

# Windows Phone 8 Development Internals



 Professional

Andrew Whitechapel  
Sean McKenna

# Windows Phone 8 Development Internals



## Build and optimize Windows Phone 8 apps for performance and security

Drill into Windows Phone 8 design and architecture, and learn best practices for building phone apps for consumers and the enterprise. Written by two senior members of the core Windows Phone Developer Platform team, this hands-on book gets you up to speed on the Windows 8 core features and application model, and shows you how to build apps with managed code in C# and native code in C++. You'll also learn how to incorporate Windows Phone 8 features such as speech, the Wallet, and in-app purchase.

### Discover how to:

- Create UIs with unique layouts, controls, and gesture support
- Manage databinding with the Model View ViewModel pattern
- Build apps that target Windows Phone 8 and Windows Phone 7
- Use built-in sensors, including the accelerometer and camera
- Consume web services and connect to social media apps
- Share code across Windows Phone 8 and Windows 8 apps
- Build and deploy company hub apps for the enterprise
- Start developing games using Direct3D
- Test your app and submit it to the Windows Phone Store

### Download C# and C++ code samples at:

<http://aka.ms/WinPhone8DevInternals/files>

[microsoft.com/mspress](http://microsoft.com/mspress)

ISBN: 978-0-7356-7623-7



**U.S.A. \$54.99**

Canada \$57.99

[Recommended]

Programming/Windows Phone 8

## About the Authors

**Andrew Whitechapel** is a senior program manager for the Windows Phone Developer Platform team, responsible for internal components within the platform. He has written several books, including *Windows Phone 7 Development Internals*.

**Sean McKenna** is a senior program manager on the Windows Phone Developer Platform team. He has been responsible for several key features, including local database support and app-to-app communication.



# Windows Phone 8 Development Internals

Andrew Whitechapel  
Sean McKenna

Copyright © 2013 by Andrew Whitechapel and Sean McKenna

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-7623-7

1 2 3 4 5 6 7 8 9 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at [msspinput@microsoft.com](mailto:msspinput@microsoft.com). Please tell us what you think of this book at <http://www.microsoft.com/learning/booksurvey>.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

**Acquisitions and Development Editor:** Russell Jones

**Production Editor:** Rachel Steely

**Editorial Production:** Dianne Russell, Octal Publishing, Inc.

**Technical Reviewer:** Peter Torr

**Copyeditor:** Bob Russell, Octal Publishing, Inc.

**Indexer:** WordCo Indexing Services, Inc.

**Cover Design:** Twist Creative • Seattle

**Cover Composition:** Karen Montgomery

**Illustrator:** Rebecca Demarest

*We would like to dedicate this book to Narins Bergstrom  
and Urmila Nadkarni, with thanks for their endless patience and  
support.*

—ANDREW WHITECHAPEL AND SEAN MCKENNA



# Contents at a Glance

*Foreword* xxiii

*Introduction* xxv

## **PART I**      **CORE FEATURES**

---

CHAPTER 1	Vision and architecture	3
CHAPTER 2	App model and navigation	33
CHAPTER 3	UI visuals and touch	77
CHAPTER 4	Data binding and MVVM	139
CHAPTER 5	Phone and media services	187
CHAPTER 6	Sensors	233
CHAPTER 7	Web connectivity	273
CHAPTER 8	Web services and the cloud	315
CHAPTER 9	Background agents	349
CHAPTER 10	Local storage and databases	395

## **PART II**      **WINDOWS PHONE 7 TO WINDOWS PHONE 8**

---

CHAPTER 11	App publication	439
CHAPTER 12	Profiling and diagnostics	467
CHAPTER 13	Porting to Windows Phone 8 and multitargeting	509
CHAPTER 14	Tiles and notifications	539
CHAPTER 15	Contacts and calendar	587
CHAPTER 16	Camera and photos	621
CHAPTER 17	Networking and proximity	667
CHAPTER 18	Location and maps	707

## **PART III**      **NEW WINDOWS PHONE 8 FEATURES**

---

CHAPTER 19	Speech	753
CHAPTER 20	The Wallet	785
CHAPTER 21	Monetizing your app	811
CHAPTER 22	Enterprise apps	845

**PART IV      NATIVE DEVELOPMENT AND WINDOWS PHONE 8  
                         CONVERGENCE**

---

CHAPTER 23	Native development	867
CHAPTER 24	Windows 8 convergence	901
CHAPTER 25	Games and Direct3D	933
	<i>Index</i>	965

# Contents

<i>Foreword</i> .....	<i>xxiii</i>
<i>Introduction</i> .....	<i>xxv</i>

## **PART I**      **CORE FEATURES**

---

<b>Chapter 1</b>	<b>Vision and architecture</b>	<b>3</b>
	A different kind of phone .....	3
	The user interface .....	4
	The role of apps .....	6
	Windows phone architecture .....	8
	Platform stack .....	8
	App types .....	9
	Background processing .....	11
	Security model .....	13
	Windows and Windows Phone: together at last .....	16
	Building and delivering apps .....	18
	Developer tools .....	18
	App delivery .....	20
	Getting started with “Hello World” .....	21
	Creating a project .....	22
	Understanding the project structure .....	24
	Greeting the world from Windows Phone .....	27
	Deploying to a Windows Phone device .....	29
	The Windows Phone Toolkit .....	29
	Summary .....	31

---

### **What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[microsoft.com/learning/booksurvey](https://microsoft.com/learning/booksurvey)

<b>Chapter 2</b>	<b>App model and navigation</b>	<b>33</b>
	The app lifecycle . . . . .	34
	Normal termination . . . . .	37
	App deactivated—fast app resume . . . . .	37
	App deactivated—the tombstone case . . . . .	38
	Setting a resume policy . . . . .	43
	<i>Obscured</i> and <i>Unobscured</i> . . . . .	45
	The page model . . . . .	47
	Page creation order . . . . .	49
	Navigation and state . . . . .	52
	App state . . . . .	53
	Page state . . . . .	56
	Cancelling navigation . . . . .	59
	Backstack management . . . . .	60
	Navigation options . . . . .	63
	Using <i>NavigateUri</i> . . . . .	64
	Pages in separate assemblies . . . . .	64
	<i>Fragment</i> and <i>QueryString</i> . . . . .	65
	The <i>NavigationMode</i> and <i>IsNavigationInitiator</i> properties . . . . .	68
	Re-routing navigation and URI mappers . . . . .	70
	File type and URI associations . . . . .	72
	Starting an app based on a file or URI . . . . .	72
	Acting as a file type or URI handler . . . . .	73
	Summary . . . . .	76
<b>Chapter 3</b>	<b>UI visuals and touch</b>	<b>77</b>
	Phone UI elements . . . . .	77
	Standard UI elements . . . . .	77
	The visual tree . . . . .	81
	Screen layout . . . . .	84
	Working with <i>UserControls</i> vs. custom controls . . . . .	89
	Re-templating controls . . . . .	92

