.

Note: *The emulator runs as a virtual machine, so you cannot use the Windows Phone Developer Tools within a virtual machine.*

When you run an application, it is packaged as a single XAP file and deployed to the emulator, which runs it automatically.

In addition, some of the capabilities of a Windows Phone 7 device are either not supported or are not fully functional in the emulator. For more information, see “Launcher and Chooser Support in Windows Phone Emulator” on MSDN® ([http://msdn.microsoft.com/en-us/library/ff955600\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff955600(VS.92).aspx)).

EXPRESSION BLEND FOR WINDOWS PHONE

The Windows Phone Developer Tools also include Expression Blend 4 for Windows Phone. This provides an environment more suited to graphical design of the interface and makes implementation of features such as animations, gradient fills, and other attractive effects much easier than in it is in Visual Studio. You can open a project in Expression Blend or Visual Studio and then develop different parts of the project in the most suitable tool.

Figure 2 illustrates a Windows Phone solution open in Expression Blend. You can run it from here. It uses the same emulator as Visual Studio to display the application at run time.

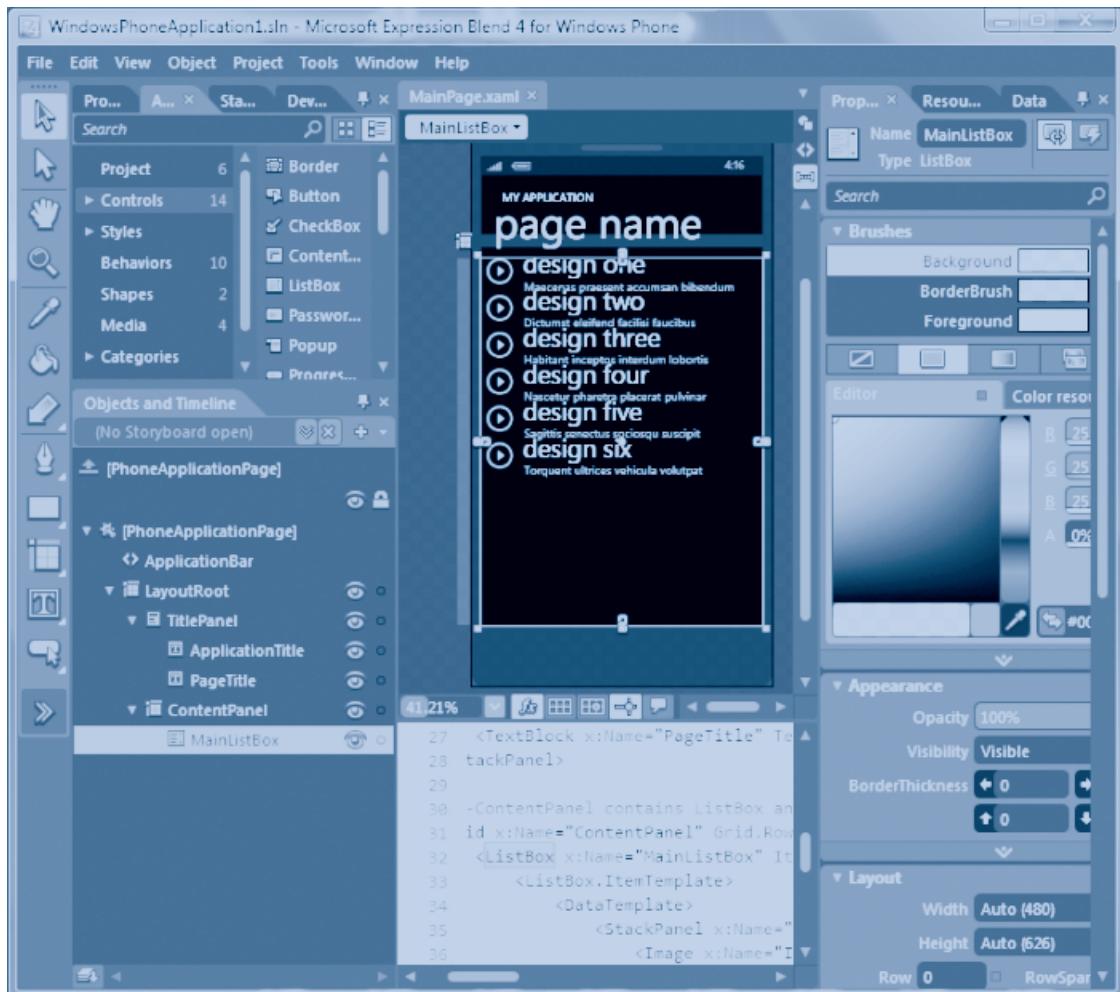


FIGURE 2
Expression Blend 4 for Windows Phone

To see a video presentation that shows the power of Expression Blend for creating a complex interactive user interface (UI), see *Authoring for Windows Phone, Silverlight 4 and WPF 4 with Expression Blend* on the Mix 10 website (<http://live.visitmix.com/MIX10/Sessions/CL02>).

Note: To ensure that projects you create in Visual Studio can be opened in Expression Blend, check that the project file (.csproj) contains a configuration element in the default project properties section that correctly identifies it. You can do this by editing the project file in a text editor. For example, if the project is named *MyTestProject*, you should ensure that the correct **<Configuration>** element is included in the project file, as shown here:

```
<Project ToolsVersion="4.0" DefaultTargets="Build" ... >
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">MyTestProject</Configuration>
    ...
  </PropertyGroup>
  ...

```

ADDITIONAL SILVERLIGHT CONTROLS FOR WINDOWS PHONE 7

You can obtain several additional and useful controls to use in your Windows Phone 7 applications by installing the Silverlight for Windows Phone Toolkit. The toolkit includes the following:

- The **GestureListener** control that allows you to detect and handle the full range of gesture events, including Tap, Double-Tap, Hold, Drag, Flick, and Pinch
- The **DatePicker** and **TimePicker** controls that make it easier for users to enter dates and times; these controls automatically localize to the correct date and time format setting
- The **ContextMenu** control that makes it easy to incorporate shortcut menus in your application
- The **WrapPanel** control that allows you to more easily manage the way content is displayed
- The **ToggleSwitch** control that implements an on/off input metaphor
- Additional UI themes for Windows Phone 7 applications

To obtain the Silverlight for Windows Phone Toolkit, see the Silverlight Toolkit page on CodePlex (<http://silverlight.codeplex.com/>).

Using a Hardware Device during Development

You can develop and debug applications using a physical Windows Phone 7 device; in fact, this is recommended at least as a final step because the emulator does not provide all the functionality of a hardware device and may not portray your application exactly as it will appear and behave on a real phone.

CONNECTING A PHYSICAL DEVICE

To connect a physical Windows Phone 7 device to your development computer, you must install the Microsoft Zune® software. If it was not provided with your device, you can download the Zune client software from the Zune website (<http://go.microsoft.com/fwlink/?linkid=192182>). The Zune client allows you to connect your device using a USB cable or through a wireless connection.

REGISTERING AND UNLOCKING THE DEVICE

Before you can deploy, test, and debug Windows Phone applications on a physical device (a process known as side loading), you must create a developer account at the Microsoft Windows Phone Marketplace that grants you the right to publish and distribute your applications to the phone. If you have not already created an account on App Hub (<http://create.msdn.com/en-us/home/membership>), do so now.

After your account is verified, you must run the registration tool to unlock your phone. This uses the Zune software, which must be installed on your development computer, and the Windows Phone Developer Registration tool that is installed with the Windows Phone Developer Tools. After the registration succeeds, you can deploy applications to the phone. You can also unregister the device if required using the same tool.

Note: *You can install a maximum of 10 applications through side loading on a device. This does not include applications that have been installed through Windows Marketplace.*

If you try to register and unlock a device that was previously registered, you must either rename the device using the Zune software, or go to the Windows Phone 7 portal and unregister the device.

For full details of the registration and unlocking process, see “How to: Use the Developer Registration Tool for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff769508\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff769508(VS.92).aspx)).

DEPLOYING APPLICATIONS TO THE DEVICE

After you connect and register your device, you can deploy an application to it during development by selecting **Windows Phone 7 Device** in the drop-down list in the **Standard** toolbar in Visual Studio or Visual Studio Express.

Applications are deployed as a single package with the .xap file extension that contains all the files required to run the application. Visual Studio creates the XAP package and deploys it to the emulator or physical device when you run the application in the development environment.

After you finish developing and testing your application, you can host it in your Windows Marketplace account ready for download to any Windows Phone 7 device. In the future, you will also be able to distribute internal applications for your organization through a private cloud-based service, although the mechanics of this are not currently determined.

Note: *A XAP package is actually a .zip file, and you can examine the contents of the package by renaming the file with the .zip file name extension and opening it in Windows Explorer. You will see the XML manifest file, the startup Extensible Application Markup Language (XAML) file, the assembly for your application and any other assemblies you added to the project, and the resources (such as images) that the application uses.*

Using the Windows Phone Connect Tool

If you are building media applications that interact with the media APIs, you will not be able to debug these while the Zune software is running. For example, if your application uses Media Launchers or Choosers, the XNA Framework for song playback, or the Silverlight **MediaElement** control to play video or audio content, you must use the Windows Phone Connect Tool to debug your application.

The Windows Phone Connect Tool is part of the October 2010 update to the Windows Phone Tools, which is available from the Microsoft Download Center (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=49B9DoC5-6597-4313-912A-F0CCA9C7D277&displaylang=en>).

The Windows Phone Connect Tool (WPCConnect.exe) is installed by default in the %ProgramFiles%\Microsoft SDKs\Windows Phone\v7.0\Tools\WPCConnect folder. To use the tool:

1. Connect your Windows Phone to your computer and ensure that it is detected by the Zune software.
2. Close the Zune software after your Windows Phone is connected.

3. Open a command prompt and navigate to the folder containing WPConnect.exe.
4. At the command prompt, enter the command WPConnect.exe.

For more information, see “How to: Use the Windows Phone Connect Tool” on MSDN ([http://msdn.microsoft.com/en-gb/library/gg180729\(v=VS.92\).aspx](http://msdn.microsoft.com/en-gb/library/gg180729(v=VS.92).aspx)).

Developing Windows Phone 7 Applications

The Windows Phone Developer Tools install a series of templates into Visual Studio 2010, Visual Studio 2010 Express, and Expression Blend that provide a starting point for developing Windows Phone applications.

To use a Windows Phone 7 template in Visual Studio and Visual Studio Express

1. On the Visual Studio **File** menu, point to **New**, and then click **Project**.
2. In the **Installed Templates** tree view, expand **Visual C#**, and then click **Silverlight for Windows Phone**.
3. In the central pane of the **New Project** dialog box, select the type of application you want to start with. If you want to display a list of items on the start page, select **Windows Phone List Application**. If you want to begin with a simple empty start page, select **Windows Phone Application**. If you are adding a new class library project to an existing Windows Phone 7 application, select **Windows Phone Class Library**. If you want to build an application that contains a **Panorama** or a **Pivot** control for displaying information, select **Windows Phone Panorama Application** or **Windows Phone Pivot Application**.
4. Enter a name and specify the location for the new project, and then click **OK**.

To use a Windows Phone 7 template in Expression Blend for Windows Phone

1. Do one of the following:
 - In the startup dialog box, click **New Project**.
 - On the **File** menu, click **New Project**.
2. If you want to begin with a simple start page, select **Windows Phone Application** in the list of installed templates. If you

want to display a list of data-bound items on the start page, select **Windows Phone Databound Application**. If you want to create a new control project, select **Windows Phone Control Library**. If you want to build an application that contains a **Panorama** or a **Pivot** control for displaying information, select **Windows Phone Panorama Application** or **Windows Phone Pivot Application**.

3. Enter a name and specify the location for the new project, and then click **OK**.

The templates create a complete application with a “start” page that contains a list of sample data items and a “details” page that is displayed when you click an item in the list. You can see this application running in the emulator in Figure 1.

The template applications follow a basic Model-View-ViewModel (MVVM) pattern. The two UI pages are views of the data, although they do contain some code-behind to implement transitions and navigation, and to set the data context of the views. There are two view models, one for each view, that contain the public properties bound to the controls in the views and logic to create the list of items displayed in the views.

You can run the application as it stands to see how it works and to experiment with the emulator. The emulator can be rotated and resized as required. It also contains an implementation of Internet Explorer that you can use to browse the Internet or your own web-based applications.

For information about writing applications for Windows Phone 7, see the following resources:

- “Windows Phone 7 Jump Start Training” on the App Hub (http://create.msdn.com/en-US/education/catalog/article/wp7_jump_start).
- “Getting Started with Windows Phone Development” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402529\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402529(VS.92).aspx)).
- “Windows Phone Development” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402535\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402535(VS.92).aspx)).
- “Application Features for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402551\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402551(VS.92).aspx)).

Note: For information about packaging and submitting your application to Windows Marketplace, see Chapter 7, “Interacting with Windows Marketplace.”



Trial versions are a great way to tempt users to try and subsequently purchase your applications.

DEVELOPING TRIAL APPLICATIONS FOR WINDOWS PHONE 7

Windows Phone 7 allows you to easily publish trial versions of your application that users can install from Windows Marketplace. This is a useful way to demonstrate your application and allow users to decide if it is suitable for their requirements before they buy the full version. The trial version must reasonably represent the functionality and quality of the full application in order to obtain certification.

In general, you should avoid setting a time limit in trial versions so that users can install the trial more than once and will not find that it is blocked if they decide to evaluate it more fully at a later date. Instead, limit the use of some features so that the user can experience the main functionality and will want to buy the application to unlock these additional features.

Windows Phone 7 implements the **IsTrial** method of the **LicenseInformation** class (located in the **Microsoft.Phone.Marketplace** namespace). Your application can call this method to discover whether the user is running a trial version or the full purchased version, and either display appropriate messages or direct the user to the page where they can purchase the full version. You can display the Windows Marketplace page for the current application by calling the **Show** method of the **MarketplaceDetail** class without specifying a content identifier. For more information about the **MarketplaceDetail** class, see Appendix C, “Leveraging Device Capabilities.”

Note: When you use the **IsTrial** method, be aware that this call involves communication with the Windows Marketplace servers. To minimize execution delays and communication cost for the user, call the method once when your application starts and cache the returned value for use throughout execution of the application.

Testing Trial Versions in an Emulator or Device

When you test an application using the emulator or an unlocked physical device, the **IsTrial** method will always return **false**. Instead, you can emulate trial mode using the **Guide** class (located in the **Microsoft.Xna.Framework.GamerServices** namespace). When you set the **SimulateTrialMode** property to **true**, the **IsTrial** method will return **true**. You should query the **IsTrial** method as soon as possible and cache the result because the value may change if the application later detects a valid license.

For more information about implementing trial versions of your applications, see “Creating Trial Applications for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff967554\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff967554(VS.92).aspx)).

DEVELOPING WEB APPLICATIONS FOR WINDOWS PHONE 7

An alternative approach to building applications for Windows Phone 7 is to rely on the built-in Web browser within the operating system of the phone to display web pages, instead of designing and building a phone-based UI and using client-side code. This does not provide the same user experience, but it may be a useful approach when you already have a suitable application that you want to expose to phone users. You can also use a combination of the Web browser and a custom phone application. For example, you may consume services such as the Windows Live® log-on page to allow users to log on to a service and then use this logon in your phone-based application.

For a website to be usable on the small screen of a Windows Phone, you will most likely need to provide a more compact view of the pages than would be delivered to a normal desktop browser. You can use features in ASP.NET to detect the type of browser making the request and adapt the output simply by applying a different style sheet (for example, to remove extraneous content such as navigation bars and related content panels). Alternatively, you may decide to generate completely different pages with a simplified navigation model when you detect that the client is a small-screen device such as a phone.

These topics are not covered in this guidance, but you can learn more from the video presentation, *Designing and Developing for the Rich Mobile Web* on the Mix 10 website (<http://live.visitmix.com/MIX10/Sessions/CL23>) or by downloading the white paper, *Designing Web Sites for Phone Browsers*, from the Microsoft Download Center (<http://go.microsoft.com/?linkid=9713253>).

Debugging Windows Phone 7 Applications

If you are working in Visual Studio or Visual Studio Express, you can use the standard debugging tools to break, step through, and profile a Windows Phone 7 application. You can also view variable values and the execution tree and processes.

A Windows Phone 7 application uses Silverlight and Windows Presentation Foundation (WPF) to define the interfaces, so any debugging techniques you use in Silverlight and WPF also apply to Windows Phone 7. For example, there are some specific issues related to WPF styling and data binding that can arise in an application. These include the following:

- **Errors in the auto-generated partial class** for a XAML component that prevent the component from being available to code in the page or the application. In most cases, you can

resolve this by right-clicking the XAML file in Solution Explorer and then clicking **Run Custom Tool** to regenerate the partial class.

- **Errors in the XAML style definitions for components.** These may occur in the App.xaml file or in other files within your application. Typically, the exception message indicates only that a COM component error has occurred and provides an HRESULT value. The simplest approach to resolving these types of errors are to minimize the surface area by temporarily removing some or all of the XAML styling definitions until the error disappears, and then reading them one by one to locate the faulty definition.
- **Data binding errors**, such as using the wrong property name in a binding statement. This does not generate a run-time error, but the control property (such as the text) is not set. You can see a list of data binding errors in the Output window of Visual Studio or Visual Studio Express as the application executes. For example, you will see the following message if you specify binding to a property named **City** that does not exist in the data context.

```
System.Windows.Data Error: BindingExpression path error:  
'City' property not found on [your data context component  
name]. BindingExpression: Path='City' DataItem='[your  
data context item]'; target element is 'System.Windows.  
Controls.TextBlock' (Name='ItemText'); target property  
is 'Text' (type 'System.String').
```

- **Remote service access errors.** If your application cannot retrieve data from a remote service, try copying the URL of the service from your code into a web browser to see if it is accessible. You can also download and install a web debugging proxy utility such as Fiddler or a similar utility to view all the HTTP(S) traffic between your application and the service. Fiddler is available from the Fiddler website (<http://www.fiddler2.com/fiddler2/>).

One specific point to note when debugging Windows Phone 7 applications using the emulator is that certain actions within the phone may halt the debugger. For example, if the code opens a task or the launcher, or if it is tombstoned, execution may not continue automatically after the interruption. To continue debugging, press F5 in Visual Studio within 10 seconds of the task completing or the applications being reactivated.

Note: Tools that intercept and analyze network traffic, such as Fiddler, are not supported in the current release of the phone tools. For example, if you configure Fiddler to track all network traffic, the emulator will not be able to connect to the network.

Unit Testing Windows Phone 7 Applications

Unit testing is the process of exercising individual business objects, state, and validation for your applications. A comprehensive set of unit tests written before you develop the code, and executed regularly during the entire development cycle, help to ensure that individual components and services are performing correctly and providing valid results.

You can use a special version of the Silverlight Unit Testing Framework adapted for use with Silverlight 3 to run unit tests for Windows Phone 7 applications. You can obtain this from Jeff Wilcox's site (<http://www.jeff.wilcox.name/2010/05/sl3-utf-bits/>). Jeff is a Senior Software Development Engineer at Microsoft on the Silverlight for Windows Phone team.

The Silverlight Unit Testing Framework adds the Silverlight Unit Test Application templates to the **New Project** dialog box in Visual Studio and Visual Studio Express, and you can use this to add a test project to your application. The framework allows you to execute tests that carry the standard Visual Studio test attributes and include the standard test assertions such as **IsTrue** and **IsNotNull**. It can be used to run all the unit tests in a solution or selected tests; it reports the results on the phone, showing full details of each test that passed or failed.

You can also run tests on the desktop using a traditional test framework such as the Microsoft Test Framework (MSTest), instead of deploying the application to the phone or the emulator and running it there under the test framework. This is generally faster, but you must be aware that there are some differences in the execution environment, and tests may behave differently on the desktop when compared to running on the emulator or a physical device. When performing integration testing, you should run the tests on the emulator or a physical device.

There are some tests that you cannot run in an emulator. For example, you cannot test the Global Positioning System (GPS) or the accelerometer. In these cases, you may prefer to use alternative testing techniques, such as creating mock objects that represent the actual service or component that your application uses and substituting these for the physical service or component.