Resources . . . . .	94
Data resources . . . . .	94
XAML resources and resource dictionaries . . . . .	95
Implicit styles . . . . .	99
Dependency and attached properties . . . . .	101
Dependency properties . . . . .	101
Attached properties . . . . .	104
The app bar and notification area . . . . .	108
Transient panels . . . . .	113
Routed events . . . . .	121
Logical touch gestures . . . . .	124
Manipulation events . . . . .	126
Mouse events . . . . .	132
<i>FrameReported</i> events . . . . .	133
Keyboard input . . . . .	135
Summary . . . . .	137

## **Chapter 4 Data binding and MVVM 139**

Simple data binding and <i>INotifyPropertyChanged</i> . . . . .	139
Data-binding collections . . . . .	147
Dynamic data-bound collections . . . . .	150
Command binding . . . . .	153
Template resources . . . . .	157
Sorting and grouping bound collections . . . . .	158
Type/value converters . . . . .	161
Element binding . . . . .	164
Data validation . . . . .	165
Separating concerns . . . . .	171
The Model-View-ViewModel pattern . . . . .	174
The Visual Studio databound application project . . . . .	175
MVVM in Pivot apps . . . . .	180

Row filtering in Pivot apps . . . . .	182
Improving the Visual Studio Databound Application template . . . . .	184
Summary . . . . .	186
<b>Chapter 5 Phone and media services</b>	<b>187</b>
Launchers and Choosers . . . . .	187
Search extensibility . . . . .	192
App Connect . . . . .	193
App Instant Answer . . . . .	200
Audio and video APIs . . . . .	202
Media playback . . . . .	203
The <i>MediaPlayerLauncher</i> . . . . .	203
The <i>MediaElement</i> class . . . . .	204
The <i>MediaStreamSource</i> and <i>ManagedMediaHelpers</i> classes . . . . .	205
The <i>MediaElement</i> controls . . . . .	207
Audio input and manipulation . . . . .	211
The <i>SoundEffect</i> and <i>SoundEffectInstance</i> classes . . . . .	211
Audio input and the microphone . . . . .	214
The <i>DynamicSoundEffectInstance</i> class . . . . .	223
Music and Videos Hub . . . . .	227
The Clipboard API . . . . .	230
Summary . . . . .	232
<b>Chapter 6 Sensors</b>	<b>233</b>
Orientation . . . . .	233
Phone hardware . . . . .	236
Sensor APIs . . . . .	237
The accelerometer . . . . .	238
Reactive extensions . . . . .	242
The Level Starter Kit . . . . .	245
Shake . . . . .	252
Compass . . . . .	256

Gyroscope .....	261
Motion APIs .....	266
Summary .....	271
<b>Chapter 7 Web connectivity</b>	<b>273</b>
The <i>WebClient</i> and <i>HttpRequest</i> classes .....	273
Using the async pack .....	278
Using the Task Parallel Library .....	280
The <i>WebBrowser</i> control .....	281
Local webpages .....	282
Integrating with JavaScript .....	284
Live SDK .....	287
SkyDrive .....	290
Facebook .....	294
Twitter .....	300
The Data Sense feature .....	312
Summary .....	314
<b>Chapter 8 Web services and the cloud</b>	<b>315</b>
Web services .....	315
Localhost and the emulator .....	317
Connecting to the web service .....	321
SOAP vs. REST .....	323
WCF data services .....	325
Creating an OData client .....	326
Filtered queries .....	329
Dynamic query results .....	331
Paging the data .....	332
Caching the data .....	333
JSON-formatted data .....	337
Web service security .....	339

Windows Azure . . . . .	340
Windows Azure web services. . . . .	341
Windows Azure Mobile Services for Windows Phone. . . . .	345
Summary. . . . .	348
<b>Chapter 9 Background agents</b>	<b>349</b>
Background tasks . . . . .	349
Alarms and reminders . . . . .	350
Alarms . . . . .	351
Reminders . . . . .	354
The Background Transfer Service. . . . .	358
Generic Background Agents . . . . .	362
GBA components. . . . .	366
Updating tiles . . . . .	373
The lock-screen background . . . . .	378
Lock-screen notifications . . . . .	382
Background audio. . . . .	384
Background audio application. . . . .	388
Background audio agent . . . . .	392
Summary. . . . .	394
<b>Chapter 10 Local storage and databases</b>	<b>395</b>
Local storage . . . . .	395
Isolated storage APIs. . . . .	395
Windows Runtime storage . . . . .	402
Win32 APIs . . . . .	407
LINQ-to-SQL . . . . .	411
Defining the database in code. . . . .	412
Performing queries . . . . .	416
Create/update/delete . . . . .	416
Associations. . . . .	418
Handling schema changes . . . . .	423
Prepopulating a reference database . . . . .	427
Maximizing LINQ-to-SQL performance. . . . .	431



The Windows Phone Performance Analysis Tool . . . . .	493
Profiling native code . . . . .	502
Performance best practices. . . . .	503
Keep the user engaged. . . . .	504
Stay off of the UI thread . . . . .	504
Simplify the visual tree . . . . .	505
Easy wins with photos . . . . .	506
Use <i>LongListSelector</i> instead of <i>ListBox</i> for large collections . . . .	506
Use panorama thoughtfully. . . . .	507
Summary. . . . .	508

## **Chapter 13 Porting to Windows Phone 8 and multitargeting      509**

Lighting up a Windows Phone 7 App with Windows Phone 8 features . . . . .	509
Creating Windows Phone 8 tiles in a Windows Phone 7 App . . . .	510
Quirks mode and breaking changes . . . . .	520
Quirks mode. . . . .	520
Breaking changes. . . . .	521
Managing platform-specific projects . . . . .	528
Create distinct apps for Windows Phone 7.1 and Windows Phone 8 . . . . .	529
Share a Windows Phone 7.1 library. . . . .	530
Link source code files . . . . .	531
Create abstract base classes and partial classes . . . . .	534
Windows Phone 7.8 SDK . . . . .	536
Test coverage for Windows Phone 7.x apps. . . . .	537
Summary. . . . .	538

## **Chapter 14 Tiles and notifications      539**

Tile sizes and templates. . . . .	539
Secondary tiles . . . . .	546
Pinning tiles. . . . .	550
Cross-targeting Windows Phone 7. . . . .	557