To view a video presentation about unit testing using the Silverlight Unit Testing Framework, see *Unit Testing Silverlight and Windows Phone Applications* on the Mix 10 website (<http://live.visitmix.com/MIX10/Sessions/CL59>).

AUTOMATED UNIT TESTING

It is possible to automate testing on the Windows Phone 7 emulator if you need to implement a continuous integration process and build Windows Phone 7 projects on a separate build server that does not have the Windows Phone Developer Tools installed. This requires setting up a folder for the external dependencies and editing the project files. For more information, see the post, “Windows Phone 7 Continuous Integration,” on Justin Angel’s blog (<http://justinangel.net/#BlogPost=TFS2010WP7ContinuousIntegration>).

It is also possible to automate building and deploying an application to a phone or emulator without using Visual Studio. This approach uses the Smart Device Connectivity API implemented in the assembly **Microsoft.SmartDevice.Connectivity.dll**. For more information about using this API, see “Smart Device Connectivity API Reference” on MSDN ([http://msdn.microsoft.com/en-us/library/bb545992\(VS.90\).aspx](http://msdn.microsoft.com/en-us/library/bb545992(VS.90).aspx)) and “Windows Phone 7 Emulator Automation” on Justin Angel’s blog (<http://justinangel.net/#BlogPost=WindowsPhone7EmulatorAutomation>).

Additional Tools and Frameworks

Not all the tools and frameworks you use in Silverlight applications on the desktop are suitable for developing Windows Phone 7 applications. For example, dependency injection containers such as the Unity Application Block (Unity) that demand certain security permissions or dynamically generate Microsoft intermediate language (MSIL) code cannot be deployed to the phone.

However, there are several tools and frameworks available that can be used successfully in Windows Phone 7 applications. The following are some examples:

- The Moq framework for implementing mock objects during unit testing:
<http://code.google.com/p/moq/downloads/detail?name=Moq.4.0.10818.0.zip>
- The FUNQ dependency injection container:
<http://funq.codeplex.com/>
- The Ninject dependency injection container:
<http://ninject.org/>

- The WebAii testing framework:
<http://www.telerik.com/products/web-testing-tools/webaii-framework-features.aspx>
- The Prism Library for Windows Phone 7 (part of the main Prism library): <http://www.microsoft.com/prism>

If you want to access services exposed through Windows Communication Foundation (WCF) Data Services in Open Data Protocol (OData) format, you may consider using the special version of the OData client library for Windows Phone 7, which is implemented in the assembly **System.Data.Services.Client.dll**. To download it, see “OData Client Library for Windows Phone 7 Series” in the Microsoft Download Center (<http://www.microsoft.com/downloads/details.aspx?FamilyID=b251b247-70ca-4887-bab6-dccdec192f8d&displaylang=en>).

You must use the WCF Data Service Client Utility (DataSvcUtil.exe) to create a suitable proxy class to access your services when using the OData client library. This utility is provided with the .NET Framework 4.0; it is typically installed in the folder Windows\Microsoft.NET\Framework[64]\v4.0. For more information, see “WCF Data Service Client Utility” on MSDN (<http://msdn.microsoft.com/en-us/library/ee383989.aspx>).

For more information about accessing remote data services in Windows Phone 7 Silverlight applications, see Chapter 2, “Designing Applications for Windows Phone 7.”

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Appendix B

Silverlight and XNA in Windows Phone 7

Windows® Phone 7 supports development of applications using both the Silverlight® and XNA® frameworks. You can also interact with XNA from a Silverlight application to perform tasks that are not supported in Silverlight if you choose it as the primary development framework for your application.

This appendix will help you understand the differences between the Silverlight and XNA development models, and choose the appropriate model for your application. It describes the basic differences between Silverlight and XNA, and it discusses how XNA is focused on game-oriented applications rather than business applications. However, you can write games using Silverlight if you want. Windows Phone 7 devices contain a graphics processing unit (GPU) that can provide acceptable performance for games written in Silverlight.

Basic Differences between Silverlight and XNA

The following sections and the table that follows them indicate the basic differences between Silverlight and XNA.

EXECUTION MODEL

The execution model for Silverlight is event-driven and application-focused. Interactive controls and components expose events that you can handle to perform processing based on user input, environmental effects, and more. XNA uses a gaming loop and is focused on the content displayed on the screen.

PERFORMANCE

XNA is optimized for performance and GPU usage in order to support a fully immersive gaming experience. Unlike Silverlight, XNA supports three-dimensional (3-D) graphics. You cannot interop with the graphics mechanism from Silverlight.

SOUNDS

XNA supports more advanced options for playing sounds, including multi-channel audio. You can only record sounds by using XNA. If you are developing using Silverlight, you can use interop with the XNA framework to record sounds using the built-in microphone on the device.

SCREEN DISPLAY

XNA uses the full screen all the time, and you must generate all the content for display within your code. There are no controls available. Silverlight provides a comprehensive range of controls, and it can interact with the built-in launchers and choosers within the Windows Phone 7 operating system to provide a wide range of capabilities for interactive applications.

SUMMARY OF THE BASIC DIFFERENCES

Platform features	Windows Phone application (Silverlight-based application model)	Windows Phone game (XNA Framework application model)
Application programming model	Event-driven application model that is ideal for mainly user interface (UI)-based applications	Traditional frame loop for a more simulation-based experience and content.
Control-based, data-bound UI development	UIElement and related types	None, developers must write their own.
Video playback	Rich video integration via MediaElement	Full-screen playback via the system media player.
2-D graphics	Rich paths, shapes, brushes, and more	High performance 2-D rendering for a large number of sprites.
3-D graphics	Perspective effects using PlaneProjection transform	Provides hardware-accelerated 3-D APIs.
Primary data serialization model	XAML	Content manager.
Designer tooling	Microsoft® Expression Blend® design software, Microsoft Visual Studio® XAML designer	XNA Content Pipeline integrates third-party tools for creating 3-D model and texture assets for the application.
Device sensors	Same API in both programming models.	
Access to user's songs and pictures	Same API in both programming models.	

For more information about Silverlight, see the Silverlight website (<http://www.silverlight.net/>). For more information about XNA, see the App Hub website (<http://www.xna.com/>).

The XNA Game Execution Model

XNA uses a game loop that is generated by the design tools. The life cycle for a game consists of four main stages, as shown in Figure 1.

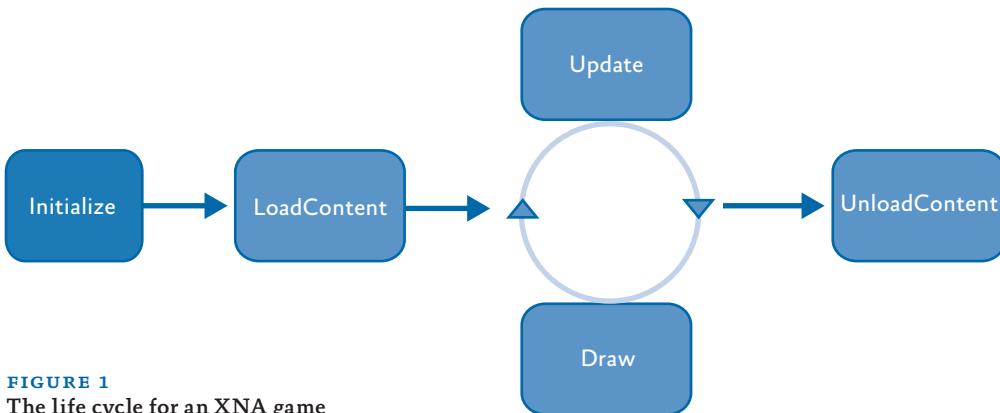


FIGURE 1
The life cycle for an XNA game

The game first initializes and then loads content such as sprites, images, sounds, and other components of the game. It then initiates the game loop. The game loop consists of two processes that run asynchronously: code that draws the frame containing the complete screen to a buffer, and code that updates the screen from the buffer. This loop runs until the user closes the game, at which point the content is unloaded.

Draw and **Update** are executed at different rates. **Draw** is executed as frequently as possible based on the maximum frame rate that the device can provide, whereas **Update** is executed at a regular rate. The draw rate should generally be higher than the update rate.

For more information about creating games using XNA, see “Getting Started with XNA Game Studio Development” on MSDN® (<http://msdn.microsoft.com/en-us/library/bb203894.aspx>). For a walkthrough of creating a simple game using XNA, see “Your First Game: Microsoft XNA Game Studio in 2D” at on MSDN (<http://msdn.microsoft.com/en-us/library/bb203893.aspx>).

Using Interop from Silverlight to XNA

The following are some of the features of Windows Phone 7 that you may want to use in a Silverlight application where interop with XNA is required:

- Accessing input gestures on the device, such as drag, pinch, and hold gestures
- Playing multi-channel audio
- Advanced sound effects (such as using the XNA **DynamicSoundEffectsInstance** class, which lets you submit packets on the fly, vary the pitch, and apply 3-D effects to the sound)
- Recording audio

For the last two features in the preceding list, you must create an XNA asynchronous dispatcher service and initialize it in the constructor or startup event in your App.xaml.cs file. This is required to raise the events in Silverlight when they are raised by the XNA objects.

CREATING AND USING AN XNA DISPATCHER SERVICE

The XNA run-time event mechanism is different from the Silverlight event mechanism. Events raised by XNA objects are available within the XNA game loop, but they are not automatically available to the Silverlight event mechanism. To expose XNA events to Silverlight application code, you can take advantage of the Silverlight ability to use application extension services. These are services that add functionality to an application, and that can be reused by many applications. They circumvent the requirement to use inheritance. For more information, see “Application Extension Services” on MSDN ([http://msdn.microsoft.com/en-us/library/dd833084\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/dd833084(VS.95).aspx)).

The following code example shows an XNA asynchronous event dispatcher service that you can use in your Silverlight applications.

```
C#
using System;
using System.Windows;
using Microsoft.Xna.Framework;
using System.Windows.Threading;

namespace YourAppNameSpace
{
    public class XNAAsyncDispatcher : IApplicationService
    {
        private DispatcherTimer frameworkDispatcherTimer;
```

```

public XNAAsyncDispatcher(TimeSpan dispatchInterval)
{
    this.frameworkDispatcherTimer = new DispatcherTimer();
    this.frameworkDispatcherTimer.Tick
        += new EventHandler(frameworkDispatcherTimer_Tick);
    this.frameworkDispatcherTimer.Interval = dispatch
        Interval;
}

void IApplicationService.StartService(ApplicationService
Context context)
{
    this.frameworkDispatcherTimer.Start();
}
void IApplicationService.StopService()
{
    this.frameworkDispatcherTimer.Stop();
}
void frameworkDispatcherTimer_Tick(object sender, EventArgs e)
{
    FrameworkDispatcher.Update();
}
}
}

```

The **DispatcherTimer** class is a .NET Framework class that is included in Silverlight. It executes on the application thread and can raise events at preset intervals. In the preceding **XNAAsyncDispatcher** example, the **Tick** event executes the **Update** method of the **FrameworkDispatcher** class, which is part of the XNA Framework. The **FrameworkDispatcher** class is an alternative mechanism to drive the update loop if the base class is not an XNA game.

For more information, see “DispatcherTimer Class” (<http://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher-timer.aspx>) and “FrameworkDispatcher.Update” (<http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.frameworkdispatcher.update.aspx>) on MSDN.

After you create an XNA dispatcher service class, you can instantiate it in the constructor of your application class in the App.xaml.cs file. The typical tick rate for capturing and exposing XNA events is 50 milliseconds.

```

C#
public App()
{
    // other constructor code here ...
    this.ApplicationLifetimeObjects.Add(new
        XNAAsyncDispatcher
        (TimeSpan.FromMilliseconds(50)));
}

```

For information about using these and other device capabilities, see Appendix C, “Leveraging Device Capabilities”.

Excluded Classes and Assemblies

There are limitations on the classes and assemblies you can use or deploy to Windows Phone 7, depending on whether you are using the XNA or the Silverlight development model. For a full list of these, see “The Silverlight and XNA Frameworks for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff402528\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402528(VS.92).aspx)).

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Appendix C

Leveraging Device Capabilities

Windows® Phone 7 devices offer a range of capabilities that, at first glance, do not seem to be applicable to business applications, which generally are more concerned with displaying and processing data. However, these capabilities are part of the overall environment for applications that run on mobile devices, and they can often add useful extra functionality to mobile applications. Some of these features are also available on desktop and laptop computers, but several are specific to mobile devices such as phones.

For example, Windows Phone 7 devices implement motion and altitude detection; location detection; media access and recording functionality for photos, music, audio, and video; contacts and messaging management for email and SMS services; integrated web access with search; access to installable applications; touch and gesture detection; as well as the standard telephone call capabilities.

Many of the features are accomplished through built-in mini-applications or tasks implemented by classes within the phone operating system that you can invoke when required. These tasks are known as launchers and choosers. Launchers usually invoke a specific task with a predefined outcome, such as adding a phone number to the contacts list. Choosers allow the user to perform some action that involves them choosing a result. An example is the **CameraCaptureTask** chooser, which allows users to take photos and then choose which photos to keep.

Note: *Some of the tasks and capabilities of a Windows Phone 7 device are not supported or not fully functional in the emulator.*

For more information, see “Launcher and Chooser Support in Windows Phone Emulator” on MSDN® ([http://msdn.microsoft.com/en-us/library/ff955600\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff955600(VS.92).aspx)).

Scenarios for Device Capabilities

Designers and developers can take advantage of several of the mobile device capabilities in their Windows Phone 7 applications, both to extend the functionality for users and to incorporate features that are particularly useful within the scenario of using a mobile device. The following are typical scenarios for the device capabilities of Windows Phone 7:

- Using the Location Service to determine the location of the user. This information may be useful, and it could be included in the data submitted to a server by an application when required. Alternatively, it may be used to display a map on the phone that is centered on the current location.
- Using the extended device information capabilities to get information about the device, manage the memory used by the application, and maximize its performance.
- Using the accelerometer to detect short bursts of motion of the device in order to implement functions commonly associated with certain types of motion. For example, shaking the device vigorously could cancel a partly completed process and allow the user to restart it.
- Using the touch detection capabilities to capture gestures such as hold, drag, pinch, and flick. For example, a hold gesture might, after a specified delay, bring up a screen containing extra information, while a flick gesture might scroll a list.
- Using the camera to capture data or other relevant information in the form of an image to be included in the data submitted to a server or stored on the device. For example, capturing a barcode could initiate a web search for that specific product.
- Using the sound recording feature to record voice from the phone's microphone. For example, allowing users to submit data in an audio format or include recorded sound clips in their responses sent to a server.
- Using the media features of the device to stream information that is useful to users as they use an application. For example, a user may be able to access audio files that contain help on using the application, or sounds and pictures could be part of the application experience.
- Using the contacts and messaging features to integrate the application with the user's lists of contacts, and to invoke phone and messaging features. For example, the application could

prompt users to add useful contacts, including the appropriate email address and phone number, to the list of contacts stored on their phone. It might also invoke the messaging features to make it easy for the user to dial a number such as the customer support department, create a new email message with a specified subject and body text, or create an SMS (text) message.

- Using the vibration capability to alert users of events occurring in the application. For example, the application could alert users that new data is available, or that an asynchronous delivery of data has completed.
- Invoking the web browser to display a specified web page. This might be useful for providing additional content or advice to the user, or for allowing the user to access specific pages on your own website.
- Invoking the search feature with a specified search string. This could be useful to help users more easily search for specific information related to the application they are using.
- Invoking the Windows Marketplace hub and showing a specific application that the user can download and install. For example, you might use this feature to highlight new versions of the application.

The following sections describe the basic usage of the device capabilities available in Windows Phone 7.

Accelerometer

All Windows Phone 7 physical devices include an accelerometer that measures acceleration forces due to gravity, or due to forces caused by moving the device, and indicates the orientation of the device. The Accelerometer API in the operating system of the device can be accessed by Microsoft® Silverlight® for Windows Phone and XNA® for Windows Phone applications.

To use the Accelerometer API, you must do the following:

- Add a reference to the assembly **Microsoft.Devices.Sensors** to your project.
- Instantiate an accelerometer.
- Specify a callback in the form of an event handler or lambda expression that will receive the data.
- Start the accelerometer.

The following code example shows how to use the Accelerometer API.

```
C#
var accel = new Accelerometer();
accel.RadingChanged += new EventHandler
<AccelerometerReadingEventArgs>(Accelerometer_ReadingChanged);
try
{
    accel.Start();
}
catch
{
    // Error starting the accelerometer.
}
```

You can then access the readings as they change in your event handler. Remember that the event handler runs on a different thread from the user interface (UI), so you must invoke a method on the UI thread if you want to update the UI with the discovered values.

```
C#
void Accelerometer_ReadingChanged(object sender,
                                    AccelerometerReadingEventArgs e)
{
    string xCoordinate = e.X.ToString("0.00");
    // Acceleration in the X direction.
    string yCoordinate = e.Y.ToString("0.00");
    // Acceleration in the Y direction.
    string zCoordinate = e.Z.ToString("0.00");
    // Vertical acceleration.
    DateTimeOffset time = e.Timestamp;
    // When this reading was obtained.
}
```

The values returned are of type **Double** and indicate the force applied in the three planes of the device. When it is laid flat on its back, they will return values very close to **X=0, Y=0, Z=-1**. When held vertically, they return approximately **X=0, Y=-1, Z=0**. Figure 1 illustrates how the values indicate the orientation of forces on the device.

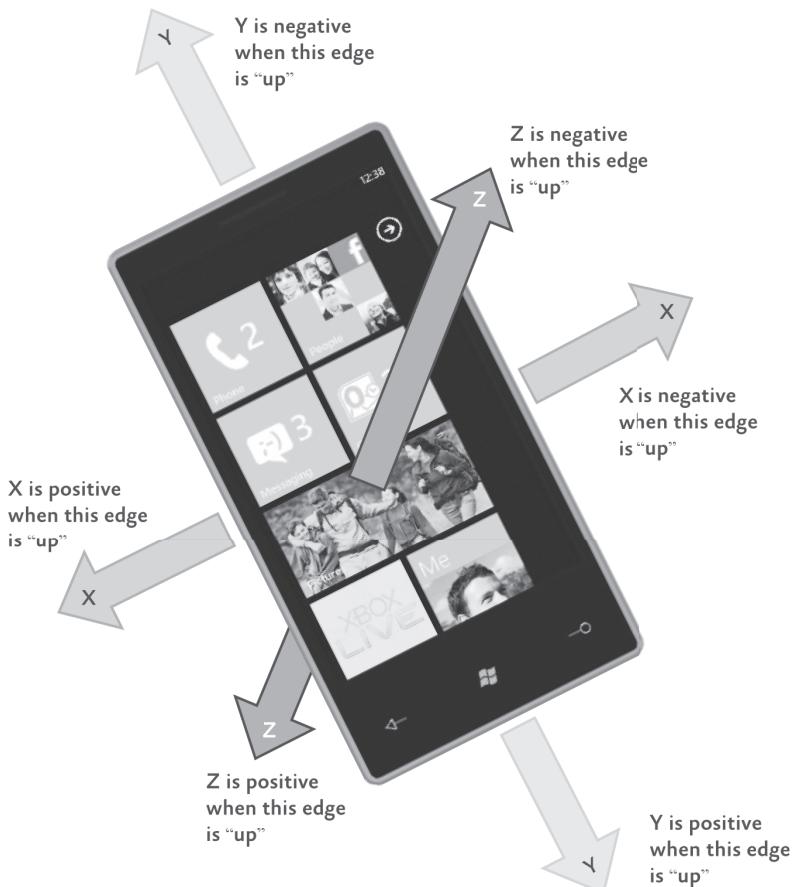


FIGURE 1
The orientation of forces detected by the accelerometer

Note: The accelerometer is very sensitive, and you may consider disregarding changes of less than 0.05 or 0.1 in these values. It also returns values at the rate of 50 times per second, so the values will appear to vary continuously.

The accelerometer also exposes the **State** property, which returns a value from the **SensorState** enumeration indicating the current state of the accelerometer. Typical values are **Initializing**, **Ready**, **NoData**, **NoPermissions**, **Disabled**, and **NotSupported**.

To stop the accelerometer when you no longer require it, which significantly reduces the processing load on the device, call the **Stop** method.

For more information, see “Accelerometer for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431793\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431793(VS.92).aspx)). You can also download a code sample that demonstrates using the Accelerometer API from “Code Samples for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431744\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)).

Camera

All Windows Phone 7 physical devices include a high resolution camera. This is managed by the operating system, but your applications can invoke the camera using the **CameraCaptureTask** chooser.

To use the camera, you must do the following:

- Add a reference to the assembly **Microsoft.Phone.Tasks** to your project.
- Create a new **CameraCaptureTask** chooser instance as a global scoped variable (the application will be deactivated when the chooser is launched).
- Specify a callback in the form of an event handler, or a lambda expression that will receive the picture when the user closes the chooser.
- Show the chooser.

The camera is designed to be used in landscape mode, and it generates a picture in landscape format. The Exchangeable Image File (EXIF) headers in the image file indicate the correct orientation of the picture, and the photos application in the camera will automatically display the image in the correct orientation. However, in your own applications, you may need to rotate the image to obtain the correct orientation.

A library that you can use to read EXIF information in an image is available from “Understanding and Reading Exif Data” on The Code Project website (http://www.codeproject.com/KB/silverlight/Exif_Data.aspx). Details of how to rotate an image can be found in post, “Handling picture orientation in CameraCaptureTask in Windows Phone 7,” on Tim Heuer’s blog (<http://timheuer.com/blog/archive/2010/09/23/working-with-pictures-in-camera-tasks-in-windows-phone-7-orientation-rotation.aspx>).

The following code example shows how you can use the **CameraCaptureTask** chooser to capture a photo.

C#

```
CameraCaptureTask ctask = new CameraCaptureTask();
ctask.Completed += new EventHandler<PhotoResult>
    (ctask_Completed);
ctask.Show();
```

You then access the picture that the user captured in your event handler. The user may have taken and discarded several photos, and the chooser returns only the one the user decided to keep. The following code example shows how you can access the chosen photo and save it into isolated storage on the device as a JPEG image. To use this code, you must add references to the namespaces **System**, **Windows.Media.Imaging**, **System.IO.IsolatedStorage**, and **Microsoft.Phone** (for the **PictureDecoder** class).

```
C#
void ctask_Completed(object sender, PhotoResult e)
{
    WriteableBitmap theImage = PictureDecoder.DecodeJpeg
        (e.ChosenPhoto);
    String pictureName = "MyPhoto.jpg";

    // Create a virtual store and file stream.
    // Check for an existing duplicate name.
    var myStore = IsolatedStorageFile.GetUserStoreForApplication();
    if (myStore.FileExists(pictureName))
    {
        myStore.DeleteFile(pictureName);
    }
    IsolatedStorageFileStream myFileStream = myStore.
        CreateFile(pictureName);

    // Encode the WriteableBitmap into the JPEG stream and
    // place it into isolated storage.
    Extensions.SaveJpeg(theImage, myFileStream,
        theImage.PixelWidth, theImage.PixelHeight, 0, 85);
    myFileStream.Close();
}
```

You can also use interop with the XNA operating system classes in Windows Phone 7 to save the photo to the Media Library, or to interact with the Media Library.

For more information about using the camera in your applications, see “[CameraCaptureTask Class](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.camercapturetask(VS.92).aspx)” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.camercapturetask\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.camercapturetask(VS.92).aspx)). You can also download a code sample that demonstrates capturing camera images from “[Code Samples for Windows Phone](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431744\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)).

Contacts and Messaging

Mobile computing is typically an activity based around contacts and messaging. All Windows Phone 7 devices support a contacts list, standard phone capability, SMS (text) messaging, and email. An application can integrate with these features to make it easier for users to add specific contacts to their contacts list, make a phone call, send pre-populated SMS messages, and send pre-populated emails to their contacts. All of these features display screens where the user must manually confirm the action. For security reasons, your code cannot perform any of these tasks in the background without the user being aware.

The contacts and messaging functionality is managed by the operating system, but your applications can invoke the appropriate task launcher or chooser to initiate it. For example, you can prompt the user to call to a specific phone number using the **PhoneCallTask** launcher. The user can click the **call** button to initiate the call or click the **don't call** button to cancel the operation and close the launcher.

The following task launchers and choosers are available for contacts and messaging:

- **SaveEmailAddressTask.** This makes it easy for a user to add an email address you specify in your code to their contacts list. You can use the task callback function to check if the user actually did add the email address to their contacts list.
- **SavePhoneNumberTask.** This makes it easy for a user to add a phone number you specify in your code to their contacts list. You can use the task callback function to check if the user actually did add the phone number to their contacts list.
- **EmailAddressChooserTask.** This allows the user to select a specific email address from their contacts list. The selected address is available to code in the task callback function.
- **PhoneNumberChooserTask.** This allows the user to select a specific phone number from their contacts list. The selected number is available to code in the task callback function.
- **PhoneCallTask.** This prompts the user to dial a number you specify in your code. It does not provide a callback.
- **EmailComposeTask.** This opens the email editor with the To and CC addresses and the subject and message body text set to values that you specify in your code. It does not provide a callback.
- **SmsComposeTask.** This opens the SMS message editor with the phone number and message text set to values that you specify in your code. It does not provide a callback.

To use these task launchers and choosers, you must do the following:

- Create a new instance of the appropriate task launcher or chooser class.
- Specify the properties of the task launcher or chooser. For example, set the number to call and, optionally, a display name for the **PhoneCallTask**.
- If the task launcher or chooser provides a callback, specify a function or lambda expression that will be executed when the user completes the task launcher or chooser.
- Show the task launcher or chooser.

Some of the task launchers and choosers accept a reference to a callback that is executed when it completes. For example, the **EmailAddressChooserTask** passes the chosen email address to the callback inside an instance of the **EmailResult** arguments class. All the callbacks receive a task-specific arguments type, and all of these include a **TaskResult** property that indicates if the task was completed successfully or cancelled by the user.

The following code example shows the simple case where the task does not provide callback functionality. It invokes the phone dialing feature with a specified phone number displayed. The user dials the number by clicking the **call** button.

C#

```
PhoneCallTask phoneTask = new PhoneCallTask();
phoneTask.DisplayName = "Tailspin Support";
phoneTask.PhoneNumber = "5551234567";
phoneTask.Show();
```

For some tasks that do accept a callback, such as the **SaveEmailAddressTask** and **SavePhoneNumberTask**, you may not be interested in whether the user actually did add the suggested contact to their contacts list. In this case, you simply omit setting the callback for the task. The following code example shows how you can prompt a user to add a specific email address (with a display name) to the contacts list.

C#

```
SaveEmailAddressTask saveEmailTask = new SaveEmailAddressTask();
saveEmailTask.Email = "Tailspin <customerservice@tailspin.com>";
saveEmailTask.Show();
```

For tasks where you do specify a callback, you access the result through the event argument passed to the callback handler. For example, the following code invokes the chooser for an email address and specifies a callback named **EmailChooser_Completed**.

C#

```
EmailAddressChooserTask emailAddressTask =
    new EmailAddressChooserTask();
emailAddressTask.Completed
    += new EventHandler<EmailResult>(EmailChooser_Completed);
emailAddressTask.Show();
```

The callback can check if the user selected an email address. If so, it invokes the email editor with a pre-populated email ready for the user to edit and send.

C#

```
void EmailChooser_Completed(object sender, EmailResult result)
{
    if (result.TaskResult == TaskResult.OK)
    {
        EmailComposeTask emailComposeTask = new EmailComposeTask();
        emailComposeTask.To = result.Email;
        emailComposeTask.Subject = "My Message Subject";
        emailComposeTask.Body = "This is the content of my message";
        emailComposeTask.Show();
    }
}
```

For more information about the tasks described in this section, and the other types of tasks available on Windows Phone 7 devices, see “Microsoft.Phone.Tasks Namespace” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks(VS.92).aspx)).

Device Information

Windows Phone 7 includes the **DeviceExtendedProperties** class that you can use to obtain information about the physical device. The information you can retrieve includes the following:

- The device manufacturer, the device name, and its unique device ID
- The version of the phone hardware and the version of the firmware running on the phone
- The total physical memory (RAM) installed in the device, the amount of memory currently in use, and the peak memory usage of the current application

However, you must be aware that the phone will alert the user when some device information is retrieved and the user can refuse to allow the application to access it. You should access device information only if it is essential to your application. Typically, you will use device information to generate statistics or usage data, and to monitor memory usage of your application. You can use this data to adjust the behavior of your application to minimize impact on the device and other applications.

You retrieve device information using the **GetValue** or the **TryGetValue** method, as shown in the following code example.

C#

```
using Microsoft.Phone.Info;
...
string manufacturer = DeviceExtendedProperties.GetValue(
    "DeviceManufacturer").ToString();
string deviceName = DeviceExtendedProperties.GetValue(
    "DeviceName").ToString();
string firmwareVersion = DeviceExtendedProperties.GetValue(
    "DeviceFirmwareVersion").ToString();
string hardwareVersion = DeviceExtendedProperties.GetValue(
    "DeviceHardwareVersion").ToString();
long totalMemory = Convert.ToInt64(
    DeviceExtendedProperties.GetValue("DeviceTotalMemory"));
long memoryUsage = Convert.ToInt64(
    DeviceExtendedProperties.GetValue(
        "ApplicationCurrentMemoryUsage"));
object tryValue;
long peakMemoryUsage = -1;
if (DeviceExtendedProperties.TryGetValue(
    "ApplicationPeakMemoryUsage", out tryValue))
{
    peakMemoryUsage = Convert.ToInt64(tryValue);
}
// The following returns a byte array of length 20.
object deviceID = DeviceExtendedProperties.GetValue(
    "DeviceUniqueId");
```

The device ID is a hash represented as an array of 20 bytes, and is unique to each device. It does not change when applications are installed or the firmware is updated.

When running in the emulator, the manufacturer name returns "Microsoft," the device name returns "XDeviceEmulator," and (in the initial release version) the hardware and firmware versions return 0.0.0.0.

For more information, and a list of properties for the **Device ExtendedProperties** class, see "Device Information for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff941122\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941122(VS.92).aspx)).

Location and Mapping

Windows Phone 7 physical devices contain features that allow the device to determine its geographical position in terms of latitude and longitude. The data is obtained through a combination of the built-in Global Positioning System (GPS) feature, the location of cellular phone stations that the device is communicating with, and information obtained from Wi-Fi connections. It also contains a control that you can use to display a map in your application.

When using the Location Service in an application, you should be aware of the following factors:

- Location data may not be available at all times.
- The accuracy depends on the types of information available from the sensors, and on the settings you specify for the Location Service.
- Using the Location Service, especially when you specify high accuracy, consumes additional power and more quickly drains the device's battery.
- Initializing and obtaining the first reading when the Location Service is started may take up to a minute.
- The user must give his permission for code to access location information. If the user has disabled the Location Service, your application must be able to operate without location information or warn the user that it cannot run unless he enables the Location Service.

Good practice for using the Location Service is to turn it on only when required, and turn it off as soon as is practical. Also consider using the lower accuracy, power-optimized setting unless your application absolutely requires higher accuracy. For example, you may only want to know the city or region where the device is located.

You can use the Location Service in two ways. You can set it

running asynchronously and obtain position readings from the callback handlers attached to the **GeoCoordinateWatcher** instance when data becomes available. Alternatively, you can start the **GeoCoordinateWatcher** instance synchronously so that your code will wait until the first reading is obtained, or until the process times out after a specified interval.

To use the Location Service, you must do the following:

- Add a reference to the assembly **System.Device** to your project. The classes you will use are in the **System.Device.Location** namespace.
- Instantiate a **GeoCoordinateWatcher**.
- Specify the properties for the **GeoCoordinateWatcher**.
- Optionally specify callbacks in the form of event handlers or lambda expressions that will receive the data if you are using the service asynchronously.
- Start the **GeoCoordinateWatcher**.

ASYNCHRONOUS LOCATION SERVICE OPERATION

The following code example creates and starts a **GeoCoordinateWatcher** instance asynchronously.

C#

```
// Create a GeoCoordinateWatcher and specify the accuracy required.  
// Use GeoPositionAccuracy.Default unless high accuracy required.  
GeoCoordinateWatcher watcher =  
    new GeoCoordinateWatcher(GeoPositionAccuracy.High);  
// Check if the user has provided permission to use the service.  
if (watcher.Permission == GeoPositionPermission.Denied)  
{  
    // Permission to use Service Location denied by user.  
}  
else  
{  
    // Set movement threshold in meters.  
    // Use at least 20 to avoid excessive  
    // change events generated by minor changes  
    // in environment or signal quality.  
    // Higher values also help to maximize battery  
    // life for the device.  
    watcher.MovementThreshold = 30;  
    // Add handlers for StatusChanged and PositionChanged events.  
    watcher.StatusChanged  
        += new EventHandler<GeoPositionStatusChangedEventArgs>  
            (Watcher_StatusChanged);
```

```

watcher.PositionChanged
+= new EventHandler<GeoPositionChangedEventArgs
<GeoCoordinate>>(Watcher_PositionChanged);
// Start data acquisition.
watcher.Start();
}

```

The **StatusChanged** event handler can detect if the watcher service is ready and contains data, allowing you to display a corresponding message to the user and interact with the service in your application code as required. The value is one of the **GeoPositionStatus** enumeration values: **Ready**, **Disabled**, **Initializing**, and **NoData**.

C#

```

void Watcher_StatusChanged(object sender, GeoPositionStatus
   ChangedEventArgs e)
{
    string currentStatus = e.Status.ToString();
}

```

Remember that the event handlers run on a different thread from the UI, so you must invoke a method on the UI thread if you want to update the UI with the discovered values.

The **PositionChanged** event handler is executed each time a position change greater than that defined for the **MovementThreshold** property occurs. It receives an instance of the **GeoCoordinate** class that contains the current location information.

C#

```

void Watcher_PositionChanged(object sender,
                           GeoPositionChangedEventArgs
                           <GeoCoordinate> e)
{
    // Check to see if location data is available.
    if (e.Position.Location.IsUnknown)
    {
        // Data is not available
    }
    else
    {
        double latitude = e.Position.Location.Latitude;
        double longitude = e.Position.Location.Longitude;
        double altitude = e.Position.Location.Altitude;
        double course = e.Position.Location.Course;
        double speed = e.Position.Location.Speed;
    }
}

```

```
        double hAccuracy = e.Position.Location.HorizontalAccuracy;
        double vAccuracy = e.Position.Location.VerticalAccuracy;
        DateTimeOffset time = e.Position.Timestamp;
    }
}
```

Note that some values may not be available (NaN) depending on the current location method in use and the specified accuracy. When you no longer need to obtain location information, you should preserve battery life by stopping the watcher. To do this, call the **Stop** method.

SYNCHRONOUS LOCATION SERVICE OPERATION

If you want your code to wait for the first position value to be available from the service, or if you just want to obtain a single position value, use the **TryStart** method and specify a timeout value. The first parameter of this method should be set to **true** in this release of the service. The **GeoCoordinateWatcher** class exposes the **Status** property, which is one of the **GeoPositionStatus** enumeration values. If the status is **Ready**, you can check for and retrieve the data exposed by the service from the **Position** property of the watcher.

C#

```
GeoCoordinateWatcher watcher = new GeoCoordinateWatcher
    (GeoPositionAccuracy.High);
watcher.TryStart(true, TimeSpan.FromSeconds(60));

// Check to see if location data is available.
if (watcher.Status == GeoPositionStatus.Ready
    && !watcher.Position.Location.IsUnknown)
{
    double latitude = watcher.Position.Location.Latitude;
    double longitude = watcher.Position.Location.Longitude;
    double altitude = watcher.Position.Location.Altitude;
    double course = watcher.Position.Location.Course;
    double speed = watcher.Position.Location.Speed;
    double hAccuracy = watcher.Position.Location.HorizontalAccuracy;
    double vAccuracy = watcher.Position.Location.VerticalAccuracy;
    DateTimeOffset time = watcher.Position.Timestamp;
}
else
{
    // Data is not available.
}
watcher.Stop();
```

Note that some values may not be available (NaN) depending on the current location method in use and the specified accuracy.

For more information, see “Location for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431803\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431803(VS.92).aspx)). You can also download a code sample that demonstrates using the Location Service from “Code Samples for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431744\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)).

USING LOCATION INFORMATION AND DISPLAYING A MAP

Location information provides only the coordinates of the device. You can subscribe to a mapping service or use helper routines to convert the coordinates into a street-level address or other format more suited to your requirements.

If you simply want to display a map, you can use the web browser on the device to open a mapping site at the appropriate location. For example, the following code opens the Bing™ Maps website using the latitude and longitude values. To use this code, you must add a reference to the **System.Text** namespace.

```
C#
StringBuilder mapUrl = new StringBuilder(
    "http://www.bing.com/maps/default.aspx?lvl=15&style=r&v=2&cp=");
mapUrl.Append("lat");
mapUrl.Append("longitude");
mapUrl.Append("v");
mapUrl.Append("cp");
WebBrowserTask webTask = new WebBrowserTask();
webTask.URL = mapUrl.ToString();
webTask.Show();
```

The Bing Maps Silverlight Control

Alternatively, for a more immersive experience, you can use the Bing Maps Silverlight Control for Windows Phone. This control allows users to search for locations and interact with the map using tap, stretch, pinch, and pan gestures. It also gives you a great deal of control over how the map is displayed, the centered location, the view it displays (such as Aerial, Birdseye, or Road), the zoom level, and much more. In addition, you can handle events exposed by the control within your application to interact with the user if required.

Note: You must register and obtain a key from the Bing Maps Account Center from the Bing page on Microsoft.com (<http://www.microsoft.com/maps/developers/>) in order to use Bing Maps in your applications. If you do not specify a valid key for the Bing Maps control in your application, it displays a message that the credentials are invalid.

As an example, if you have established the current location of the device using a **GeoCoordinateWatcher**, as described earlier in this section, you can display a map using the Bing Maps control embedded in a page of your application. The following code example sets the properties of a Bing Maps control to display a specific location. It sets the zoom level to 12 and displays the controls for zooming in and out and the scale indicator.

C#

```
Location loc = new Location();
loc.Latitude = watcher.Position.Location.Latitude;
loc.Longitude = watcher.Position.Location.Longitude;
loc.Altitude = watcher.Position.Location.Altitude;
double zoom = 12.0;
myMapControl.ScaleVisibility = System.Windows.Visibility.Visible;
myMapControl.ZoomBarVisibility = Visibility.Visible;
myMapControl.SetView(loc, zoom);
```

To use the Bing Maps control, you must reference the corresponding namespaces in your application. The minimum set is **Microsoft.Phone.Controls.Maps**, **Microsoft.Phone.Controls.Maps.Design**, and **Microsoft.Phone.Controls.Maps.Platform**. If you use the extended features of the **MapCore** class that implements the Bing Maps control, you may also require the namespaces **Microsoft.Phone.Controls.Maps.AutomationPeers**, **Microsoft.Phone.Controls.Maps.Core**, and **Microsoft.Phone.Controls.Maps.Overlays**.

For more information about using the Bing Maps control to provide an interactive mapping experience for users of your application, see “Bing Maps Silverlight Control for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff941096\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff941096(VS.92).aspx)).

Media

Windows Phone 7 physical devices can store and display photos, play video and audio files, and have a built-in FM radio. To display photos and images, you can use the standard Silverlight controls such as the **Image** control. You can include the image file in your project and set the **Build Action** property to **Content** and the **Copy to Output Directory** property to **Copy if newer** or **Copy Always**, so that it is

included in the XAP file that uploads to the device. You then specify the image as the source for the control. Alternatively, you can specify a remote URL for the image, and it will be downloaded and displayed just as in a web browser. Images must be in JPEG (.jpg) or PNG (.png) format.

Note: *Avoid including images and other content as resources unless you are building a reusable component that you will distribute. This helps to minimize the size of the application's assembly. Large assemblies have a noticeable effect on performance. For more information about maximizing performance for media on the device, see the section, "Resource Management and Performance," in Chapter 2, "Designing Applications for Windows Phone 7."*

To display a video or play an audio file, you can use the Silverlight **MediaElement** control. You can include the video or audio file in your project and set the **Build Action** property to **Content** and the **Copy to Output Directory** property to **Copy if newer** or **Copy Always**, and then specify it as the source of the control.