Push notifications .....	558
Push notification server .....	561
Push notification client .....	565
Registration web service .....	569
Additional server features .....	573
Batching intervals .....	574
XML payload .....	575
Response information .....	576
Additional client features .....	576
The <i>ErrorOccurred</i> event .....	576
User opt-in/out .....	578
Implementing a push viewmodel .....	579
Push notification security .....	584
Summary .....	585

## **Chapter 15 Contacts and calendar 587**

Contacts .....	587
Understanding the People Hub .....	587
Querying device contacts .....	590
Adding contact information .....	602
Creating a custom contacts store .....	605
Calendar .....	614
Querying the calendar .....	614
Creating a new appointment .....	618
Summary .....	619

## **Chapter 16 Camera and photos 621**

Acquiring a single photo .....	621
Working with the media library .....	624
Reading photos .....	624
Adding new images .....	629

Capturing photos .....	633
The <i>PhotoCamera</i> class .....	634
The <i>PhotoCaptureDevice</i> class (Windows Phone 8 only) .....	639
Extending the Photos Hub .....	649
Apps pivot .....	650
The photo apps picker and photo edit picker .....	651
Lenses .....	655
Launching from the camera .....	656
Rich media editing .....	659
Sharing photos .....	661
Share picker .....	661
Auto-upload .....	662
Summary .....	665

## **Chapter 17 Networking and proximity 667**

Sockets .....	667
The Windows Runtime sockets API .....	668
.NET sockets .....	675
The WinSock API .....	680
Finding your app on nearby devices .....	680
Finding peers through Bluetooth .....	683
Finding peers through tap-to-connect .....	686
Reconnecting without retapping .....	689
Tap-to-connect with a Windows 8 app .....	694
Connecting to other Bluetooth devices .....	695
NFC 696	
Reading and writing NFC tags .....	697
Sending a URI across devices .....	702
Summary .....	705

<b>Chapter 18 Location and maps</b>	<b>707</b>
Architecture	707
Determining the current location (Windows Phone 7)	708
Bing maps (Windows Phone 7)	712
The Bing <i>Map</i> control	712
Bing maps web services	716
Bing Maps Launchers	720
Getting location (Windows Phone 8)	722
Maps API (Windows Phone 8)	727
The <i>Map</i> control	727
Route and directions	734
Maps Launchers	739
Continuous background execution (Windows Phone 8)	741
Testing location in the simulator	747
Location best practices	748
Summary	749

## PART III NEW WINDOWS PHONE 8 FEATURES

---

<b>Chapter 19 Speech</b>	<b>753</b>
Voice commands	753
The GSE	754
Building a simple voice commands App	756
Updating <i>PhraseLists</i> at run time	766
Speech recognition in apps	767
Simple recognition with built-in UX	767
Customizing the recognizer UI	769
Adding custom grammars	770
Text-to-Speech	776
Simple TTS with <i>SpeakTextAsync</i>	777
Taking control with speech synthesis markup language	777

Putting it together to talk to your apps .....	781
Summary.....	784
<b>Chapter 20 The Wallet</b>	<b>785</b>
Understanding the Wallet.....	785
The Wallet and Near Field Communication .....	785
The Wallet Hub.....	786
The Wallet object model .....	787
Managing deals .....	787
Transaction items.....	796
Payment instruments.....	806
Wallet agents.....	806
Summary.....	809
<b>Chapter 21 Monetizing your app</b>	<b>811</b>
Advertising.....	811
Integrating the Ad Control.....	812
Ad Control design guidelines.....	816
Registering with Microsoft pubCenter.....	819
Trial mode.....	823
In-app purchase .....	829
Division of responsibilities .....	830
Types of IAP content .....	831
Configuring the mock IAP library .....	832
Purchasing durable content .....	835
Purchasing consumable content .....	840
Configuring IAP inventory .....	841
Summary.....	844

<b>Chapter 22 Enterprise apps</b>	<b>845</b>
Windows Phone for business . . . . .	845
Managed vs. unmanaged phones . . . . .	847
MDM and managed phones . . . . .	847
Managed enrollment. . . . .	847
Unmanaged phones. . . . .	849
Unmanaged enrollment . . . . .	850
Company Apps . . . . .	855
Building a company hub app . . . . .	856
Summary. . . . .	863

---

**PART IV NATIVE DEVELOPMENT AND WINDOWS PHONE 8 CONVERGENCE**

---

<b>Chapter 23 Native development</b>	<b>867</b>
Native code overview. . . . .	867
When to use native code . . . . .	868
An introduction to modern C++ . . . . .	869
Smart pointers . . . . .	869
Foreach loops . . . . .	872
Lambdas. . . . .	872
Scoped, strongly typed <i>enums</i> . . . . .	874
What about properties and events? . . . . .	875
Managed-native interop . . . . .	878
Creating the Windows Runtime component. . . . .	880
Consuming the Windows Runtime component from C# . . . . .	885
Debugging mixed-mode projects. . . . .	888
Writing asynchronous code in C++ . . . . .	888
Moving synchronous code to the background. . . . .	889
Providing progress updates . . . . .	890
Cancelling asynchronous operations . . . . .	892

Using Windows Runtime classes in C++ . . . . .	896
Win32 API. . . . .	898
Component Object Model (COM) . . . . .	899
Supported platform COM libraries . . . . .	899
Using custom COM libraries. . . . .	900
Summary. . . . .	900

## **Chapter 24 Windows 8 convergence 901**

Windows 8 and Windows Phone 8 compared. . . . .	901
Developer tools and supported languages. . . . .	902
Application models . . . . .	903
Frameworks. . . . .	904
Background processing. . . . .	909
Authoring Windows Runtime components on Windows Phone. .911	
Sharing code between Windows and Windows Phone . . . . .	911
Deciding when and how to share code . . . . .	911
Code sharing and Model-View-ViewModel . . . . .	913
Sharing .NET code by using the PCL. . . . .	914
Linked files and conditional compilation. . . . .	919
Sharing managed code with partial classes. . . . .	922
Sharing native code. . . . .	924
Sharing XAML-based UI with user controls. . . . .	927
Summary. . . . .	932

## **Chapter 25 Games and Direct3D 933**

Direct3D primer. . . . .	933
Direct3D differences on Windows Phone. . . . .	935
Visual Studio project types . . . . .	936
Direct3D and XAML projects . . . . .	937

Structure of the basic Direct3D app . . . . .	940
The <i>CoreApplication</i> and <i>CoreWindow</i> classes . . . . .	940
Starter code classes . . . . .	941
Initializing Direct3D . . . . .	945
A brief word about the Component Object Model . . . . .	946
DirectX hardware feature levels . . . . .	949
Update and render. . . . .	950
Minimal Direct3D app . . . . .	954
Touch input . . . . .	959
Direct2D and DirectXTK . . . . .	961
Summary. . . . .	964
<i>Index</i> . . . . .	965

---

**What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[microsoft.com/learning/booksurvey](https://microsoft.com/learning/booksurvey)



# Foreword

Given that you've picked up this book, it's probably safe to assume you're familiar with the Windows Phone operating system. Maybe you've seen a television or magazine ad showcasing unique features such as customizable Live tiles; maybe you've watched one of the "Smoked by Windows Phone" videos online; or maybe you're even carrying a Windows Phone with you right now as your daily driver. And, with a title like *Windows Phone 8 Development Internals*, it's also pretty safe to assume you're interested in developing for Windows Phone, whether for fun, education, profit, or just because your boss asked you to.