XML

```
<MediaElement Source="MyVideo.wmv" Width="300" Height="300"  
AutoPlay="True"/>
```

Alternatively, you can specify a remote URL for the video or audio file, and it will be streamed or downloaded and played just as in a Web browser. For a list of the supported video and audio formats, see "Supported Media Codecs for Windows Phone" on MSDN ([http://msdn.microsoft.com/en-us/library/ff462087\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff462087(VS.92).aspx)). For information about using the **MediaElement** control, see "MediaElement" on MSDN ([http://msdn.microsoft.com/en-us/library/bb980132\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/bb980132(VS.95).aspx)).

An alternative approach for video and audio files is to use the **MediaPlayerLauncher** task. You create an instance of this class, set the required properties, and then call the **Show** method. In this example, the **Build Action** property of the video file is set to **Content** and the **Copy to Output Directory** property is set to **Copy if newer** so that the video file is included in the XAP file uploaded to the device.

C#

```
MediaPlayerLauncher mediaLauncher = new MediaPlayerLauncher();  
mediaLauncher.Controls = MediaPlaybackControls.All;  
mediaLauncher.Location = MediaLocationType.Install;  
mediaLauncher.Media = new Uri("MyVideo.wmv", UriKind.Relative);  
mediaLauncher.Show();
```

Alternatively, you can copy the media item to isolated storage on the device and specify the **MediaLocation** property as **MediaLocationType.Data**. For more information, see “MediaPlayerLauncher Class” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.medialauncher\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.medialauncher(VS.92).aspx)).

Note: When using the **MediaPlayerLauncher** task in the emulator, you may find that video frames render only if you repeatedly click on the UI. Each click advances the video by one frame and then the display turns black.

SELECTING A PHOTO ON THE DEVICE

You can select an existing photo from the user’s media collection using the **PhotoChooserTask** class. You specify a callback handler to execute when the user selects a photo, and optionally the size for the picture that will be returned.

C#

```
PhotoChooserTask photoTask = new PhotoChooserTask();
photoTask.Completed += new EventHandler<PhotoResult>
    (PhotoTask_Completed);
photoTask.PixelHeight = 500;
photoTask.PixelWidth = 500;
photoTask.Show();
```

When the user selects a picture, the callback can obtain the name of the picture and the **Stream** instance pointing to it. You must decode the byte stream into an appropriate image format for display, such as in an **Image** control on the page, as shown here. To use this code, you must add a reference to the namespaces **System.Windows.Media.Imaging** and **Microsoft.Phone** (for the **PictureDecoder** class).

C#

```
void PhotoTask_Completed(object sender, PhotoResult result)
{
    if (result.Error == null && result.TaskResult == TaskResult.OK)
    {
        string fileName = result.OriginalFileName;
        WriteableBitmap thePhoto = PictureDecoder.DecodeJpeg(
            result.ChosenPhoto);
        MyImageControl.Source = thePhoto;
    }
}
```

For more information about using media such as photos, video, and audio on Windows Phone 7, see “Media for Windows Phone” ([http://msdn.microsoft.com/en-us/library/ff402560\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402560(VS.92).aspx)) and “Photos for Windows Phone” ([http://msdn.microsoft.com/en-us/library/ff402553\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402553(VS.92).aspx)) on MSDN.

Search

Your application can use the **SearchTask** class to assist users in finding information on their device and on the web. You simply specify the search string and then show the **SearchTask** instance, as shown in the following code example. Results are displayed and the user can navigate these in the usual way.

C#

```
SearchTask searchTask = new SearchTask();
searchTask.SearchQuery = "Tailspin Surveys";
searchTask.Show();
```

Sound Recording

All Windows Phone 7 physical devices include a microphone. You can access this using the methods of the XNA Framework. This is more complex than using some other features of the phone because you must initiate an asynchronous dispatcher to raise the microphone events.

To record sounds from the microphone, you must do the following:

- Add a reference to the assembly **Microsoft.Xna.Framework** to your project.
- Create an XNA asynchronous dispatcher “pump” class and initialize it in the constructor or startup event in your App.xaml.cs file. For implementation details about this, see Appendix B, “Silverlight and XNA in Windows Phone 7.”
- Get a reference to a microphone.
- Specify a function or lambda expression that will receive the sound data.
- Start the microphone.

Note: An XNA asynchronous event dispatcher is an application service that allows Silverlight to detect events raised by XNA components. Usually, these events are propagated by the XNA game loop mechanism, but this is not present in a Silverlight application. The dispatcher service collects these events and raises

them through the *Silverlight* event mechanism. For an example of how you create and instantiate an XNA asynchronous event dispatcher, see the section, “Using Interop from Silverlight to XNA,” in Appendix B, “Silverlight and XNA in Windows Phone 7.”

In your application, declare variables to hold the microphone instance and a stream that will receive the sound data bytes. This example uses the default microphone and an isolated storage file stream. To use this code, you must reference the namespaces **Microsoft.Xna.Framework.Audio** and **System.IO.IsolatedStorage**.

```
C#
Microphone mic = Microphone.Default;
IsolatedStorageFileStream myFileStream;
```

Specify the callback for the **BufferReady** event and the size of the recording buffer. The maximum size is one second, though lower values tend to give better performance.

```
C#
mic.BufferReady += Mic_BufferReady;
mic.BufferDuration = TimeSpan.FromMilliseconds(500);
```

Open the file stream that will receive the recorded data (in this case, an isolated storage file), and then start the microphone to begin recording.

```
C#
// Create a virtual store and file stream.
// Check for an existing duplicate name.
var myStore = IsolatedStorageFile.GetUserStoreForApplication();
String fileName = "MyAudio.dat";
if (myStore.FileExists(fileName))
{
    myStore.DeleteFile(fileName);
}
myFileStream = myStore.CreateFile(fileName);
mic.Start();
```

The callback for the **BufferReady** event must capture the data from the microphone and store it in your target location.

```
C#
void Mic_BufferReady(object sender, EventArgs e)
{
    Microphone mic = sender as Microphone;
    byte[] content = new byte[mic.GetSampleSizeInBytes
```

```

        (mic.BufferDuration)];
        mic.GetData(content);
        myFileStream.Write(content, 0, content.Length);
    }
}

```

To stop recording, call the **Stop** method of the **Microphone** class and close the stream you are writing to.

C#

```

mic.Stop();
myFileStream.Close();
}

```

You can check the status of the microphone at any time by querying the **State** property of the **Microphone** class. This returns a value from the **MicrophoneState** enumeration. Possible values are **Started** and **Stopped**.

For more information, see “Microphone Class” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.audio.microphone\(XNAGameStudio.40\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.audio.microphone(XNAGameStudio.40).aspx)).

Note: *The files created from the byte stream returned by the **Microphone** class are not valid sound files. If you want to convert a saved byte stream as a .wav or .wmv file, you must use an auxiliary class or library to add the appropriate headers to the file and save it in the appropriate format. For a description of the .wav file format, see “Wave File Format” on The Sonic Spot website (<http://www.sonicspot.com/guide/wavefiles.html>).*

The following series of posts by Microsoft academic developer evangelist Dan Waters describe the techniques of audio programming and provide code you can reuse:

“Intro to Audio Programming, Part 1: How Audio Data is Represented” at <http://blogs.msdn.com/b/dawate/archive/2009/06/22/intro-to-audio-programming-part-1-how-audio-data-is-represented.aspx>

“Intro to Audio Programming, Part 2: Demystifying the WAV Format” at <http://blogs.msdn.com/b/dawate/archive/2009/06/23/intro-to-audio-programming-part-2-demystifying-the-wav-format.aspx>

“Intro to Audio Programming, Part 3: Synthesizing Simple Wave Audio using C#” at <http://blogs.msdn.com/b/dawate/archive/2009/06/24/intro-to-audio-programming-part-3-synthesizing-simple-wave-audio-using-c.aspx>

“Intro to Audio Programming Part 4: Algorithms for Different Sound Waves in C#” at <http://blogs.msdn.com/b/dawate/archive/2009/06/25/intro-to-audio-programming-part-4-algorithms-for-different-sound-waves-in-c.aspx>

Sound Playback

As an alternative to playing sounds using the media controls described in the section “Media” earlier in this chapter, you can use the XNA interop feature to initiate sound playback. This uses two of the XNA framework classes: **SoundEffect** and **SoundEffectInstance**.

To play sounds using the XNA framework classes, you must do the following:

- Add a reference to the assembly **Microsoft.Xna.Framework** to your project.
- Get a reference to an instance of the **SoundEffect** class.
- Use the **SoundEffect** to create a **SoundEffectInstance**.
- Call the **Play** method of the **SoundEffectInstance**.

The simplest approach is to create a **SoundEffect** instance and load it with a stream that points to a valid .wav file, then call the **Play** method of the **SoundEffectInstance** returned from the **CreateInstance** method. The following code example loads a .wav file from isolated storage on the device and plays it once. To use this code, you must reference the namespaces **System.IO** and **System.IO.IsolatedStorage**.

```
C#
IsolatedStorageFile myStore =
    IsolatedStorageFile.GetUserStoreForApplication();
Stream dat = myStore.OpenFile("MyWavFile.wav", FileMode.Open,
                             FileAccess.ReadWrite);
SoundEffect effect = SoundEffect.FromStream(dat);
effect.CreateInstance().Play();
```

If you want to repeat the sound, apply transformations, or specify the volume setting, you can create a **SoundEffectInstance** and set its properties before calling the **Play** method.

```
C#
SoundEffect effect = SoundEffect.FromStream(dat);
SoundEffectInstance sei = effect.CreateInstance();
sei.IsLooped = true;
sei.Pan = (float)0.5;
sei.Pitch = (float)-0.3;
sei.Volume = (float)0.6;
sei.Play();
```

Touch and Gestures

In most cases, your application will use events exposed by the UI controls to detect user input, such as the **Click** event of a **Button** control or the **TextChanged** event of a **TextBox** control. However, you can capture user touch and gestures directly if you want. This may be useful for gestures such as drag, flick, and hold. Windows Phone 7 provides native support for gesture detection using both Silverlight and XNA.

The **TouchPanel** class in the XNA Framework libraries is designed to provide advanced gesture detection capabilities for games and other XNA-based interactive applications. You can use the XNA interop capabilities of Silverlight to implement gesture detection using this approach if you want.

However, for a Silverlight application, you can take advantage of two other simpler approaches to gesture detection. You can use the built-in manipulation mechanism for Silverlight controls, or you can install and use the **GestureListener** control that is included in the Silverlight for Windows Phone Toolkit.

The following table provides a comparison of the approaches you can use.

Mechanism	Advantages	Considerations
Silverlight manipulation events	Requires no additional controls. Part of the phone operating system. Exposes a range of events to detect gestures as they occur or when they complete.	Best suited to tasks that involve moving and scaling objects. Code must translate manipulation values to detect the actual gesture type. Requires additional code to differentiate events such as tap, double-tap, and hold.
GestureListener control	Very easy to use. Exposes a range of events to detect gestures as they occur or when they complete. Automatically differentiates between types of gesture.	Best suited to tasks that require detection and differentiation of all types of event. Provided as a separate assembly under Microsoft Public License. Requires a control in the page.
XNA touch panel	Integrates with XNA game loop. Detects all gesture types.	Requires a dispatcher timer to pass events to Silverlight. More complex to implement in Silverlight.

GESTURE DETECTION USING SILVERLIGHT MANIPULATION EVENTS

The Windows Phone page and most of the UI controls automatically support manipulation events. You can detect manipulation at the individual control level, or for the page as a whole, by handling the three events that occur when manipulation of the control or page starts, each time a movement (a delta) is detected, and when the manipulation ends.

The following code example shows how you can wire up handlers for the three manipulation events of the page.

```
C#
public MainPage()
{
    InitializeComponent();
    this.ManipulationStarted += 
        this.PhoneApplicationPage_ManipulationStarted;
    this.ManipulationDelta += 
        this.PhoneApplicationPage_ManipulationDelta;
    this.ManipulationCompleted += 
        this.PhoneApplicationPage_ManipulationCompleted;
}
```

Each of these events passes a specific type of argument class to the handler. You can extract the values from the arguments to detect the type of movement and the values, as shown in the following code example. Scale values are generated by pinch or stretch gestures and are generally used to resize objects. Translation values are generated by movement over the screen, and are generally used to move objects.

```
C#
// Raised when the gesture starts. Typically used only to detect
// tap events, or to start a timer if you want to differentiate
// tap and double-tap events.
void PhoneApplicationPage_ManipulationStarted(object sender,
                                               ManipulationEventArgs e)
{
    double xStartCoordinate = e.ManipulationOrigin.X;
    double yStartCoordinate = e.ManipulationOrigin.Y;
}

// Raised repeatedly during the gesture. Can be used to
// provide dynamic visual feedback to the user.
void PhoneApplicationPage_ManipulationDelta(object sender,
                                              ManipulationDeltaEventArgs e)
{
```

```
        double xScale = e.DeltaManipulation.Scale.X;
        double yScale = e.DeltaManipulation.Scale.Y;
        double xTranslation = e.DeltaManipulation.Translation.X;
        double yTranslation = e.DeltaManipulation.Translation.Y;
    }

    // Raised when the gesture ends. The original start position
    // is available as well as the relative scale and translation
    // values and information about the gesture velocities.
    void PhoneApplicationPage_ManipulationCompleted(object sender,
                                                    ManipulationCompletedEventArgs e)
{
    double xStartOrigin = e.ManipulationOrigin.X;
    double yStartOrigin = e.ManipulationOrigin.Y;
    double xTotalScale = e.TotalManipulation.Scale.X;
    double yTotalScale = e.TotalManipulation.Scale.Y;
    double xTotalTranslation = e.TotalManipulation.Translation.X;
    double yTotalTranslation = e.TotalManipulation.Translation.Y;
    double xExpansionVelocity =
        e.FinalVelocities.ExpansionVelocity.X;
    double yExpansionVelocity =
        e.FinalVelocities.ExpansionVelocity.Y;
    double xLinearVelocity = e.FinalVelocities.LinearVelocity.X;
    double yLinearVelocity = e.FinalVelocities.LinearVelocity.Y;
}
```

For more information about using the Silverlight manipulation events, see “Gesture Support for Windows Phone” ([http://msdn.microsoft.com/en-us/library/ff967546\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff967546(VS.92).aspx)) and “How to: Handle Manipulation Events” ([http://msdn.microsoft.com/en-us/library/ff426933\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/ff426933(VS.95).aspx)) on MSDN.

GESTURE DETECTION USING THE GESTURELISTENER CONTROL

If you simply need to detect the full range of gestures and the basic position, scale, and translation information, you should consider using the **GestureListener** control that is included in the Silverlight for Windows Phone Toolkit. This control makes it easy to handle the full range of gesture events, including Tap, Double-tap, Hold, Drag, Flick, and Pinch. You can obtain the Silverlight for Windows Phone Toolkit from the Silverlight Toolkit website on CodePlex (<http://silverlight.codeplex.com/>).

To use the **GestureListener** control, you must reference the assembly **Microsoft.Phone.Controls.Toolkit** in your project and add the control to a page in your application. The following example shows the control inserted into a **Grid** that covers the complete page, except for the page title area.

XAML

```
...
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"
      Background="Black">
    <Grid.OpacityMask>
        <SolidColorBrush />
    </Grid.OpacityMask>
    <toolkit:GestureService.GestureListener>
        <toolkit:GestureListener
            Tap="OnTap"
            DoubleTap="OnDoubleTap"
            Hold="OnHold"
            DragStarted="OnDragStarted"
            DragDelta="OnDragDelta"
            DragCompleted="OnDragCompleted"
            Flick="OnFlick"
            PinchStarted="OnPinchStarted"
            PinchDelta="OnPinchDelta"
            PinchCompleted="OnPinchCompleted"/>
    </toolkit:GestureService.GestureListener>
</Grid>
...
```

Notice that the properties of the **GestureListener** allow you to specify individual event handlers for each detected event. The **GestureListener** control effectively wraps the XNA gesture detection mechanism and performs automatic conversion of the XNA events into individual Silverlight events for each gesture type.

The following code example shows the event handlers and the values you can obtain for each type of event detected by the control. Note that the control may raise a combination of events for some gestures. For example, it may raise the Drag events during a flick gesture and when the flick gesture completes.

C#

```
private void OnTap(object sender, GestureEventArgs e)
{
    // Get the screen position relative to the page.
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
```

```
}

private void OnDoubleTap(object sender, GestureEventArgs e)
{
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
}

private void OnHold(object sender, GestureEventArgs e)
{
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
}

// This is raised when a drag gesture starts.
private void OnDragStarted(object sender,
                           DragStartedGestureEventArgs e)
{
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
}

// This is raised repeatedly during a drag gesture.
private void OnDragDelta(object sender, DragDeltaGestureEventArgs e)
{
    double xOriginalPosition = e.GetPosition(this).X;
    double yOriginalPosition = e.GetPosition(this).Y;
    double xMovement = e.HorizontalChange;
    double yMovement = e.VerticalChange;
}

// This is raised when a drag gesture completes.
private void OnDragCompleted(object sender,
                           DragCompletedGestureEventArgs e)
{
    double xOriginalPosition = e.GetPosition(this).X;
    double yOriginalPosition = e.GetPosition(this).Y;
    double xMovement = e.HorizontalChange;
    double yMovement = e.VerticalChange;
    double xVelocity = e.HorizontalVelocity;
    double yVelocity = e.VerticalVelocity;
}
```

```
// This is raised in conjunction with Drag events
// after a flick gesture completes.
private void OnFlick(object sender, FlickGestureEventArgs e)
{
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
    double angle = e.Angle;
    double xVelocity = e.HorizontalVelocity;
    double yVelocity = e.VerticalVelocity;
}

// This is raised when a pinch or zoom gesture starts.
private void OnPinchStarted(object sender,
    PinchStartedGestureEventArgs e)
{
    double xPosition = e.GetPosition(this).X;
    double yPosition = e.GetPosition(this).Y;
    double distance = e.Distance;
    double angle = e.Angle;
}

// This is raised repeatedly during a pinch or zoom gesture.
private void OnPinchDelta(object sender, PinchGestureEventArgs e)
{
    double xOriginalPosition = e.GetPosition(this).X;
    double yOriginalPosition = e.GetPosition(this).X;
    double distance = e.DistanceRatio;
    double angle = e.TotalAngleDelta;
}

// This is raised when a pinch or zoom gesture completes.
private void OnPinchCompleted(object sender,
    PinchGestureEventArgs e)
{
    double xOriginalPosition = e.GetPosition(this).X;
    double yOriginalPosition = e.GetPosition(this).X;
    double distance = e.DistanceRatio;
    double angle = e.TotalAngleDelta;
}
```

For more information and to download the Silverlight for Windows Phone Toolkit, see the Silverlight Toolkit page on CodePlex (<http://silverlight.codeplex.com/>).

GESTURE DETECTION USING XNA

To detect touch and gestures using the XNA Framework classes, you must do the following:

- Add a reference to the assemblies **Microsoft.Xna.Framework** and **Microsoft.Xna.Framework.Input.Touch** to your project.
- Set the properties of the static **TouchPanel** class to specify the types of gestures you want to capture.
- Start a **DispatcherTimer** that will raise an event at appropriate intervals.
- Handle the event raised by the **DispatcherTimer** to obtain the gesture information.

The following code example specifies that the **TouchPanel** will collect all gestures except for horizontal drag and vertical drag (which constrain the results in one direction). It then creates a **Dispatcher Timer** with an interval of 50 milliseconds, though you can vary this depending on how often you want to read gesture information, adds the event handler, and starts the timer. To use this code, you must reference the namespace **System.Windows.Threading**.

C#

```
TouchPanel.DisplayHeight = 800;
TouchPanel.DisplayWidth = 480;
TouchPanel.EnabledGestures = GestureType.Tap |
    GestureType.DoubleTap |
    GestureType.Hold |
    GestureType.FreeDrag |
    GestureType.Flick |
    GestureType.Pinch;
var touchTimer = new DispatcherTimer();
touchTimer.Interval = TimeSpan.FromMilliseconds(50);
touchTimer.Tick += new EventHandler(Read_Gestures);
touchTimer.Start();
```

In the event raised by the timer, you can read each gesture. However, each gesture will consist of several individual components; for example, a double tap will result in detection of both a tap and a double tap gesture, so your code must examine the sequence of gestures to determine what action to take.

The following code example of a gesture detection event handler shows how you obtain information about each gesture that was detected. It indicates the values that are available for each of the gesture types. Remember that the event handler runs on a different thread from the UI, so you must invoke a method on the UI thread if you want to update the UI with the discovered values.

```
C#
public void Read_Gestures(object sender, EventArgs e)
{
    float xCoordinate = 0;
    float yCoordinate = 0;
    float xStartCoordinate = 0;
    float yStartCoordinate = 0;
    float xEndCoordinate = 0;
    float yEndCoordinate = 0;
    float xSecondFingerStartCoordinate = 0;
    float ySecondFingerStartCoordinate = 0;
    float xSecondFingerEndCoordinate = 0;
    float ySecondFingerEndCoordinate = 0;
    float xFlickSpeed = 0;
    float yFlickSpeed = 0;

    while (TouchPanel.IsGestureAvailable)
    {
        GestureSample gesture = TouchPanel.ReadGesture();
        switch (gesture.GestureType)
        {
            case GestureType.Tap:
                xCoordinate = gesture.Position.X;
                yCoordinate = gesture.Position.Y;
                break;
            case GestureType.DoubleTap:
                xCoordinate = gesture.Position.X;
                yCoordinate = gesture.Position.Y;
                break;
            case GestureType.Hold:
                xCoordinate = gesture.Position.X;
                yCoordinate = gesture.Position.Y;
                break;
            case GestureType.FreeDrag:
                xEndCoordinate = gesture.Position.X;
                yEndCoordinate = gesture.Position.Y;
                xStartCoordinate = xEndCoordinate - gesture.Delta.X;
                yStartCoordinate = yEndCoordinate - gesture.Delta.Y;
                // For GestureType.HorizontalDrag, Delta.Y
                // will always be zero.
                // For GestureType.VerticalDrag, Delta.X
                // will always be zero.
                break;
            case GestureType.Flick:
```

For more information about using XNA to detect gestures, see “Working with Touch Input (Windows Phone)” on MSDN (<http://msdn.microsoft.com/en-us/library/ff434208.aspx>).

Vibration Alerts

Application code can initiate the vibration capability of the device to alert users of events occurring in the application. To use this feature, you must do the following:

- Add a reference to the namespace **Microsoft.Devices** to your class.
 - Call the static methods of the default **VibrateController** class to start and stop vibration.

To initiate vibration for the default duration, call the **Start** method. To specify the duration, include a **TimeSpan** that defines the duration of the vibration as the parameter of the **Start** method. To stop vibration, call the **Stop** method.

```
C#  
VibrateController.Default.Start(); // Default duration  
VibrateController.Default.Start(TimeSpan.FromSeconds(2));  
VibrateController.Default.Stop();
```

Web Browser

You can invoke the Windows Phone 7 web browser in your applications using the **WebBrowserTask** class. You simply specify the URL to open, and call the **Show** method. The following code example shows how to do this.

C#

```
WebBrowserTask browserTask = new WebBrowserTask();
browserTask.URL = "http://www.microsoft.com/windowsmobile";
browserTask.Show();
```

For more information, see “[WebBrowserTask Class](#)” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.webbrowsertask\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.webbrowsertask(VS.92).aspx)).

Windows Marketplace

Windows Phone 7 devices obtain applications through the Windows Marketplace hub. This is available from the Start screen, but you can invoke the hub in your code in a variety of ways. You can open the hub to show specific types of content such as applications, music, or podcasts. You can also open the hub showing a filtered list of items from one of these categories using a search string, or you can just show a specific item by specifying its GUID content identifier. Finally, you can open the Reviews screen.

The following code examples show how you can use the Windows Marketplace task launchers in your application code.

C#

```
// Open the Windows Marketplace hub to show all applications.
MarketplaceHubTask marketPlace = new MarketplaceHubTask();
marketPlace.ContentType = MarketplaceContentType.Applications;
// Other options are MarketplaceContentType.Music
// and MarketplaceContentType.Podcasts.
marketPlace.Show();

// -----
// Open the Windows Marketplace hub to search for
// applications using a specified search string.
MarketplaceSearchTask marketSearch = new MarketplaceSearchTask();
marketSearch.ContentType = MarketplaceContentType.Applications;
marketSearch.SearchTerms = "Tailspin Surveys";
marketSearch.Show();
```

```
// -----  
  
// Open the Windows Marketplace hub with a specific  
// application selected for download and installation.  
MarketplaceDetailTask marketDetail = new MarketplaceDetailTask();  
marketDetail.ContentType = MarketplaceContentType.Applications;  
marketDetail.ContentIdentifier  
    = "{12345678-1234-1234-1234-123456789abc}";  
marketDetail.Show();  
  
// -----  
  
// Open the Windows Marketplace hub Review page.  
MarketplaceReviewTask marketReview = new MarketplaceReviewTask();  
marketReview.Show();
```

If you call the **Show** method of the **MarketplaceDetailTask** class without specifying a value for the **ContentIdentifier** property, the task will show the Windows Marketplace page for the current application.

For more information about the tasks described in this section, and the other types of tasks available on Windows Phone 7 devices, see “Microsoft.Phone.Tasks Namespace” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.tasks(VS.92).aspx)).

If you want to provide a link to a specific product on Windows Marketplace within a Web page or another application, you can do so by specifying the appropriate product ID in the link. The following shows the format of a link to an application on Windows Marketplace.

```
http://social.zune.net/redirect?type=phoneApp&id=12345678-1234-  
1234-1234-123456789abc&source=MSDN
```

Reactive Extensions

When you work with device features that depend on sequences of data, such as the Accelerometer or the Location Service, you must often manage multiple concurrent streams of information. Other scenarios in which the same requirements occur are when composing information returned from multiple concurrent requests to remote services, or when you need to handle inputs such as recording sound. In addition, if you are building and testing applications using an emulator instead of a physical device, some of the device capabilities (such as the Accelerometer) are not available. In these situations, you may find the Reactive Extensions (Rx) for .NET feature useful.

The reactive extensions also allow you to write compact, declarative code to manage complex, asynchronous operations.

Reactive extensions allow you to do the following:

- Represent data streams, asynchronous requests, and events as observable sequences to which applications and components can subscribe.
- Compose multiple streams into a single stream.
- Define queries that filter streams to select only specific data items.
- Apply transformations to data streams.
- Emulate data streams that are not available in the current environment (such as a device emulator).
- Read and write the streams to storage on the device, and play them back when required.

The majority of operations you perform with reactive extensions are implemented by the **Observable** class, which exposes a large number of methods for creating, merging, manipulating, and querying streams. For a full list of these methods, see “Observable Members” on MSDN ([http://msdn.microsoft.com/en-us/library/microsoft.phone.reactive.observable_members\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/microsoft.phone.reactive.observable_members(VS.92).aspx)).

For a description of how the reactive extensions are used in the Tailspin application, see Chapter 5, “Using Services on the Phone.”

For more information about reactive streams, see “Reactive Extensions for .NET Overview for Windows Phone” ([http://msdn.microsoft.com/en-us/library/ff431792\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431792(VS.92).aspx)) and “How to: Use Reactive Extensions to Emulate and Filter Location Data for Windows Phone” ([http://msdn.microsoft.com/en-us/library/ff637517\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff637517(VS.92).aspx)) on MSDN. You can also download a code sample that demonstrates using the reactive extensions from “Code Samples for Windows Phone” on MSDN ([http://msdn.microsoft.com/en-us/library/ff431744\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff431744(VS.92).aspx)).

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Appendix D

Prism Library for Windows Phone 7

Prism is a free utility library from the Microsoft® patterns & practices group. The components in this library can help developers build applications for Windows® Presentation Foundation (WPF), Microsoft Silverlight®, and the Windows Phone 7 platform that are easier to maintain and update as requirements change.

Prism was originally built to support composite application scenarios, where you might typically have a shell application and modules that contribute pieces to the overall application. Although Prism supports this application style, most of the library components offered by Prism can be useful in building any WPF or Silverlight application. For example, Prism offers components to support the Model-View-ViewModel (MVVM) pattern as well as pieces to provide loosely coupled communication between parts of an application.

Although the use of a display shell is typically not appropriate in a Windows Phone 7 application, many of the other components within Prism are useful when building Silverlight applications for Windows Phone 7. For example, you can use the **DelegateCommand** provided by Prism to avoid the requirement to implement event handlers in the code-behind of your views when using the MVVM pattern for your applications.

Prism includes a small library known as the Prism Library for Windows Phone 7, which contains a subset of the main Prism Library features specifically aimed at helping developers to implement solutions to common issues found in developing applications for Windows Phone 7. The library includes classes to help developers implement commands, navigation, observable object notifications, data template selection, interaction with notifications, interaction with the application bar, and more when building applications for Windows Phone 7.

This appendix provides an overview of the Prism Library for Windows Phone 7, which is used in the Tailspin application for Windows Phone 7 discussed in this guide. For more information and to download Prism, see the Prism home page on MSDN® (<http://www.microsoft.com/prism>). To provide feedback, get assistance, or download additional content, visit the Prism community site on CodePlex (<http://prism.codeplex.com>).

About Prism for Windows Phone 7

Windows Phone 7 applications implemented using Silverlight are naturally suited to the MVVM pattern, which discourages the use of code-behind in the views in favor of handling events and activity in the view model. However, many common scenarios—such as binding commands to interface objects, linking methods to application bar buttons, notifying of changes to object properties, and detecting changes to text-based controls in the view—are challenging to accomplish without using code-behind. The helper classes and components in Prism Library for Windows Phone 7 are specially designed to simplify these tasks in Silverlight applications created for Windows Phone 7.

In addition, the Prism Library for Windows Phone 7 includes helper classes for publishing and subscribing to events, displaying notification messages, and selecting data templates at run time. Many of these components and helper classes are used in the Tailspin. PhoneClient project. You can download the complete Tailspin solution for use in conjunction with this guide from the patterns & practices Windows Phone 7 Developer Guide community site on Code-Plex (<http://go.microsoft.com/fwlink/?LinkId=205602>).

The Prism Library for Windows Phone 7 is provided as source code in the two projects Microsoft.Practices.Prism and Microsoft.Practices.Prism.Interactivity within the Tailspin solution. These solutions implement the Prism Library for Windows Phone 7 library, and there is also a set of tests to help you explore the use of the classes in the library. These tests can also be used if you extend or modify the source code to meet your own specific requirements.

Contents of Prism for Windows Phone 7 Library

Prism Library for Windows Phone 7 contains several namespaces that subdivide the artifacts:

- **Microsoft.Practices.Prism.** This namespace contains classes concerned with detecting and reacting to change events for properties and objects.
- **Microsoft.Practices.Prism.Commands.** This namespace contains classes concerned with binding commands to user interface (UI) objects without requiring the use of code-behind in the view model, and with composing multiple commands.

- **Microsoft.Practices.Prism.Events**. This namespace contains classes concerned with subscribing to events and with publishing events on the publisher thread, UI thread, or a background thread.
- **Microsoft.Practices.Prism.ViewModel**. This namespace contains classes that help support implementing the view-model portion of the MVVM pattern, such as displaying a template at run time and simplifying the implementation of property change notification.
- **Microsoft.Practices.Prism.Interactivity**. This namespace contains classes and custom behaviors concerned with handling interaction and navigation for application bar buttons and with updating the values of view model properties bound to text and password controls.
- **Microsoft.Practices.Prism.Interactivity.InteractionRequest**. This namespace contains classes concerned with displaying notifications to users.

The following tables list the main classes in these namespaces and provide a brief description of their usage. They do not include all the classes in each namespace; they list only those that implement the primary functions of the library. For a full reference to all the Prism namespaces, see “Prism (Composite Client Application Guidance)” on MSDN (<http://www.microsoft.com/prism>).

MICROSOFT.PRACTICES.PRISM NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism** namespace. These classes are not used in the Tailspin application.

Class	Description
ObservableObject <i><T></i>	A class that wraps an object so that other classes can be notified of change events. Typically, this class is set as a dependency property on dependency objects and allows other classes to observe any changes in the property values. This class is required because, in Silverlight, it is not possible to receive change notifications for dependency properties that you do not own.
Extension Methods <i>(for List<T>)</i>	This class adds the RemoveAll method to List<T> . The method removes the all the elements that match the conditions defined by the specified predicate.

MICROSOFT.PRACTICES.PRISM.COMMANDS NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism.Commands** namespace.

Class	Description	Usage in Tailspin application
CompositeCommand	This class composes one or more ICommand implementation instances. It forwards to all the registered commands, and returns true if all the commands return true .	This class is not used in the Tailspin application.
DelegateCommand	This class is an implementation of ICommand . Its delegates can be attached to access the Execute and CanExecute methods.	Used in most of the view models for associating commands with actions that execute in response to a command being fired. For more information, see the section, "Commands," in Chapter 4, "Building the Mobile Client."

For more information about using the **DelegateCommand** and **CompositeCommand** classes, see the section, "MVVM," in "Prism (Composite Application Guidance)" on MSDN (<http://www.microsoft.com/prism>).

MICROSOFT.PRACTICES.PRISM.EVENTS NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism.Events** namespace. Classes in this namespace support decoupled communication between pieces of an application, such as between two view models. These classes are not used in the Tailspin application.

Class	Description
CompositePresentationEvent	A class that manages publication and subscription to events.
SubscriptionToken	The subscription token returned from an event subscription method. This class provides methods to compare tokens.
EventSubscription	This class provides a method for retrieving a Delegate that executes an Action depending on the value of a second filter predicate that returns true if the action should execute.

BackgroundEventSubscription	This class extends EventSubscription to invoke the Action delegate on a background thread.
DispatcherEventSubscription	This class extends EventSubscription to invoke the Action delegate in a specific Dispatcher instance.
EventAggregator	This class implements a service that stores event references and exposes the GetEvent method to retrieve a specific event type.

For more information about publishing and subscribing to events, see the section, “Communicating between Loosely Coupled Components,” in “Prism (Composite Application Guidance)” on MSDN (<http://www.microsoft.com/prism>).

MICROSOFT.PRACTICES.PRISM.VIEWMODEL NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism.ViewModel** namespace.

Class	Description	Usage in Tailspin application
DataTemplateSelector	This class implements a custom ContentControl that changes its ContentTemplate based on the content it is presenting. To determine the template it must use for the new content, the control retrieves it from its resources using the name for the type of the new content as the key.	Used in the TakeSurveyView view to select the appropriate question view, depending on the type of question. For more information, see the section, “Data Binding and the Panorama Control,” in Chapter 4, “Building the Mobile Client.”
NotificationObject	This is a base class for items that support property notification. It provides basic support for implementing the INotifyProperty Changed interface and for marshalling execution to the UI thread.	Used in the ViewModel , QuestionViewModel , QuestionOption , and QuestionAnswer classes. For more information, see the section, “Displaying Data,” in Chapter 4, “Building the Mobile Client.”

For more information about using the **DataTemplateSelector** class, see “MVVM QuickStart” in “Prism (Composite Application Guidance)” on MSDN (<http://www.microsoft.com/prism>).

MICROSOFT.PRACTICES.PRISM.INTERACTIVITY NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism.Interactivity** namespace.

Class	Description	Usage in Tailspin application
ApplicationBarButton-Command	This class implements a behavior that associates a command with an application bar button.	Used in the SurveyListView, TakeSurveyView, AppSettingsView, and FilterSettingsView views to bind the click events for application bar buttons to methods in the view models. For more information, see the section, “Commands,” in Chapter 4, “Building the Mobile Client.”
ApplicationBarButton-Navigation	This class implements a behavior that subscribes to an application bar button click event and navigates to a specified page.	Used in the FilterSettingsView view to initiate navigation to the previous page when the Cancel button is clicked. For more information, see the section, “Handling Navigation Requests,” in Chapter 4, “Building the Mobile Client.”
AppBar-Extensions	This class provides the FindButton method to find a button on the phone’s application bar.	This class is not used directly in the Tailspin application.
UpdatePassword-BindingOn-PropertyChanged	This class implements a behavior that updates the source of a binding on a password box as the text changes. By default in Silverlight, the bound property is only updated when the control loses focus.	Used in the AppSettingsView view to ensure that the user’s password input is updated in the view model even if the password box does not lose focus. For more information, see the section, “Displaying Data,” in Chapter 4, “Building the Mobile Client.”

UpdateTextBindingOnPropertyChanged	This class implements a behavior that updates the source of a binding on a text box as the text changes. By default in Silverlight, the bound property is only updated when the control loses focus.	Used in the AppSettingsView and OpenQuestionView views to ensure that the user's text input is updated in the view model even if the text box does not lose focus. For more information, see the section, "Displaying Data," in Chapter 4, "Building the Mobile Client."
---	--	--

MICROSOFT.PRACTICES.PRISM.INTERACTIVITY. INTERACTIONREQUEST NAMESPACE

The following table lists the main components in the **Microsoft.Practices.Prism.Interactivity.InteractionRequest** namespace.

Class	Description	Usage in Tailspin application
InteractionRequest	This class represents a request for user interaction. View models can expose interaction request objects through properties and raise them when user interaction is required so that views associated with the view models can materialize the user interaction using an appropriate mechanism.	Used in the SurveyList-ViewModel, AppSettingsViewModel, and FilterSettingsViewModel view models to generate interaction requests that display notification messages. For more information, see the section, "User Interface Notifications," in Chapter 4, "Building the Mobile Client."
MessageBoxRequestTrigger	This class displays a message box as a result of an interaction request.	Used in the SurveyListView, AppSettingsView, and FilterSettingsView views to display message boxes. For more information, see the section, "User Interface Notifications," in Chapter 4, "Building the Mobile Client."

ToastRequestTrigger	This class displays a pop-up toast item with specified content as a result of an interaction request. After a short period, it removes the pop-up window.	Used in the SurveyListView view to display information about the most recent synchronization with the remote service. For more information, see the section, “User Interface Notifications,” in Chapter 4, “Building the Mobile Client.”
----------------------------	---	--

For more information about interaction requests and displaying notifications, see “MVVM Advanced Scenarios” in “Prism (Composite Application Guidance)” on MSDN (<http://www.microsoft.com/prism>).

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Appendix E Microsoft Sync Framework and Windows Phone 7

Application models based on traditional client-server and web-based technologies tend to have little need for synchronizing data between locations. In most cases, all instances of clients—such as Windows® Forms applications or web browsers—access a single store that holds the data used by the application (although this store may itself be a server farm). However, the increasing use of cloud-based data stores and mobile devices means that the capability to synchronize or replicate data across separate locations and devices is becoming even more vital.

Synchronization between databases has been a common requirement for a while, and technologies to achieve this are available and widely used. However, they are often closely tied to a specific scenario and do not support more general client application scenarios. The following are some examples of the capabilities that are becoming essential for distributed applications:

- Synchronizing data between geographically separated cloud-based data repositories and applications
- Synchronizing data between one or more on-premises databases or applications and one or more cloud-based databases or applications
- Synchronizing data between rich client applications and the data source to maximize interactivity and usability of the application
- Synchronizing data between mobile and occasionally connected devices and the application data store to allow off-line operation

Some of these requirements can be satisfied using relational database replication and synchronization technologies, generally where there is a database available on the client computer (which, in some cases, may be SQL Server® Compact Edition or another file-based database mechanism). However, there is an increasing requirement to synchronize data for devices and clients that do not have a local database mechanism available. Typical examples of this are mobile phones and similar small form factor devices.

The Microsoft® Sync Framework aims to provide an extensible and easy-to-use mechanism for synchronizing data between almost any type of data source and client. In the context of this guide, it specifically enables synchronization between Windows Azure™ technology platform services and Windows Phone 7 devices.