Whatever the reason for your interest in Windows Phone development, you've selected the right book to get you started (or to help take your expertise to the next level). This book is jam-packed with expert advice, best practices, design guidelines, and clear explanations of features that will teach you not just the "how" and "what" of building great Windows Phone applications, but also the "why." And, as you'd expect from any modern development book, there's a large set of sample code that serves both to illustrate the features and concepts covered within as well as to serve as the starter code for your own applications.

Released in late 2012, Windows Phone 8 is the next evolution of the Windows Phone operating system. It offers many new end-user features such as the more customizable Start Screen, family-friendly Kids Corner, background turn-by-turn directions, deeply integrated VoIP functionality, Enterprise application support, and so much more. But today's smartphone experience is defined as much, if not more, by the rich set of applications and games that are available to download—and that's where you come in!

Windows Phone 8 is an even bigger release for developers than it is for users. Whereas Windows Phone 7 and 7.1 focused strictly on managed-code development with Microsoft Silverlight and XNA, Windows Phone 8 adds native-code development with C and C++ and facilitates the creation of high-performance games with Direct3D and XAudio2. Windows Phone 8 also shares a common kernel and many APIs with its "big brother" Windows 8 operating system, making it possible for developers to share much of their code (and their development costs) across phones, tablets, laptops, and desktops.

In addition to the incremental updates to Microsoft .NET and XAML and the large number of APIs borrowed from Windows and DirectX, Windows Phone 8 also adds APIs for more than 60 brand-new scenarios such as speech synthesis and recognition, high-performance camera access, and a world-class mapping platform. Although there's no effective way to quantify the surface area of a platform as complex as Windows Phone, it would be fair to say that this release roughly doubles the amount of functionality available to developers from the previous release—that's a lot of functionality!

There's a saying in software development that it takes 80 percent of the time to do 80 percent of the work, and another 80 percent of the time to do the last 20 percent. That's because there's so much more to building a great application than just knowing which APIs to call: your application needs to be easy to use, secure, reliable, and aesthetically pleasing. It must also perform well, protect the user's privacy, and (in many cases) generate a profit. Andrew and Sean cover all these areas and more in this book, focusing not just on a long list of APIs but rather on the entirety of the application lifecycle, from design and coding, through analysis and monetization. It's that last 20 percent that turns a good app into a great app, and this is what will set your applications apart from the others in a crowded marketplace.

Finally, I want to thank Andrew and Sean for giving me the opportunity to review this book and do my part to help developers such as yourself participate in (and hopefully profit from) the burgeoning Windows Phone application ecosystem. I'll close with the same words I used in the foreword to Andrew's previous Windows Phone development book: I also learned a lot while reviewing this book, and I know you will too.

Peter Torr  
Program Manager in the  
Windows Phone Application Platform team

# Introduction

Smartphone technology is evolving at a rapid pace. Just two short years after the release of Windows Phone 7—which itself was a major departure from the previous version—Microsoft released Windows Phone 8. This is a huge release. Almost everything in the stack, from the hardware up to the application platform layer, has changed dramatically. The operating system moves towards convergence with the Windows 8 desktop operating system via a shared core, and the app platform includes support for native development. At the same time, the app platform maintains an extremely high degree of backward compatibility, ensuring that Windows Phone 7 apps continue to work on Windows Phone 8.

The platform has been designed from the ground up to support an all-encompassing, integrated, and delightful user experience (UX). There is considerable scope for building compelling apps on top of this platform, and Windows Phone is well-positioned as an opportunity for developers to build applications that can make a real difference in people’s lives.

*Windows Phone 8 Development Internals* covers the breadth of application development for the Windows Phone 8 platform. You can build applications for Windows Phone 8 by using either managed code (in C# or Microsoft Visual Basic) or native code (in C++). This book covers both C# managed development and C++ native development. The primary development and design tools are Microsoft Visual Studio and Microsoft Expression Blend; this book focuses on Visual Studio.

Each chapter covers a handful of related features. For each feature, the book provides one or more sample applications and walks you through the significant code. This approach can both help you understand the techniques used and also the design and implementation choices that you must make in each case. Potential pitfalls are called out, as are scenarios in which you can typically make performance or UX improvements. An underlying theme is that apps should conform not only to the user interface design guidelines, but also to the notion of a balanced, healthy phone ecosystem.

## Who should read this book

---

This book is intended to help existing developers understand the core concepts, the significant programmable feature areas, and the major techniques in Windows Phone development. It is specifically aimed at existing C# and C++ developers who want to get up to speed rapidly with the Windows Phone platform. Developers who have experience with other mobile platforms will find this book invaluable in learning the ins and outs of Microsoft’s operating system and will find that the chapters that focus on native development will foster an easy transition. For the chapters that focus on managed development, native developers will likely need additional resources to pick up the C# and XAML languages.

Chapter 1, “Vision and architecture,” covers the basic architecture of the platform, while most of the chapters delve deeply into the internal system behavior. This is the type of knowledge that helps to round out your understanding of the platform and inform your design decisions, even though, in some cases, the internal details have no immediate impact on the exposed API.

## Assumptions

The book assumes that you have a reasonable level of experience with developing in C# and/or in C++. The book does not discuss basic language constructs, nor does it cover the basics of how to use Visual Studio, the project system, or the debugger, although more advanced techniques, and phone-specific features are, of course, explained in detail. Previous knowledge of XAML is useful for the managed chapters, and some exposure to COM is useful for the native chapters, but neither is essential.

Although many component-level diagrams are presented as high-level abstractions, there are also many sections that describe the behavior of the feature in question through the use of UML sequence diagrams. It helps to have an understanding of sequence diagrams, but again, that’s not essential, because they are fairly self-explanatory.

## Who should not read this book

---

This book is not intended for use by application designers—that is, if designers are defined as developers who use Expression Blend—although designers might find it useful to understand some of the issues facing developers in the Windows Phone application space. Although Windows Phone 8 does maintain support for XNA, you cannot create new XNA projects in Visual Studio 2012; therefore, this book does not cover XNA development at all.

## Organization of this book

---

This book is divided into four parts:

- **Section I** Core Features
- **Section II** Windows Phone 7 to Windows Phone 8
- **Section III** New Windows Phone 8 Features
- **Section IV** Native Development with Windows Phone 8

As of this writing, there are still many more Windows Phone 7 phones in existence than Windows Phone 8 phones. The Windows Phone Store contains well over 140,000 apps, almost all of which target Windows Phone 7. For this reason, the first eight chapters focus on the basic infrastructure, programming model, and the core features that are common to both versions. Where there are material differences, those are called out along with references to the later chapter where the version 8 behavior is explained in detail.

Next, Section II covers both the features that are significantly different between version 7 and version 8. This section also discusses the process of porting apps from version 7 to version 8, as well as how to develop new apps that target both versions.

Section III covers the major new features in version 8 that did not exist at all in version 7. These include speech functionality, wallet, in-app purchase, and enterprise applications.

Finally, whereas the first three sections concentrate on managed development, Section IV focuses purely on native development. This section includes coverage of native-managed interoperability, convergence between Windows Phone 8 and Windows 8, threading, and integration with the Windows Phone platform via native code.

## Conventions and features in this book

---

This book presents information by using conventions designed to make the information readable and easy to follow.

- In some cases, especially in the early chapters, application code is listed in its entirety. More often, particularly later in the book, only the significant code is listed. Wherever code has been omitted for the sake of brevity, this is called out in the listing. In all cases, you can refer to the sample code that accompanies this book for complete listings.
- In the XAML listings, attributes that are not relevant to the topic under discussion, and that have already been explained in previous sections are omitted. This omission applies, for example, to *Grid.Row*, *Grid.Column*, *Margin*, *FontSize*, and similarly trivial attributes. In this way, you can focus on the elements and attributes that do actually contribute to the feature at hand, without irrelevant distractions.
- Code identifiers (the names for classes, methods, properties, events, enum values, and so on) are all italicized in the text.
- In the few cases where two or more listings are given with the explicit aim of comparing alternative techniques (or “before” and “after” scenarios), the differences appear in bold.
- Boxed elements such as Notes, Tips, and other reader aids provide additional information or alternative methods for completing a step successfully.
- Text that you should type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Press Alt+Tab” means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (for example, File | Close), means that you should select the first menu or menu item, and then the next, and so on.

# System requirements

---

You can build and run the accompanying sample code, or you can create your own solutions from scratch, following the instructions in the text. In either case, you will need the following hardware and software to create the sample applications in this book:

- **Windows 8** You cannot install the Visual Studio 2012 tools for Windows Phone development on any other version of Windows.
- **The Windows Phone SDK version 8** This is a free download that includes Visual Studio 2012 Express Edition and all other standard tools, as detailed in Chapter 1.
- Some of the server-side sample projects require Visual Studio Professional, but all the Windows Phone samples work with Visual Studio Express.
- Installing the SDK requires 4 GB of free disk space on the system drive. If you use the profiler for an extended period, you will need considerably more disk space.
- 4 GB RAM (8 GB recommended).
- The Windows Phone emulator is built on the latest version of Microsoft Hyper-V, which requires a 64-bit CPU that includes Second Level Address Translation (SLAT). If you have only a 32-bit computer (or a 64-bit computer without SLAT support), you can still install the SDK, and you can test apps as long as you have a developer-unlocked phone.
- A 2.6 GHz or faster processor (4GHz or 2.6GHz dual-core is recommended).
- Internet connection to download additional software or chapter examples and for testing web-related applications.

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2012 and to install or configure features such as Internet Information Services, if not already installed.

For the latest requirements, visit the Windows Phone SDK download page at:

*<http://dev.windowsphone.com>*

## Code samples

---

All of the chapters in this book include multiple sample solutions that you can use interactively to try out new material learned in the main text. You can download all the sample projects from the following page:

*<http://aka.ms/WinPhone8DevInternals/files>*

Follow the instructions to download the WP8DevInternals.zip file.

## Installing the code samples

Perform the following steps to install the code samples on your computer so that you can refer to them while learning about the techniques that they demonstrate.

1. Unzip the WP8DevInternals.zip file that you downloaded from the book's website to any suitable folder on your local hard disk. The sample code expands out to nearly 200 MB, and you will need even more space for the binaries if you choose to build any of the samples.
2. If prompted, review the displayed end-user license agreement. If you accept the terms, select the accept option and then click Next.



**Note** If the license agreement doesn't appear, you can access it from the same webpage from which you downloaded the WP8DevInternals.zip file.

## Using the code samples

When you unzip the sample code, this creates a number of subfolders, one for each chapter. Within each chapter's subfolder there are further subfolders. In most cases, there is one subfolder per application (or per version of an application), but in some cases, multiple applications are grouped together; for example, where there is a server-side application as well as a client-side application in the solution.

All of the samples are complete, fully-functioning applications. Note, however, that in some cases, you might need to update assembly references, depending on where you install the SDK as well as where you install supplementary libraries and frameworks that don't ship with the main SDK.

For samples that demonstrate the use of some supplementary framework, you will need to download and install that framework so that you can reference its assemblies. Also note that, in some cases, this requires a user ID, such as for Facebook or Google Analytics, as described in the relevant sections. In all cases, as of this writing, you can sign up for the ID without charge.

## Acknowledgments

---

The Windows Phone development space is truly inspiring, and the Windows Phone teams at Microsoft are chock-full of smart, helpful people. The list of folks who helped us prepare this book is very long. We'd particularly like to thank Peter Torr for doing all the heavy lifting in the technical review, and Russell Jones, our intrepid editor at O'Reilly Media.

In addition, we'd like to thank all the other people who answered our dumb questions, and corrected our various misinterpretations of the internal workings of the platform, especially Tim Kurtzman, Wei Zhang, Jason Fuller, Abolade Gbadegesin, Avi Bathula, Brian Cross, Alex McKelvey and Adam Lydick.

## Errata & book support

---

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

*<http://aka.ms/WinPhone8DevInternals/errata>*

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *[mspinput@microsoft.com](mailto:mspinput@microsoft.com)*.

Please note that product support for Microsoft software is not offered through the addresses above.