About the Microsoft Sync Framework

The Microsoft Sync Framework version 3.0 is designed to make it easy to allow synchronization of databases (including complete tables and individual rows), file system content, and arbitrary data in a range of scenarios. The following are some of these synchronization scenarios:

- Between on-premises databases and single or multiple cloud databases
- Between multiple on-premises databases via the cloud (“data hub in the sky”)
- Between multiple cloud databases
- Between remote data store(s) and client applications
- Between data stores and Microsoft Excel® spreadsheet software (Pivot) or other Microsoft Office applications such as Microsoft SharePoint® team services, Exchange Server, and other enterprise solutions
- To populate remote databases from on-premises databases
- To aggregate data from multiple remote databases to on-premises databases
- To maintain a consistent view of data across “three screens” (mobile, desktop, and cloud)
- To allow reuse of the same application model and logic with just a different user interface (UI) for each client type
- To enable simple development of occasionally-connected (“offline-and-sync”) clients

Note: The Sync Framework exposes changes to data using the OData Sync protocol. This is based on the Open Data (OData) protocol. OData allows a wide range of data sources to be exposed and consumed over the web in a simple, secure, and interoperable format. It uses well-established standards and web technologies such as XML, HTTP, Atom Publishing (Atom Pub), and JavaScript Object Notation (JSON). For information about OData, see the Developers page on the Open Data Protocol website (<http://www.odata.org/developers>). For a list of OData providers and tools, see the OData SDK page on the Open Data Protocol website (<http://www.odata.org/developers/odata-sdk>).

Figure 1 shows an overview of how the Sync Framework can be used in a Windows Azure service to synchronize data with different types of clients. The service exposes synchronization endpoints to which clients can connect. The way that the synchronization occurs depends on the type of client, and the synchronization protocols it supports. The synchronization is managed by the Sync Framework itself, and can optionally include custom business logic to perform tasks such as authentication, authorization, and management.

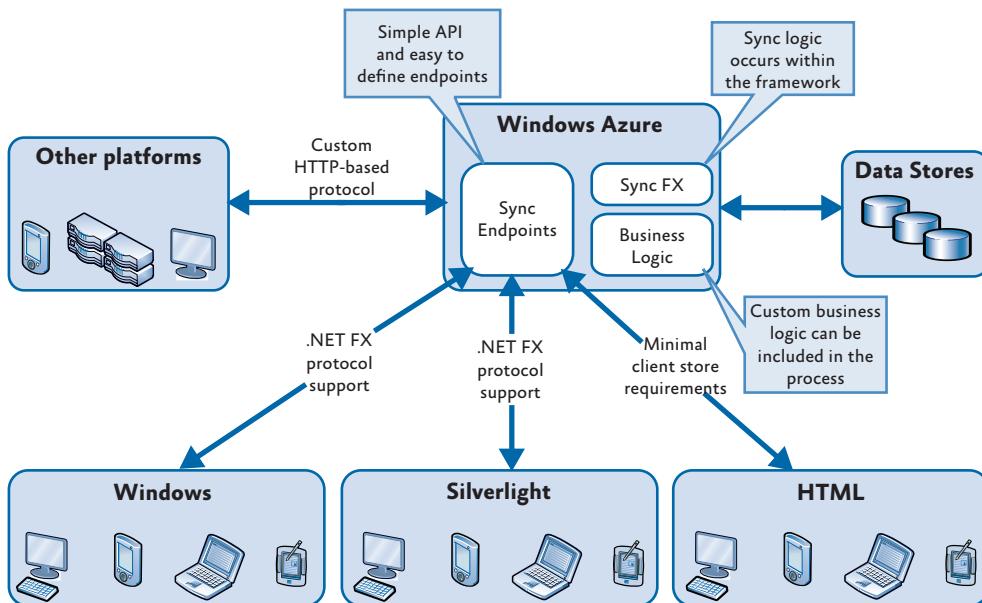


FIGURE 1
Overview of the Sync Framework capabilities

COMPONENTS OF THE SYNC FRAMEWORK

To achieve the required flexibility, the architecture of the Sync Framework consists of a central orchestration mechanism, a small synchronization runtime for each client, and individual pluggable providers for each of the data stores and client types.

In many cases, the synchronization runtime and provider can run on the client; this enables full integration with the sync framework as well as the capability for peer-to-peer synchronization. Code on the client can access the functionality of the Sync Framework using the simple API available in the provider runtime to synchronize with another data source, send changes to that data source, and access data in the data source that has changed since the last synchronization. The mechanism also allows clients and data stores to specify rules on how to resolve conflicts. Figure 2 shows a schematic of this process.

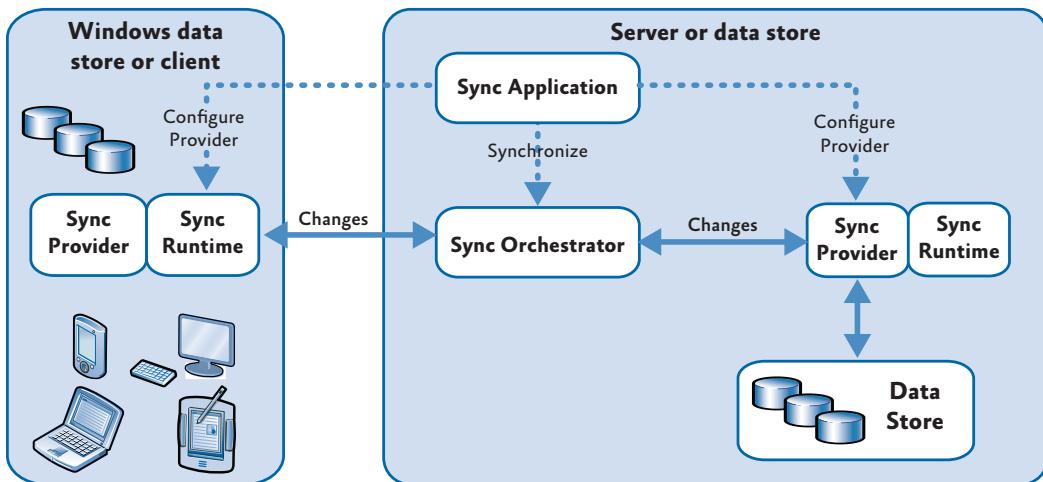
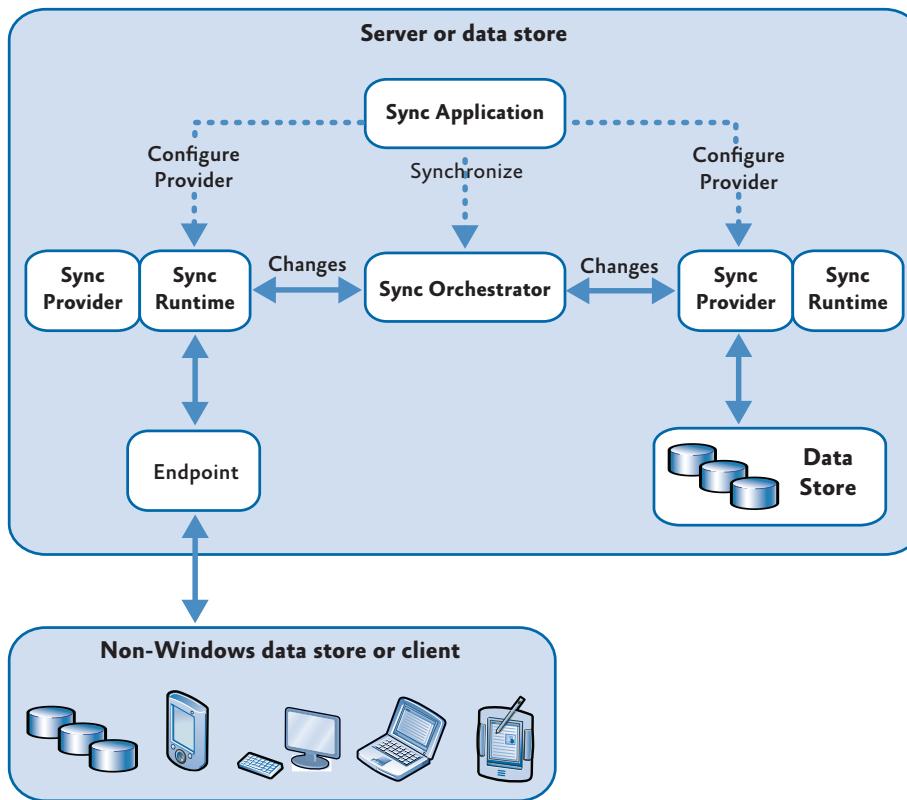


FIGURE 2

The components and process for synchronization with Windows clients

In cases in which the synchronization provider cannot execute on the client, such as with non-Windows devices or web browsers, developers can write code that accesses the provider on the remote data store using the OData Sync protocol to synchronize data, send updates, and get changes from the remote data store.

The server (in this example, running in Windows Azure) specifies an endpoint that exposes changes using the OData Sync protocol. Typically, this is a Windows Communication Foundation (WCF) service. The client may use a runtime and provider that understands the OData Sync protocol, or—where this is not available or practical—it can use custom code to read and parse the OData Sync information. Figure 3 shows a schematic of this approach.

**FIGURE 3**

The components and process for synchronization with non-Windows clients

The main advantage is that there is now a standard way to read and submit changes for synchronization between the data store, this client device, and other devices that use the same set of data.

SYNC FRAMEWORK PROVIDERS

Some providers are still under development, and others will be added to the list in the future. At present, the range of providers available, or soon to be available, includes the following:

- SQL Server using tabular data stream (TDS) protocol over HTTP(S) and a wizard in SQL Server 2008 R2
- SQL Server Compact Edition over HTTP(S)
- SQL Azure™ technology platform using the TDS protocol over HTTP(S) and a wizard in SQL Server 2008 R2
- Azure Web Roles using an HTTP(S) endpoint with access to Azure table and binary large object (BLOB) storage

- Silverlight synchronization to isolated storage using HTTP(S) to synchronize data stored as collections
- Windows Phone 7 synchronization to isolated storage using HTTP(S) to synchronize data stored as collections
- Windows Mobile 6.x support over HTTP(S) to SQL Compact Edition
- Synchronization to HTML 5 clients over HTTP(S) (coming in a future release)
- Synchronization to any existing HTTP-enabled client using HTTP(S) with a custom proxy and code
- File-based synchronization across computers and networks using standard network protocols

For more information about the Sync Framework, see the Microsoft Sync Framework Developer Center on MSDN® (<http://msdn.microsoft.com/en-us/sync/default.aspx>).

Using the Sync Framework

To perform synchronization, the developer writes code that does the following:

- Initiates a synchronization session
- Sets up references to two synchronization providers and configures them appropriately
- Starts the synchronization process

Each client or server store holds information about the changes occurring in that store. This metadata, stored local to and available to each provider, contains the knowledge used by the provider to detect changes and conflicts. The Sync Framework includes a small footprint metadata store service that developers can use when implementing custom providers. Alternatively, developers can implement the metadata functionality as part of the local data store. An example is the offline store provider for Silverlight web applications and Windows phone 7 applications.

Note: *Data converters are required if the format of the data is different between the two data stores, and if the existing providers or your custom providers do not expose data in OData Sync format. For example, if one store exposes ADO.NET DataSets and the other exposes data in XML format, the developer can create a pair of data converters and specify these when setting up the synchronization process. If there are many data store formats, it may be easier to create data converters that convert to and from a common format, and then to use the appropriate pair for each synchronization session.*

Events are raised during the synchronization process to indicate the individual changes occurring and any errors that arise (including conflicts if the data in both locations has changed). The application can handle these events to provide status information to the user, manage conflicts by applying business rules (such as “latest change wins”), and to display the results.

For a useful introduction to the synchronization process and details of the way that synchronization metadata is used, see “Introduction to Microsoft Sync Framework” on MSDN (<http://msdn.microsoft.com/en-us/sync/bb821992.aspx>). This article discusses the metadata used by the framework, the synchronization flow, and includes examples of synchronization with and without update conflicts.

Synchronization for Windows Azure and Windows Phone 7

Version 3.0 of the Sync Framework includes providers for Windows Azure and Windows Phone 7 that can exchange synchronization messages using the OData Sync protocol. The samples available for the Sync Framework include examples that use these providers. The Windows Phone 7 provider uses isolated storage on the device to hold the local data and the synchronization metadata as a series of collections in the form of a local context.

The Sync Framework includes a utility named SyncSvcUtil that developers can use to create both a set of entities that extend the **IsolatedStorageOfflineEntity** base class to define the data types, and a class that extends the **IsolatedStorageOfflineContext** class to interact with stored data.

The **IsolatedStorageOfflineEntity** base class exposes properties used by the Sync Framework provider to define and monitor changes to the local data. For example, it exposes the entity state (such as Unmodified, Modified, and Saved) if the change encountered a conflict or error or if the local entity is a tombstone (because it was deleted on the device). The **IsolatedStorageOfflineEntity** type also exposes events that indicate when the entity value changes and a method to reject the changes and restore the original version.

The concrete type created by the SyncSvcUtil that extends **IsolatedStorageOfflineEntity** exposes application-specific entity types based on the schema of the data source with which the device will synchronize. The application on the device uses these entity types when accessing and interacting with the data that is stored locally.

The **IsolatedStorageOfflineContext** base class exposes properties, methods, and events that manage the synchronization process and indicate its progress when synchronization is occurring. These

include methods to initiate a session, add and remove items, and initiate synchronization; properties that expose collections of errors and conflicts; and events that indicate the progress and result of the process.

The concrete type created by the SyncSvcUtil utility that extends **IsolatedStorageOfflineContext** exposes the schema for the stored data, collections of the stored entities, and methods that allow the application to add and delete items in the collections. The application on the device uses these methods and properties to display and manipulate the data that is stored locally.

To learn more about using the Sync Framework with Windows Azure and Windows Phone 7, see the following resources:

- “Synchronization” on MSDN:
[http://msdn.microsoft.com/en-us/library/dd938837\(SQL.10\).aspx](http://msdn.microsoft.com/en-us/library/dd938837(SQL.10).aspx)
- Sync Framework Team Blog:
<http://blogs.msdn.com/b/sync/>

These and all links in this book are accessible from the book’s online bibliography. The URL to the bibliography can be found in the preface, in the final section, entitled, “Where to Go for More Information.”

Answers to Questions

CHAPTER 2

DESIGNING WINDOWS PHONE 7 APPLICATIONS

1. Which of the following should you consider when designing the UI for a Windows Phone 7 application?

- a. Making it work in both portrait and landscape mode.
- b. Making it colorful and highly configurable.
- c. Maximizing the number of controls on each page.
- d. Reusing an existing web or desktop application UI.

Answer: Only “a” is correct. If the device has a hardware keyboard, it may require use of the device in landscape mode. You should also follow the built-in Metro theme and colors, simplify the UI as far as possible, and avoid trying to reuse existing non-phone UI. Phone applications are generally used in a different kind of environment from desktop and web applications, where there are often interruptions and it is harder for users to concentrate on the application. The small screen and the use of finger-touch input require a simplified UI to prevent errors.

2. Which of the following languages and frameworks can be used to create Windows phone 7 applications?

- a. Microsoft® Silverlight®.
- b. C++.
- c. XNA®.
- d. Microsoft Visual Basic®.

Answer: Only “a” and “c” are correct. In the current release, Microsoft Visual Basic development system is not supported. Although OEMs are able to use native code on the phone, application developers cannot use C++.

3. Which of the following are **not** events that may occur as part of the tombstoning and reactivation process for a Windows Phone 7 application?

- a. Activated.
- b. Loading.
- c. Resuming.
- d. Deactivated.

Answer: Only “**a**”, “**b**”, and “**d**” are events that may occur as part of the tombstoning and reactivation process. The other event that may occur is *Closing*.

4. Which of the following ways would you consider storing the data used by an application?

- a. Using isolated storage on the phone.
- b. Using the built-in SQL ServerTM Compact Edition database on the phone.
- c. Using the file system on the phone.
- d. Storing it on the server and synchronizing it with the phone.

Answer: Only “**a**” and “**d**” are valid. There is no built-in database or file system in Windows Phone 7 devices. Storing data on the server is a useful approach, but you must consider how the application will work when there is no network connectivity.

5. Which of the following techniques should you consider for securing a Windows Phone 7 application?

- a. Never store any data on the phone.
- b. Encrypt all sensitive data stored on the phone.
- c. Use HTTPS for all requests containing sensitive data when communicating with the server.
- d. Perform a security analysis of your design and the code.

Answer: With the possible exception of “**a**”, all of these are correct. If your application does not require access to any local data, then “**a**” is also appropriate, but this is an unlikely scenario in most business applications!

6. Which of the following are useful techniques for maximizing performance of a Windows Phone 7 application?

- a. Turn off features such as the Location service if you do not require them, or turn them off as soon as possible after using them.
- b. Use single-letter names for variables in your code.
- c. Optimize your code using good coding practice and simplifications where appropriate.
- d. Make full use of the Graphics Processing Unit (GPU) by avoiding code that cannot run on it.

Answer: Only “**a**”, “**c**”, and “**d**” are valid ways to maximize performance of Windows Phone 7 applications. All .NET Framework source code is compiled to Microsoft intermediate language (MSIL) code that does not include the variable names. Applications must be compiled in Release mode for distribution and will not contain debugging symbols.

CHAPTER 4

BUILDING THE MOBILE CLIENT

1. Which of the following are good reasons to use the MVVM pattern for your Windows Phone 7 application?
 - a. It improves the testability of your application.
True. This is a valid reason for using the MVVM pattern.
 - b. It facilitates porting the application to another platform, such as the desktop.
False. The view and view model are often tightly coupled and use features of the Windows Phone 7 platform, making these elements difficult to port.
 - c. It helps to make it possible for designers and developers to work in parallel.
True. Designers can work on the XAML files that make up the view independently of the developers working on the code in the view models.
 - d. It may help you to avoid risky changes to existing model classes.
True. Sometimes you may have existing model objects that encapsulate complex domain logic that you don't want to change. In this case, the view model acts as an adaptor, exposing the model to the view.

For more information, see the section, “Benefits of MVVM,” in Chapter 4.

2. Which of the following are good reasons to not use the MVVM pattern for your Windows Phone 7 application?
 - a. You have a very tight deadline to release the application.

True. The MVVM pattern tends to add complexity to the application, especially during the initial development. However, you should carefully evaluate the long-term benefits of using the MVVM pattern.
 - b. Your application is relatively simple with only two screens and no complex logic to implement.

True. Such a simple application may not warrant the overheads of implementing the MVVM pattern.
 - c. Windows Phone 7 controls are not ideally suited to the MVVM pattern.

False. The Windows Phone 7 controls support binding in the same was as standard Silverlight controls. In some cases though, you may have to develop custom behaviors to get specific functionality.
 - d. It's unlikely that your application will be used for more than six months before it is completely replaced.

True. One of the advantages of the MVVM pattern is improved maintainability. For an application with a short shelf life, it may not be worth the overheads to the development process associated with the MVVM pattern.

For more information, see the section, “Benefits of MVVM,” in Chapter 4.

3. Which of the following differentiate **Pivot** and **Panorama** controls?
 - a. The **Pivot** control enables navigation in two dimensions; the **Panorama** control only supports navigation in a single dimension.

False. Both controls use horizontal navigation.
 - b. The **Panorama** control provides a large canvas that extends beyond the confines of the screen. The **Pivot** control allows the user to navigate between different views.

True. These describe core usage scenarios for the two controls.

- c. The **Pivot** control is ideally suited to displaying filtered lists.

True. This is how Tailspin uses the Pivot control.

- d. The **Panorama** control is ideally suited to displaying media items.

False. Although the Panorama control can display media items as the TakeSurveyView page in the Tailspin application illustrates, there's nothing to stop you from displaying media items on a PivotItem on a Pivot control.

For more information, see the section, “User Interface Elements,” in Chapter 4.

4. Which of the following describe the role of the view model locator?

- a. The view model locator configures bindings in the MVVM pattern.

False. Although the view model locator establishes the link between a view and a view model, the bindings are configured in the view's XAML code.

- b. In the Tailspin mobile client, the view model locator is responsible for instantiating view model objects.

False. In the Tailspin application, the dependency injection container instantiates the view model objects and manages their lifetimes.

- c. The view model locator hooks up views to view models.
True. This is the core functionality of the view model locator.

- d. Data template relations offer an alternative approach to a view model locator.

True. This is an alternative way to connect views to view models.