## We want to hear from you

---

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

*<http://www.microsoft.com/learning/booksurvey>*

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

## Stay in touch

---

Let's keep the conversation going! We're on Twitter: *<http://twitter.com/MicrosoftPress>*

**PART I**

# Core Features

<b>CHAPTER 1</b>	Vision and architecture . . . . .	3
<b>CHAPTER 2</b>	App model and navigation . . . . .	33
<b>CHAPTER 3</b>	UI visuals and touch. . . . .	77
<b>CHAPTER 4</b>	Data binding and MVVM . . . . .	139
<b>CHAPTER 5</b>	Phone and media services . . . . .	187
<b>CHAPTER 6</b>	Sensors. . . . .	233
<b>CHAPTER 7</b>	Web connectivity. . . . .	273
<b>CHAPTER 8</b>	Web services and the cloud. . . . .	315
<b>CHAPTER 9</b>	Background agents . . . . .	349
<b>CHAPTER 10</b>	Local storage and databases . . . . .	395



# Vision and architecture

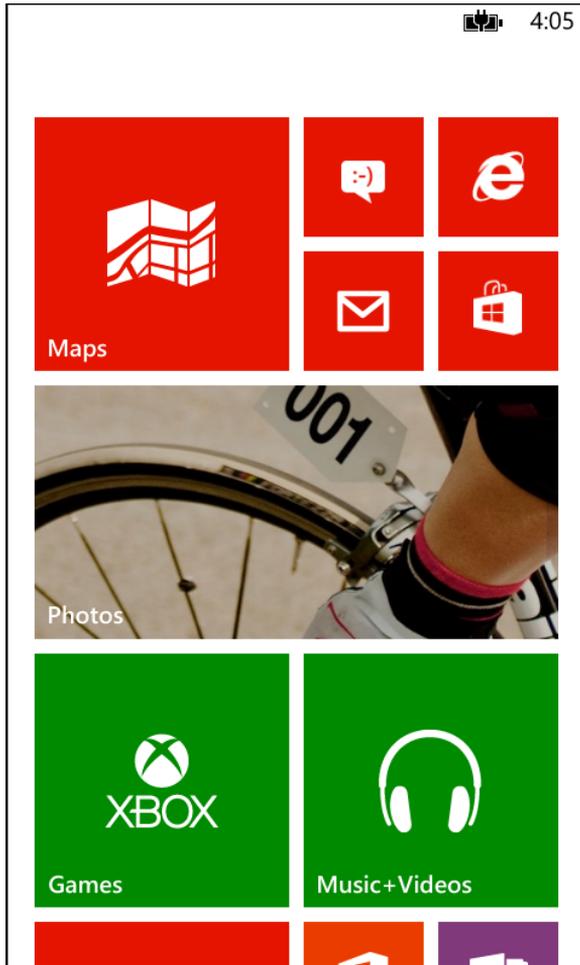
This chapter covers three core topics: the principles behind the Windows Phone UI and the role that Windows Phone Store apps play in it; a primer on the architecture of the Windows Phone development platform; and an overview of what is required to build and deliver Windows Phone apps. Together, these topics form a critical foundation that will support the detailed examinations of individual platform features that follow in subsequent chapters. And, just so you don't leave this chapter without getting your hands a little bit dirty, you will walk through a simple "Hello World" project to ensure that you're all set to tackle the more involved topics ahead.

## A different kind of phone

---

When Windows Phone 7 was released in the fall of 2010, it represented a significant departure not only from previous Microsoft mobile operating systems, but also from every other mobile operating system (OS) on the market. The user interface was clean, bold, and fluid, with a strong focus on the user's content, rather than app chrome. The Start screen (see Figure 1-1) provided a level of personalization available nowhere else. Live tiles provided key information at a glance as well as the ability to start not only apps, but specific parts of those apps, such as opening a favorite website, perhaps, or checking a friend's Facebook status. The developer platform offered unrivalled efficiency and familiar tools. It also gave app developers the ability to extend core phone experiences rather than building isolated apps.

With Windows Phone 8, Microsoft has significantly expanded the capabilities of the OS, but the fundamental philosophy remains the same. Indeed, much of the original Windows Phone philosophy is now being adopted in the core Windows OS, Microsoft Office, Microsoft Xbox, and other Microsoft products, making it all the more valuable to understand its basic tenets.



**FIGURE 1-1** The distinctive Windows Phone Start screen offers unrivaled personalization.

## The user interface

The distinctive Windows Phone user interface (UI) is built upon a set of core principles. Understanding these principles will help you to understand not only why the phone looks the way it does, but how you can build beautiful apps that integrate well into the overall experience. After all, in the mobile app marketplace, it is generally not the app with the most features that wins out, but the one which is the easiest and the most enjoyable to use.

For an in-depth review of these principles, watch the talk from Jeff Fong, one of the lead designers for Windows Phone on Channel9 (<http://channel9.msdn.com/blogs/jaime+rodriguez/windows-phone-design-days-metro>).

## Light and simple

The phone should limit clutter and facilitate the user's ability to focus on completing primary tasks quickly. This is one of the principles that drew significant inspiration from the ubiquitous signage in major mass transit systems around the world. In the same way that a subway station uses signs that are bold and simple to comprehend in order to move hundreds of thousands of people through a confined space quickly, Windows Phone intelligently reveals the key information that the user needs among the dozens of things happening at any one time on the phone, while keeping the overall interface clean and pleasing to the eye.

## Typography

One element that is common across virtually any user interface is the presence of text. Sadly, it is often presented in an uninteresting way, focusing on simply conveying information rather than making the text itself beautiful and meaningful. Windows Phone uses a distinct font, Segoe WP, for all of its UI. It also relies on font sizing as an indicator of importance. The developer platform provides built-in styles for the various flavors of the Segoe WP typeface, making it simple to incorporate into your app.

## Motion

Someone who only experienced the Windows Phone UI through screenshots would be missing out on a significant part of what makes it unique: motion. Tactical use of motion—particularly when moving between pages—not only provides an interesting visual flourish at a time when the user could not otherwise be interacting with the phone, but it also is a clear connection between one experience and the next. When the user taps an email in her inbox and sees the name of the sender animate seamlessly into the next screen, it provides direct continuity between the two views such that there can be no doubt about what is happening.

## Content, not chrome

If you've ever tried browsing around a new Windows Phone that has not yet been associated with a Microsoft Account, you'll find that there isn't very much to look at. Screen after screen of white text on a black background (or the reverse if the phone is set to light theme), punctuated only by the occasional endearing string—"It's lonely in here."—encouraging you to bring your phone to life. The moment when you sign in with a Microsoft Account, however, everything changes. The phone's UI recedes to the background and your content fills the device; contacts, photos, even your Xbox Live avatar all appear in seconds and help to make your phone incredibly personal.

## Honesty in design

This is perhaps the most radical of the Windows Phone design principles. For years, creators of graphical user interfaces (GUIs) have sought to ease the transition of users moving critical productivity tasks from physical devices to software by incorporating a large number of *skeuomorphic* elements in

their designs. Skeuomorphic elements are virtual representations of physical objects, such as a legal pad for a note-taking app or a set of stereo-like knobs for a music player. Windows Phone instead opts for a look that is “authentically digital,” providing the freedom to design UI that’s tailored to the medium of a touch-based smartphone, breaking from the tradition of awkwardly translating a set of physical elements into the digital realm.

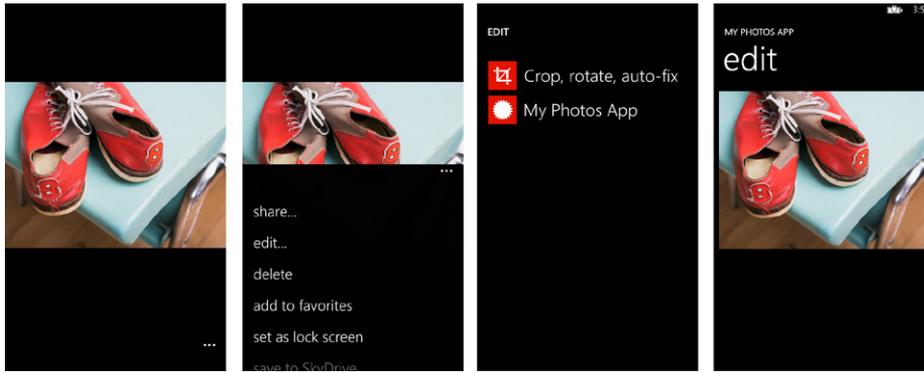
## The role of apps

In addition to its distinctive UI, Windows Phone takes a unique approach to the role of Windows Phone Store apps in the experience. Historically, mobile operating systems only provided simple entry points for users to start apps—Apple’s iPhone is the canonical example of this, with each app able to display one and only one icon on the phone’s home screen. Although this model is simple and clean, it creates a disjointed environment that obstructs how users want to interact with their content.

With Windows Phone, Microsoft made an explicit shift from the app-focused model to a content and experience-focused model, in which the user is encouraged to think primarily about what he wants to do, rather than how he wants to do it. Something as simple as making a phone call, for example, should not require remembering which cloud services your friend is a member of so that you can start the appropriate app to look up her phone number. Rather, you should simply be able to launch a unified contacts experience which aggregates information from all of your apps and services.

The content and experience-focused approach doesn’t make Windows Phone Store apps less important, it just changes how they fit in the experience. Windows Phone provides an immersive “hub” experience for each of the primary content types on the phone—photos, music, people, and so on—and each of these hubs offers a rich set of extensibility points for apps to extend the built-in experience. These extensibility points offer additional ways for users to invoke your app, often with a specific task in mind for which you might be uniquely positioned to handle. Table 1-1 lists the extensibility points supported in Windows Phone 7.1 and Windows Phone 8.

Consider photos as an example. There are thousands of apps in the Windows Phone Store that can do something with photos: display them, edit them, or post them to social networks. In a purely app-focused world, the user must decide up-front which tasks he wants to perform and then remember which app would be the most appropriate for that task. In the Windows Phone model, he simply starts the Photos hub, in which he will not only see all of his photos, aggregated across numerous sources, but all of the apps that can do something with those photos. Figure 1-2 shows an example of the photos extensibility in Windows Phone, with “My Photos App” registering as a photo editor, which the user can access through the Edit entry on the app bar menu for a given photo.



**FIGURE 1-2** With Windows Phone, apps can extend built-in experiences such as the photo viewer.

**TABLE 1-1** Windows Phone extensibility points

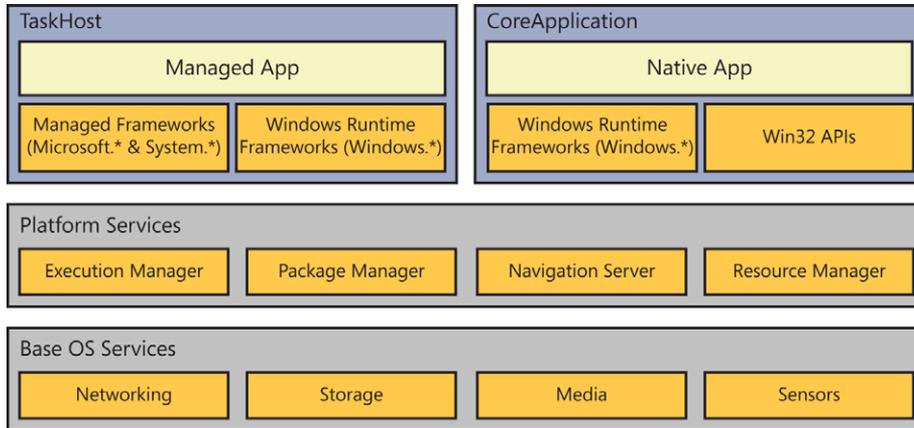
App	Extensibility point	Windows Phone 7.1	Windows Phone 8.0
Music & Videos	Now playing tile	✓	✓
Music & Videos	History list	✓	✓
Music & Videos	New List	✓	✓
Photos	Apps pivot	✓	✓
Photos	Photo viewer – share	✓	✓
Photos	Photo viewer – apps	✓	
Photos	Photo viewer – edit		✓
Search	Search quick cards	✓	✓
Wallet	Wallet items—coupons, transactions, loyalty cards		✓
Lock screen	Background photo		✓
Lock screen	Quick status		✓
Lock screen	Detailed status		✓
Speech	Voice commands		✓
People	Custom contact stores		✓
Camera	Lenses		✓
Maps	Navigation		✓

# Windows phone architecture

Now that you understand the user experience (UX) philosophy that drives Windows Phone, it's time to dig a little bit deeper and review some of the core parts of the phone's architecture.

## Platform stack

No chapter on architecture would be complete without the venerable block diagram, and we don't aim to disappoint. Figure 1-3 shows the basic logical components of the Windows Phone 8 platform.



**FIGURE 1-3** Windows Phone 8 layers two app models on top of a shared set of platform and OS services.

At the top of the stack sit two distinct app models. The box labeled "TaskHost" represents the XAML app model, which has been the primary model since the launch of Windows Phone 7. To its right is a box labeled "CoreApplication," a new app model for Windows Phone, which is a subset of the new Windows 8 app model. In the Windows Phone 8 release, this app model only supports pure native apps using Direct3D for UI.