For more information, see the section, “Connecting the View and the View Model,” in Chapter 4.

5. Where does the Back button take you?

- a. To the previous view in the navigation stack.

True. This is what happens within your application.

- b. It depends on what the code in the view model does.

False. The behavior of the hardware Back button is determined by the operating system.

- c. If the current view is the last one in the navigation stack, you leave the application.
True. This returns you to the phone's Start screen.
- d. If your application is on the top of the phone's application stack, it takes you back to your application.
True. You can use the Back button to navigate back to your application from the phone's environment.

For more information, see the section, "Navigation," in Chapter 4.

6. Why should you not use code-behind when you're using the MVVM pattern?
 - a. The view model locator always intercepts the events, so code-behind code never executes.
False. The view model locator does not intercept control events.
 - b. The MVVM pattern enforces a separation of responsibilities between the view and the view model. UI logic belongs in the view model.
True. This is the separation of responsibilities between the view and the view model.
 - c. If you are using the MVVM pattern, other developers will expect to see your code in the view model classes and not in the code-behind.
True. This is the expected place for UI logic.
 - d. Code-behind has a negative effect on view performance.
False. Whether code is in code-behind files or in view model classes has no effect on the application's performance.

For more information, see the section, "Using the Model-View-View Model," in Chapter 4.

CHAPTER 5

USING SERVICES ON THE PHONE

1. Why is it difficult to encrypt data in isolated storage?
 - a. The Windows Phone 7 platform does not include any useful encryption APIs.
False. The WP7 platform includes the following cryptographic algorithms: AES, HMACSHA1, HMACSHA256, Rfc2898DeriveBytes, SHA1, and SHA256.

- b. There is no way to securely encrypt data in isolated storage without requiring the user to enter a password or PIN at least once every time they use the application.
True. The Windows Phone 7 platform does not include a way to securely encrypt passwords without the user entering a password or a PIN. If the user has to enter a password or a PIN frequently, this will have a negative effect on the application's usability.
- c. Any encryption mechanism will increase CPU usage and reduce battery life.
False. Although encrypting data may involve some additional CPU usage, it's unlikely to have a significant impact on battery life.
- d. You cannot control the format of the data in isolated storage.
False. The only requirement is that you can serialize the data that you want to store.

For more information, see the section, “Using Isolated Storage on the Phone,” in Chapter 5.

2. What happens when your application is reactivated?
 - a. You return to the first screen in your application.
False. You are returned to the last screen that was visible before the application was deactivated.
 - b. The operating system makes sure that the screen is displayed as it was when the application was deactivated.
False. It's your application's responsibility to restore whatever state is required to reset the application's appearance and behavior.
 - c. The operating system recreates the navigation stack within your application.
True. If you were several pages deep within the application when it was deactivated, you will still be several pages deep when the application is reactivated.
 - d. The **Launching** event is raised.
False. The Launching event is raised when the application is run from scratch, not when it is reactivated. There is an **Activated event that is raised when the application is reactivated.**

For more information, see the section, “Handling Activation and Deactivation,” in Chapter 5.

3. What data should you save when you handle the deactivation request?

- a. State data required to rebuild the state of the last screen that was active before the application was deactivated.

True. It's the application's responsibility to manage the state data used to redisplay the UI, although the operating system will remember which page in your application was visible when it is deactivated.

- b. State data required to rebuild the state of previous screens that the user had navigated through before the application was deactivated.

True. It's the application's responsibility to manage the state data used to redisplay the UI, including any screens in the application's navigation stack.

- c. Data that is normally persisted to isolated storage by the application at some point.

True. There is no guarantee that an application will be reactivated, so you should save anything important.

- d. The currently active screen.

False. The operating system will record which screen was active for you.

For more information, see the section, “Handling Activation and Deactivation,” in Chapter 5.

4. Why does Tailspin use the Reactive Extensions (Rx) for .NET?

- a. To handle notifications from the Microsoft Push Notification Service

False. You don't need to use the Rx for this task.

- b. To handle UI events

False. You don't need to use the Rx for this task.

- c. To manage asynchronous tasks

True. This is a core use of the Rx.

- d. To make the code that implements the asynchronous and parallel operations more compact and easier to understand

True. This is a benefit of using Rx.

For more information, see the section, “Synchronizing Data between the Phone and the Cloud,” in Chapter 5.

5. What factors should you take into account when you use location services on the phone?
 - a. The level of accuracy your application requires for its geo-location data
True. You can specify the required level of accuracy, and this will affect how the phone obtains its current position: using GPS, or from triangulation.
 - b. Whether the device has a built-in GPS
False. The hardware specification for the Windows Phone 7 device includes a GPS module.
 - c. How quickly you need to obtain the current location
True. Using GPS is often slower, though more accurate, than triangulation.
 - d. Whether the user has consented to your application using the phone's GPS data
True. The guidelines for the phone state that you must obtain the user's consent before using geo-location data from the phone in your application.

For more information, see the section, "Using Location Services on the Phone," in Chapter 5.

6. What special component do you need to handle XNA interop when you're recording audio?
 - a. None.
False. You can't hook into events from XNA objects directly.
 - b. You must state that your application will use XNA interop in the WMAppManifest.xml file.
False. One of the capabilities you can request in the WMAppManifest.xml file is for the microphone but this in itself doesn't enable the XNA interop.
 - c. An XNA Asynchronous Dispatcher class.
True. This class must create a DispatcherTimer object to handle events from XNA.
 - d. An **Observable** instance.
False. The Observable class is part of the Reactive Extensions and is not required to use XNA objects, although they may be useful.

For more information, see the section, "Using XNA Interop to Record Audio," in Chapter 5.

CHAPTER 6

CONNECTING WITH SERVICES

1. What do you need to do to make a website “mobile-friendly?”
 - a. Configure your site to identify requests from mobile browsers and to return pages optimized for mobile web browsers.

True. This is the strategy that Tailspin uses on its website. However, an alternative approach would be to use progressive enhancement and feature detection to detect which features the browser supports, instead of making a hard check on a user agent string and then rendering the HTML.
 - b. Create web pages using the WML mark-up language.

False. Modern mobile web browsers can use XHTML.
 - c. Minimize the page size as far as possible.

True. Depending in the user's contract, they may pay for band-width use, so keeping page sizes small keeps costs down.
 - d. Don't use JavaScript.

False. Modern mobile web browsers typically support JavaScript.

For more information, see the section, “Installing the Mobile Client Application,” in Chapter 6.

2. How does Tailspin pass authentication requests to the web service?
 - a. Tailspin uses basic authentication with the credentials in an authorization header.

True. This is Tailspin's current approach.
 - b. Tailspin uses Windows Live® ID.

False. However, this might be a mechanism for Tailspin to adopt in the future.
 - c. Tailspin uses OAuth.

False. However, this might be a mechanism for Tailspin to adopt in the future.
 - d. Tailspin uses the Windows Identity Framework (WIF).

False. Tailspin uses WIF to process the authentication request in the web service.

For more information, see the section, “Authenticating with the Surveys Service,” in Chapter 6.

3. What notification methods does the Microsoft Push Notification Service (MPNS) support?

- a. Toast notifications.

True. Used for important notifications for immediate viewing, such as breaking news.

- b. Tile notifications.

True. Used for informational notifications such as a temperature change for a weather application.

- c. SMS notifications.

False. This is not a feature of MPNS.

- d. Raw notifications.

You can use raw notifications in addition to tile and toast notifications to send information directly to your application.

For more information, see the section, “Notifying the Mobile Client of New Surveys,” in Chapter 6.

4. Why does the client need to register with MPNS before it can receive notifications?

- a. Because MPNS requires clients to authenticate before it will send notifications.

False. There's no requirement for MPNS clients to authenticate.

- b. Because MPNS can then notify your service that the client is ready to receive notifications.

False. It is not the responsibility of MPNS to notify your service about clients. Clients must also register directly with your service.

- c. Because the client must obtain a unique URI to send to your service.

True. Your client obtains a unique URI from the MPNS that it then forwards to your service. Your service can then pass the URI to the MPNS when it asks the MPNS to notify your client.

- d. Because the free version of MPNS has a limit on the number of clients who can receive notifications from your service.

False. The free version of the MPNS has limits on the number of messages that you can send in a day, but does not limit the number of clients.

For more information, see the section, “Notifying the Mobile Client of New Surveys,” in Chapter 6.

5. How does Tailspin transport data between the client and the web service?

- a. Tailspin uses the Microsoft Sync Framework to handle the data transport.

False. However, Tailspin may consider this in the future.

- b. Tailspin uses the Windows Communication Foundation (WCF) Data Service framework.

False. However, Tailspin may consider this in the future when it fully supports Windows Azure table storage.

- c. Tailspin uses data transfer objects with a WCF REST endpoint.

True. Tailspin currently uses this approach. The data transfer requirements are relatively simple, so this approach was not too difficult to implement.

- d. The mobile client application uploads directly to Windows Azure blob storage.

False. This approach would not be secure or robust enough. The application also uses Windows Azure table storage.

For more information, see the section, “Accessing Data in the Cloud,” in Chapter 6.

6. Why does Tailspin filter data on the server and not on the client?

- a. To minimize the amount of data moved over the network

True. Bandwidth costs are a significant consideration for any mobile application.

- b. To simplify the application

False. In some ways this complicates the application because the client has to send the filter criteria up to the server.

- c. For security reasons

False. Not in Tailspin’s scenario, but in other applications the filtering may determine what data the user is allowed to see rather than what they want to see.

- d. To minimize storage requirements on the phone

True. This is not the primary reason for Tailspin but it does mean that the phone does not download data that is not relevant to the user and cache it while it filters.

For more information, see the section, "Filtering Data," in Chapter 6.

CHAPTER 7

INTERACTING WITH WINDOWS MARKETPLACE

1. Which of the following are valid ways that you can distribute your Windows Phone 7 applications?

- a. Publish it as a full product for sale through Windows Marketplace.
- b. Create an installer and sell it on a CD-ROM.
- c. Create a trial version and offer it free to users through Windows Marketplace.
- d. Plug in each phone to the USB port and copy it directly to the phone.

Answer: Only "a" and "c" are valid. Applications can only be installed through Windows Marketplace. The single exception to this is when developing an application and using a phone that you have registered with the Developer Portal to side-load the application through Microsoft Visual Studio® development system and the Zune® *digital media player* software during development, debugging, and testing.

2. Which of the following services are available through the Windows Phone Developer Portal?

- a. Developer registration service.
- b. Tools for building Windows Phone 7 applications.
- c. Windows Marketplace account management.
- d. Application submission service for your applications.

Answer: All of these services are provided by the Windows Phone Developer Portal.

3. Which of the following are **not** requirements for obtaining certification for an application?

- a. Applications must be free from viruses and malicious software.
- b. Applications must be reliable.
- c. Applications must allow users to select music files on their phone.
- d. Applications must minimize use of resources.

Answer: Only “**a**” and “**b**” are actually published requirements. However, there is a requirement that applications must make efficient use of resources and so “**d**” is, in reality, also a requirement.

4. Which of the following programming techniques are permitted in an application?
 - a. The application can terminate if the phone does not have extended memory installed.
 - b. The code can use PInvoke or COM interop to execute native code.
 - c. The application can use code that is not type safe if this is absolutely necessary.
 - d. The application can use reflection to discover and invoke types that require security permissions.

Answer: The programming techniques “**a**”, “**b**”, and “**c**” are not permitted. Although the use of reflection is not banned, technique “**d**” is likely to cause the application to fail certification. It may not have the required security permission demands in the auto-generated manifest because the automated submission process does not discover security permissions required using this technique.

5. For which of the following functions must the application prompt the user to obtain consent before using that feature?
 - a. The location service.
 - b. Notifications.
 - c. Person-to-person communication, such as chat features.
 - d. Downloading additional content or data.

Answer: The application must obtain consent from the user for “**a**” and “**b**”. Obtaining consent is not sufficient for “**c**”; the application must also establish that the user is at least 13 years old. The only time that consent must be obtained for “**d**” is if the download is large (typically greater than 50 MB) or it uses services or communication protocols that may incur excessive communication costs.

6. Which of the following tasks are executed during the application submission process?
- Unpacking and validating the content of the XAP file.
 - Recreating the permissions demands in the manifest file.
 - Removing any content that does not conform to content-restriction policies.
 - Adding a Digital Rights Management header.

Answer: Only “a”, “b”, and “d” are automatically executed during the submission process. It is up to the developer to ensure that the content of their application meets the published content policy guidelines.

Index

A

3-D graphics, 215
accelerometer, 223-226
accounts, 193-195
acknowledgments, xxv-xxvii
Activated event, 25-26
activation and deactivation, 106-110
AddMediaAnswer method, 166
AddSurveyAnswers method, 167
Ad Exchange, 195-196
advertisements, 195-196
appendices *see* device capabilities; Microsoft Sync Framework; Prism library; Silverlight and XNA; tools, frameworks, and processes
App Hub website, 193-194
ApplicationBarButtonCommand class, 86
ApplicationBarIconButton control, 84
application design *see* design
ApplicationFrameNavigationService class, 64, 89-90
applications
 computer vs. mobile phone, 20-21
 debugging, 209-211
 deploying, 205-206
 developing, 206-209
 foreground, 190
 mobile client, 137
 mobile client construction, 56-57
 packaging, 191-192
 settings, 101-103

Tailspin scenario, 49-51
Tailspin Surveys mobile client
 application, 57-59
 testing, 74-75
 unit testing, 211-212
approval process, 193-195
AppSettingsViewModel class, 75
 mobile client construction, 64, 77
 services, 112-113
AppSettingsView page, 63, 75, 77, 111-112
AppSettingsView.xaml file, 112
asynchronous location service
 operation, 233-235
asynchronous operations, 111
audience, xviii
audio data, 127
authentication, 139-143
Azure Access Control Services (ACS), 142

B

Back button, 54, 189
backgroundImage parameter, 161
basic design considerations, 17-18
behaviors, 69
benefits, 71
bibliography, xxiii
BindChannelAndUpdateDevice UriInService method, 151-152
Bing Maps Silverlight Control, 236-237
BuildPivotDimensions method, 79-80
business model, 49

- C**
- CallUpdateReceiveNotifications method, 152
 - CameraCaptureCommand command, 128-129
 - CameraCaptureTask chooser
 - device capabilities, 226-227
 - overview, 221
 - services, 127-129
 - cameras, 226-227
 - Cancel button, 90-91
 - capabilities *see* device capabilities
 - choosers, 128-130
 - CameraCaptureTask chooser, 127-129, 226-227
 - described, 221
 - EmailChooser_Completed callback, 230
 - list, 228-229
 - cloud, 162-171
 - creating a WCF REST service in, 164-167
 - synchronizing between phone and, 114-122
 - code restrictions, 188
 - commands, 85-88
 - community support, 14-15
 - connectivity *see* services connectivity
 - connectivity considerations, 31
 - contacts and messaging, 228-230
 - ContainerLocator class, 59-60
 - ContextMenu control, 67, 203
 - contributors and reviewers, xxvi-xxvii
 - controls, 54
 - control templates, 66-67
 - credentials, 63
 - CustomServiceHostFactory class, 143-144
- D**
- data
 - binding on the settings screen, 75-77
 - data-driven applications, 4-5
 - displaying, 75-85
 - filtering, 172-181
 - format and synchronization considerations, 35
 - formats and accessing remote services, 34-37
 - image and audio data, 127
 - surveys, 103-106
 - synchronizing between phone and cloud, 114-122
 - DataContext attribute, 76-77
 - DatePicker control, 203
 - Deactivated event, 25-26
 - DefaultTask element, 62
 - definitions, 9-14
 - DelegateCommand class, 86-87
 - dependency injection, 59-60
 - design, 17-44
 - application deactivation and tombstoning, 25-28
 - availability of components and frameworks, 42-43
 - basic design considerations, 17-18
 - connectivity considerations, 31
 - data format and synchronization considerations, 35
 - data formats and accessing remote services, 34-37
 - design and implementation, 19
 - design considerations, 23-43
 - device resource usage, 38
 - good programming practices, 38-39
 - GPU usage, 41-42
 - life cycle and events, 27
 - memory utilization, 39-41
 - Microsoft data and synchronization technologies for Windows Phone 7, 36
 - mobile phone applications, 20-21
 - remote services, 20
 - resource management, 19-20
 - resource management and performance, 38-42
 - security considerations, 32-33

Silverlight and Windows Phone 7, 22-23
storage considerations, 29-31
suitable application types, 21-22
types of applications, 19
UI design and style guidelines, 24
user input considerations, 28-29
development, 5-7
 environment setup, 199-203
 process goals, 56
 resources, 6-9
 Web and service developers, 8-9
 Web application development, 209
device capabilities, 221-255
 accelerometer, 223-226
 asynchronous location service operation, 233-235
 Bing Maps Silverlight Control, 236-237
 camera, 226-227
 CameraCaptureTask chooser, 226-227
 contacts and messaging, 228-230
 DeviceExtendedProperties class, 230-232
 device information, 230-232
 EmailChooser_Completed callback, 230
 GeoCoordinateWatcher class, 235-236
 GeoCoordinateWatcher instance, 233-234
 GestureListener control, 246-249
 GetValue method, 231-232
 GPS, 232-233
 location and mapping, 232-233
 manipulation events, 245-246
 maps and location information, 236-237
media, 237-239
Microsoft.Phone namespace, 227
Observable class, 255
photo selection, 239-240
PositionChanged event
 handler, 234
reactive extensions, 254-255
scenarios, 222-223
SearchTask class, 240
sound playback, 243
sound recording, 240-242
StatusChanged event
 handler, 234
synchronous location service operation, 235-236
System.IO.IsolatedStorage namespace, 227
System.Windows.Media.Imaging, 227
touch and gestures, 244
TouchPanel class, 250-252
TryGetValue method, 231-232
vibration alerts, 252
WebBrowserTask class, 253
Windows Marketplace, 253-254
XNA gesture detection, 250-252
DeviceExtendedProperties class, 30, 55, 230-232
devices
 identifying for notification, 174-175
 information, 230-232
 listing for notifications, 179-180
 resource usage, 38
 see also device capabilities
diagnostics, 132
Digital Rights Management (DRM), 193
dispatcher service, 218-220
DispatcherTimer Class, 218-219

- E**
- EmailChooser_Completed
 - callback, 230
 - enumerable and observable sequences, 111-112
 - environment setup, 199-203
 - errors, 209-211
 - and diagnostics logging, 132-133
 - notifications, 92
 - example application, xxi
 - Exchangeable Image File (EXIF), 226
 - Expression Blend for Windows Phone, 202-203
- F**
- Fabrikam and Adatum, 49
 - features available on Windows Phone 7 devices, 3
 - feedback and support, 14-15
 - FilteringService class, 179-180
 - filters
 - filtering data, 172-181
 - storing filter data, 177-178
 - tenant filter table, 173
 - FilterSettingsViewModel class, 89
 - folders, 60-61
 - foreach construct, 111
 - foreground applications, 190
 - forward, xiii-xv
 - FrameworkDispatcher class, 218-219
 - FrameworkElement control, 84
 - Funq dependency injection container, 59-60, 212
 - further information, xxiii, 14-15
 - mobile client construction, 96
 - services, 135
 - services connectivity, 183
- G**
- game execution model, 217
 - GeoCoordinateWatcher class, 124-126
 - device capabilities, 235-237
- H**
- HeaderTemplate data template, 83
 - how to use this guide, xix-xxi
 - newsgroup support, 14-15
 - HttpClient class, 168
 - HttpNotificationChannel object, 149
 - HttpWebRequestExtensions class, 168
- I**
- ICommand interface, 85
 - image and audio data, 127-132
 - informational notifications, 91-92
 - InitializeComponent method, 69
 - INotifyPropertyChanged interface, 75
 - introduction, 1-15
 - IRegistrationService interface, 149
 - IsBeingActivated method, 109-110
 - ISettingsStore interface, 101
 - isolated storage, 98-99
 - ISurveysServiceClient interface, 167-168
 - ISV, 48
 - IT professional role (Poe), xxiii

J

JavaScript Object Notation (JSON), 163
JSON serialization vs. XML, 55

L

launchers
described, 221
list, 228-229
life cycle and events, 27
ListBox control, 66
localization, 190-191
location and mapping, 232-233
Location Service, 123-126, 232-233, 236
location services, 54, 135
described, 11

M

manipulation events, 245-246
maps
Bing Maps Silverlight Control, 236-237
location and mapping, 232-233
Maps property, 84
media, 237-239
MediaElement control, 238
memory
storage limits, 30
utilization, 39-41
Metro design style, 24
Microphone class, 242
Microsoft Ad Exchange, 195-196
Microsoft data and synchronization technologies, 36-37
Microsoft.Phone namespace, 227
Microsoft.Practices.Prism.
Commands namespace, 260
Microsoft.Practices.Prism.Events
namespace, 260-261
Microsoft.Practices.Prism.
Interactivity.InteractionRe-
quest namespace, 263-264
Microsoft.Practices.Prism.
Interactivity namespace, 262
Microsoft.Practices.Prism
namespace, 259

Microsoft.Practices.Prism.

ViewModel namespace, 261
Microsoft Push Notification

Service (MPNS), 55, 111, 146-150, 154-159
restrictions, 189

Microsoft Sync Framework, 265-272
overview, 267-269
providers, 269-270
Sync Framework, 271-272
synchronization for Win-
dows Azure, 271-272
SyncSvcUtil utility, 271-272
Tailspin scenario, 116
using, 270-271

MobileCapableWebFormViewEn-
gine class, 138-139

mobile client application, 137
mobile client construction, 53-96
application components, 56-57

ApplicationFrameNavi-
gationService class, 89-90

AppSettingsViewModel
class, 64, 77

BuildPivotDimensions
method, 79-80

Cancel button, 90-91

ContextMenu control, 67

control templates, 66-67

DataContext attribute, 76-77

DelegateCommand class,
86-87

dependency injection, 59-60
development process goals, 56

FilterSettingsViewModel
class, 89

ListBox control, 66

more information, 96

MVVM pattern, 59, 67-94

navigation, 61-65

non-functional goals, 55

Panorama control, 66, 81-83

Pivot control, 66, 77-78

PivotItem control, 78-79

SaveCommand command,
88-89

SettingAreNotConfigured
 property, 63
styling and control
 templates, 66-67
SurveyDataTemplate data
 template, 64
SurveyListViewModel class,
 79, 87-88, 93-94
SurveyListViewModelFixture
 class, 74-75
SurveyListView page, 62, 86
SurveyListView.xaml file,
 92-93
Surveys client application, 62
TailSpin.PhoneClient project,
 60-61
TailSpin.PhoneClient.
 ViewModels ViewModel
 Locator class, 71-72
Tailspin Surveys mobile client
 application, 57-59
TakeSurveyViewModel class,
 84
 templates, 66-67
UI description, 65
UI design, 61-67
UI elements, 66-67
UpdateTextBindingOn
 PropertyChanged
 behavior, 84-85
usability goals, 54
ViewModelLocator class,
 72-74
 see also MVVM pattern
mobile phone applications, 20-21
model classes, 97-98
Model-View-Controller (MVC),
 138
Moq framework, 212
more information, xxiii, 14-15
 mobile client construction,
 96
 services, 135
 services connectivity, 183
MPNS *see* Microsoft Push
 Notification Service (MPNS)
MVVM pattern, 67-94
 accessing services, 94

ApplicationFrameNavigation
 Service class, 89-90
application testing, 74-75
benefits, 56, 71
commands, 85-88
data binding on the settings
 screen, 75-77
defined, 11
displaying data, 75-85
error notifications, 92
handling navigation requests,
 88-91
informational notifications,
 91-92
informational warnings,
 91-92
mobile client construction,
 59, 67-94
ObservableCollection class,
 80
overview, 68-70
Panorama control, 81-84
Pivot control, 77-81
RaisePropertyChanged
 methods, 77
SaveCommand command,
 88-89
settings screen, 75-77
SurveyListViewModelFixture
 class, 74
UI notifications, 91-94
view model connection to
 the view, 71-73
ViewModelLocator class,
 71-73
warning notifications, 91-92

N

navigation, 61-65
navigation requests, 88-91
NavigationService class, 64
NewSurveyNotification, 155
NewSurveyNotificationCommand
 class, 155-156, 179
Ninject dependency injection
 container, 212
non-functional goals, 55

- notifications
 - errors, 92
 - identifying for, 174-175
 - payloads, 160-161
 - raw notification, 147
 - registering for, 148-162
 - sending, 155-160
 - tile notification, 147
 - toast notification, 147
 - UI notifications, 91-94
 - warning notifications, 91-92
- O**
 - Observable class, 255
 - ObservableCollection class, 80
 - ObserveOnDispatcher method, 87-88
 - OnNotified method, 158-159
 - Open Authentication (OAuth) 2.0 protocol, 141-143
 - Open Data Protocol (OData), 34-37
 - defined, 12
 - OpenQuestionView.xaml, 84-85
 - overview, 221
 - Microsoft Sync Framework, 267-269
 - MVVM pattern, 68-70
- P**
 - packaging, 191-192
 - Panorama control
 - defined, 12
 - mobile client construction, 66, 81-83
 - MVVM pattern, 81-84
 - services, 110
 - Password property, 101-103
 - performance, 38-42
 - Person class, 69
 - PhoneApplicationService.
 - ApplicationIdleDetectionMode property, 189
 - phone specialist role (Christine), xxii
 - PhotoChooserTask class, 239
 - photo selection, 239-240
- Pivot control, 77-81
 - defined, 12
 - mobile client construction, 66, 77-78
- PivotItem control, 78-79
- platform standardization, 2-4
- PositionChanged event handler, 234
- PostJson method, 153-154
- preface, xvii-xxiii
- prerequisites, xxi-xxii
- Prism library, 213, 257-264
 - contents, 258-259
 - Microsoft.Practices.Prism.
 - Commands namespace, 260
 - Microsoft.Practices.Prism.
 - Events namespace, 260-261
 - Microsoft.Practices.Prism.
 - Interactivity.Interaction Request namespace, 263-264
 - Microsoft.Practices.Prism.
 - Interactivity namespace, 262
 - Microsoft.Practices.Prism namespace, 259
 - Microsoft.Practices.Prism.
 - ViewModel namespace, 261
 - programming practices, 38-39
 - project folders, 60-61
 - providers, 269-270
 - push notifications, 55, 111, 146-150, 154-159
 - PushTileNotification method, 156-157
 - R**
 - RaisePropertyChanged method, 77
 - raw notification, 147
 - reactivation, 110
 - Reactive Extensions (Rx), 111-112, 119, 254-255
 - Tailspin scenario, 116-117

RegisterRoutes method, 164-165
 RegistrationService class, 178
 registration web service, 154-155
 remote services, 20
 Representational State Transfer (REST), 162
 defined, 12-13
 RequestTo method, 153
 resources
 development, 6-9
 management and design, 19-20
 management and performance, 38-42
 Web and service developers, 8-9
 XNA, 8
 reviewers and contributors, xxvi-xxvii
 roles, xxii-xxiii

S

SaveAndUpdateMediaAnswers method, 168-171
 SaveCommand command, 88-89
 scenarios, xxii-xxiii
 device capabilities, 222-223
 Microsoft Sync Framework, 266
 phone form factor, 22
 see also Tailspin scenario
 SearchTask class, 240
 security, 99-100
 authentication, 139-143
 considerations, 32-33
 SendMessage method, 157-159
 senior software developer role (Markus), xxiii
 sequences, 111-112
 serializable model classes, 100
 services, 97-135
 accessing, 94
 activation and deactivation handling, 106-110
 application settings, 101-103
 AppSettingsViewModel class, 112-113
 asynchronous operations, 111
 audio data, 127
 CameraCaptureCommand command, 128-129
 CameraCaptureTask chooser, 127-129
 data synchronization between phone and cloud, 114-122
 error and diagnostics logging, 132-133
 GeoCoordinateWatcher class, 124-126
 getNewSurveys task, 120-122
 image and audio data, 127-132
 IsBeingActivated method, 109-110
 ISettingsStore interface, 101
 isolated storage, 98
 location services, 123-126
 model classes, 97-98
 more information, 135
 Panorama control, 110
 Password property, 101-103
 reactivation, 110
 Reactive Extensions (Rx), 111-112
 security, 99-100
 storage format, 100
 SurveyAnswer class, 103
 survey data, 103-106
 SurveyListViewModel class, 109
 SurveysSynchronization Service class, 118-120
 SurveyStore class, 103-105
 synchronization methods, 119
 TakeSurveyView page, 111
 TaskSummaryResult object, 113-114
 ToggleSwitch control, 112
 Tombstoning class, 107-108
 TryToGetCurrentLocation method, 126
 ViewModel class, 108-109
 WebException exception, 114

XNA to record audio, 130-132
see also services connectivity
services connectivity, 137-183
AddMediaAnswer method, 166
AddSurveyAnswers method, 167
authenticating with the surveys service, 139-143
backgroundImage parameter, 161
BindChannelAndUpdateDeviceUriInService method, 151-152
CallUpdateReceive
 Notifications method, 152
cloud data access, 162-171
CustomServiceHostFactory class, 143-144
data consuming, 163-164
data filtering, 172-181
data filtering for storage, 177-178
data in the Windows Phone 7 client application, 167-171
device listing for notifications, 179-180
device notification, 174-175
FilteringService class, 179-180
GetNewSurveys method, 168
GetSurveys method, 165-166
IRegistrationService interface, 149
ISurveysServiceClient interface, 167-168
Microsoft Push Notifications Service (MPNS), 146-148, 159
MobileCapableWebForm-ViewEngine class, 138-139
mobile client application installation, 137
more information, 183
NewSurveyNotification class, 155
NewSurveyNotification
 Command class, 155-156, 179
notification of devices, 174-175
notification payloads, 160-161
notification registration, 148-162
notification sending, 155-160
OnNotified method, 158-159
Open
 Authentication(OAuth)
 2.0 protocol, 141-143
PostJson method, 153-154
PushTileNotification method, 156-157
raw notification, 147
RegisterRoutes method, 164-165
RegistrationService class, 178
registration web service, 154-155
Representational State Transfer (REST), 162
RequestTo method, 153
SaveAndUpdateMediaAnswers method, 168-171
SendMessage method, 157-159
Simple Web Token (SWT), 141-143
SimulatedWebService
 AuthorizationManager class, 144-146
SSL, 164
surveys, 175-176
surveys to synchronize with the mobile client, 180-181
TenantFilter class, 179-181
TenantFilterStore class, 177
tenant filter table, 173
tile notification, 147
toast notification, 147
ToastNotificationPayload
 Builder class, 161
UpdateReceiveNotifications method, 149-151

UserDeviceStore class, 177-178
 user device table, 174
 WCF REST service in the cloud, 164-167
 where clause, 173
 Windows Communication Foundation (WCF), 162-163
 SettingAreNotConfigured property, 63
 settings screen, 75-77
 side loading, 204
 Silverlight, 5
 controls, 203
 resources, 7-8
 and Windows Phone 7, 22-23
 Silverlight and XNA, 215-220
 audio, 127
 excluded classes and assemblies, 220
 interop from Silverlight to XNA, 218
 summary of differences, 216
 XNAAsyncDispatcher class, 130, 219-220
 XNA dispatcher service, 218-220
 XNA game execution model, 217
 Simple Web Token (SWT), 141-143
 software architect role (Jana), xxii
 sound playback, 243
 sound recording, 240-242
 SSL, 164
 standard controls, 54
 standardized platform, 2-4
 StartSync method, 87, 118-119
 StatusChanged event handler, 234
 storage
 considerations, 29-31
 filter data, 177-178
 format, 100
 isolated storage, 98
 memory, 30
 strategy, 45-46
 styling
 and control templates, 66-67
 Metro design style, 24
 themes, 54
 Subscribe method, 113-114
 subscribers, 49
 support, 3, 14-15
 SurveyAnswer class, 103
 SurveyDataTemplate data template, 64
 SurveyListViewModel class, 79, 87-88, 93-94, 109
 SurveyListViewModelFixture class
 mobile client construction, 74-75
 MVVM pattern, 74
 SurveyListView page, 62, 65, 86
 SurveyListView.xaml file, 92-93
 surveyors, 49
 surveys
 data, 103-106
 services connectivity, 175-176
 to synchronize with the mobile client, 180-181
 Surveys application
 application architecture, 47-48
 authentication, 139-143
 client application, 62
 in Tailspin scenario, 45-51
 SurveysSynchronizationService class, 118-120, 171
 SurveyStore class, 103-105
 SyncCompleted method, 87-88
 Sync Framework, 271-272
 synchronization
 methods, 119
 between phone and cloud, 114-122
 for Windows Azure, 271-272
 see also Microsoft Sync Framework
 synchronous location service
 operation, 235-236
 SyncSvclUtil utility, 271-272
 System.IO.IIsolatedStorage
 namespace, 227
 System.Windows.Media.Imaging, 227

T

TailSpin.PhoneClient project, 60-61
 Tailspin scenario, 45-51
 application components, 49-51
 business model, 49
 Fabrikam and Adatum, 49
 goals and concerns, 46-47
 ISV, 48
 strategy, 45-46
 subscribers, 49
 surveyors, 49
 Surveys application
 architecture, 47-48
 Tailspin Surveys mobile client application, 57-59
 TakeSurveyViewModel class, 84
 TakeSurveyView page, 59, 111
 task launchers
 described, 221
 list, 228-229
 TaskSummaryResult object, 113-114
 team, xxvi-xxvii
 technical support, 14-15
 TenantFilterStore class, 177
 tenant filter table, 173
 terminology, 9-14
 test applications, 208
 ThemedResourceLocator class, 54, 66
 themes, 54
 tile notifications, 147
 TimePicker control, 203
 ToastNotificationPayloadBuilder class, 161
 toast notifications, 147
 ToggleSwitch control, 112, 203
 Tombstoning
 defined, 13
 described, 25
 Tombstoning class, 107-108
 tools, frameworks, and processes, 199-213
 additional tools and frameworks, 212-213
 application debugging, 209-211

application development, 206-207
 application development to the device, 205-206
 development environment
 setup, 199-201
 Expression Blend for Windows Phone, 202-203
 hardware devices during development, 204
 physical device connection, 204
 registering and unlocking devices, 204
 Silverlight controls, 203
 trial applications development, 208
 unit testing applications, 211-213
 unit testing automation, 212
 Web application development, 209
 Windows Phone Connect Tool (WPConnect.exe), 205-206
 touch and gestures, 244
 touch input, 28
 TouchPanel class, 250-252
 trial applications, 208
 TryGetValue method, 231-232
 TryToGetCurrentLocation method, 126

U

UI
 description, 65
 design, 61-67
 design and style guidelines, 24
 elements, 66-67
 notifications, 91-94
 unit testing, 211-212
 UpdateReceiveNotifications method, 149-151
 UpdateTextBindingOnPropertyChanged, 84-85
 usability goals, 54
 UserDeviceStore class, 177-178

- user device table, 174
- user input considerations, 28-29
- V**
 - validation process, 193-195
 - vibration alerts, 252
 - view model, 71-73
 - ViewModel class, 77, 108-109
 - ViewModelLocator class, 59, 72-74
 - MVVM pattern, 71-73
- W**
 - warning notifications, 91-92
 - Web
 - application development, 209
 - and service developers, 8-9
 - WebAii testing framework, 213
 - WebBrowserTask class, 253
 - WebException exception, 114
 - WebFormViewEngine class, 138
 - websites for mobile devices, 139
 - where clause, 173
 - Windows Communication Foundation (WCF), 162-163
 - Windows Identity Foundation (WIF), 141
 - Windows Marketplace, 185-196
 - accessing, 196
 - advertisement display, 195-196
 - application certification requirements, 187-188
 - application code restrictions, 188
 - ApplicationIdleDetectionMode property, 190
 - Application Media and Visual Content, 190-191
 - defined, 14
 - development and publishing life cycle, 185-187
 - device capabilities, 253-254
 - packaging the application, 191-192
 - repackaging process, 191-192
- run-time behavior, performance, and metrics, 189
- user opt-in and privacy, 189-190
- validation process, 191-192
- validation process summary, 193-195
- XAP files, 191-192
- Windows Phone Capability Detection Tool (Capability-Detection.exe), 193
- Windows Phone Connect Tool (WPConnect.exe), 205-206
- Windows Phone Developer Tools, 201
- Windows Presentation Foundation (WPF), 209
- WMAAppManifest.xml file, 192-193
- WrapPanel control, 203
- X**
 - XAP files, 191-193
 - opening, 205
 - XML vs. JSON serialization, 55
 - XNA, 5-6
 - dispatcher service, 218-220
 - game execution model, 217
 - gesture detection, 250-252
 - to record audio, 130-132
 - resources, 8
 - see also* Silverlight and XNA
 - XNAAsyncDispatcher class, 130, 219-220
- Z**
 - Zune website, 204

More Resources for Developers

Microsoft Press® books

VISUAL STUDIO

Inside the Microsoft® Build Engine: Using MSBuild and Team Foundation Build
Sayed Ibrahim Hashimi, William Bartholomew
978-07356-2628-7

Microsoft .NET: Architecting Applications for the Enterprise
Dino Esposito, Andrea Saltarello
978-07356-2609-6

Microsoft .NET and SAP
Juergen Daiberl, et al.
978-07356-2568-6

Microsoft Visual Basic® 2008 Express Edition: Build a Program Now!
Patrice Pelland
978-07356-2541-9

Microsoft Visual Basic 2008 Step by Step
Michael Halvorson
978-07356-2537-2

Microsoft Visual C#® 2008 Step by Step
John Sharp
978-07356-2430-6

Programming Microsoft Visual C#® 2008: The Language
Donis Marshall
978-07356-2540-2

Microsoft Visual Studio® Tips
Sara Ford
978-07356-2640-9

Windows® via C/C++, Fifth Edition
Jeffrey Richter, Christophe Nasarre
978-07356-2424-5



Microsoft XNA® Game Studio 3.0: Learn Programming Now!
Rob Miles
978-07356-2658-4

WEB DEVELOPMENT
Developing Service-Oriented AJAX Applications on the Microsoft Platform
Daniel Larson
978-07356-2591-4

Introducing Microsoft Silverlight™ 3
Laurence Moroney
978-07356-2573-0

JavaScript Step by Step
Steve Suehring
978-07356-2449-8

Microsoft ASP.NET and AJAX: Architecting Web Applications
Dino Esposito
978-07356-2621-8

Microsoft ASP.NET 3.5 Step by Step
George Shepherd
978-07356-2426-9

Programming Microsoft ASP.NET 3.5
Dino Esposito
978-07356-2527-3

Microsoft Visual Web Developer™ 2008 Express Edition Step by Step
Eric Griffin
978-07356-2606-5

.NET FRAMEWORK
CLR via C#, Second Edition
Jeffrey Richter
978-07356-2163-3

3D Programming for Windows
Charles Petzold
978-07356-2394-1

Official Microsoft E-Reference Libraries

Access Microsoft E-Reference Libraries from any Internet connection—and create your own online “bookshelf” of problem-solving resources from Microsoft Press.

For details and subscription information, go to:
microsoft.com/learning/books/ereference/default.mspx

DATA ACCESS/ DATABASE

Microsoft SQL Server® 2008 Internals
Kalen Delaney, et al.
978-07356-2624-9

Inside Microsoft SQL Server 2008: T-SQL Querying
Itzik Ben-Gan, et al.
978-07356-2603-4

Programming Microsoft SQL Server 2008
Leonard Lobel, Andrew J. Brust, Stephen Forte
978-07356-2599-0

Smart Business Intelligence Solutions with Microsoft SQL Server 2008
Lynn Langit, et al.
978-07356-2580-8

OTHER TOPICS
Agile Portfolio Management
Jochen Krebs
978-07356-2567-9

Agile Project Management with Scrum
Ken Schwaber
978-07356-1993-7

How We Test Software at Microsoft
Alan Page, Ken Johnston, Bj Rollison
978-07356-2425-2

Practical Project Initiation
Karl E. Wiegers
978-07356-2521-1

Simple Architectures for Complex Enterprises
Roger Sessions
978-07356-2578-5

Software Estimation: Demystifying the Black Art
Steve McConnell
978-0-7356-0535-0

Microsoft®
Press