**Note** Although Win32/COM APIs are only shown in the CoreApplication box in Figure 1-3, they are actually callable by managed apps, as well, as long as they are wrapped in a custom Windows Runtime component.

The two app models rely on a shared set of core platform services. For the most part, Store apps only ever see these services indirectly, but because they play a major role in ensuring that those apps work properly and, after all, this is an “Internals” book, we should explore them briefly.

- **Package Manager** The Package Manager is responsible for installing/uninstalling apps and maintaining all of their metadata throughout the app lifecycle. It not only keeps track of which apps are installed and licensed, it also persists information about any app tiles that the user might have pinned to the Start screen and the extensibility points for which an app might have registered so that they can be surfaced in the appropriate places in the OS.
- **Execution Manager** The Execution Manager controls all of the logic associated with an app’s execution lifetime. It creates the hosting process for the app to run in and raises the events associated with app startup/shutdown/deactivation. It performs a similar task for background processes, which also includes proper scheduling of those tasks.
- **Navigation Server** The Navigation Server manages all of the movement between foreground apps on the phone. When you tap an app tile on the Start screen, you are navigating from the “Start app” to the app you chose, and the Navigation Server is responsible for relaying that intent to the Execution Manager so that the chosen app can be started. Likewise, when you press and hold the Back key and choose an app that you started previously, the Navigation Server is responsible for telling the Execution Manager which app to reactivate.
- **Resource Manager** The Resource Manager is responsible for ensuring that the phone is always quick and responsive by monitoring the use of system resources (especially CPU and memory) by all active processes and enforcing a set of constraints on them. If an app or background process exceeds its allotted resource pool, it is terminated to maintain the overall health of the phone.

All of this is built on top of a shared Windows Core, which we will describe in more detail later in this chapter.

## App types

So far, we’ve been referring to Windows Phone apps generically, as if they were all built and run in basically the same way. In fact, Windows Phone 8 supports several different app flavors, depending on your needs. These are described in Table 1-2.

**TABLE 1-2** Windows Phone 8 app types

App type	Description	Languages supported	UI framework	APIs supported
XAML	The most common app type for Windows Phone 7.x. These apps are exclusively written in XAML and managed code.	C# Visual Basic	XAML	Microsoft .NET Windows Phone API Windows Runtime API
Mixed mode	<p>These apps follow the XAML app structure but allow for the inclusion of native code wrapped in a Windows Runtime component.</p> <p>This is well-suited for apps for which you want to reuse an existing native library, rather than rewriting it in managed code.</p> <p>It is also useful for cases in which you want to write most of the app in native code (including Direct3D graphics) but also need access to the XAML UI framework and some of the features that are only available to XAML apps such as the ability to create and manipulate Start screen tiles.</p>	C# Visual Basic C/C++	XAML Direct3D (via DrawingSurface)	.NET Windows Phone API Windows Runtime API Win32/COM API (within Windows Runtime components)
Direct3D	Best suited for games, pure native apps using Direct3D offer the ability to extract the most out of the phone's base hardware. Also, because they are based on the Windows app model, they offer the greatest degree of code sharing between Windows and Windows Phone.	C/C++	Direct3D	Windows Runtime API Win32/COM API

### What about XNA?

In Windows Phone 7.x, there were two basic app types from which to choose: Microsoft Silverlight and XNA. As described earlier, managed Silverlight applications are fully supported in Windows Phone 8, but what of XNA? In short, the XNA app model is being discontinued in Windows Phone 8. Existing Windows Phone 7.x XNA games (and new games written targeting Windows Phone 7.x), which includes a number of popular Xbox Live titles, will run on 8.0, but developers will not be able to create new XNA games or new Silverlight/XNA mixed-mode apps targeting the Windows Phone 8.0 platform. Many of the XNA assemblies, such as *Microsoft.Xna.Framework.Audio.dll*, will continue to work in Windows Phone 8.0, however. Further, Windows Phone 7.x XNA games are allowed to use some features of Windows Phone 8, such as in-app purchase, using reflection.

## Background processing

When it comes to background execution on a mobile device, users often have conflicting goals. On one hand, they want their apps to continue providing value even when they're not directly interacting with them—streaming music from the web, updating their Live tile with the latest weather data, or providing turn-by-turn navigation instructions. On the other hand, they also want their phones to last at least through the end of the day without running out of battery and for the foreground app they're currently using to not be slowed down by a background process that needs to perform significant computation.

Windows Phone attempts to balance these conflicting requirements by taking a scenario-focused approach to background processing. Rather than simply allowing apps to run arbitrarily in the background to perform all of these functions, the platform provides a targeted set of multitasking features designed to meet the needs (and constraints) of specific scenarios. It is these constraints which ensure that the user's phone can actually last through the day and not slow down unexpectedly while performing a foreground task.

### Background OS services

Windows Phone offers a set of background services that can perform common tasks on behalf of apps.

**Background transfer service** The Background Transfer Service (BTS) makes it possible for apps to perform HTTP transfers by using the same robust infrastructure that the OS uses to perform operations such as downloading music. BTS ensures that downloads are persisted across device reboots and that they do not impact the network traffic of the foreground app.

**Alarms** With the Alarms API, apps can create scenario-specific reminders that provide deep links back into the app's UX. For example, a recipes app might provide a mechanism for you to add an alarm that goes off when it's time to take the main course out of the oven. It might also provide a link that, when tapped, takes the user to the next step in the recipe. Not only does the Alarms API remove the need for apps to run in the background simply to keep track of time, but they can take advantage of the standard Windows Phone notification UI for free, making them look and feel like built-in experiences.

### Background audio agents

Background audio playback is a classic example of scenario-based background processing. The simplest solution to permitting Windows Phone apps to play audio from the background would be to allow those apps to continue running even when the user navigates away. There are two significant drawbacks to this, however:

- Windows Phone already includes significant infrastructure and UI for playing and controlling background audio using the built-in Music & Video app. Leaving every app to build this infrastructure and UI itself involves a significant duplication of effort and a potentially confusing UX.

- A poorly written app running unconstrained in the background could significantly impact the rest of the phone

To deal with these drawbacks, Windows Phone reuses the existing audio playback infrastructure and invokes app code only to provide the bare essentials of playlist management or audio streaming. By constraining the tasks that an audio agent needs to perform, it can be placed in a minimally invasive background process to preserve both the foreground app experience and the phone's battery life.

## Scheduled tasks

Scheduled tasks offer the most generic solution for background processing in Windows Phone apps, but they are still ultimately driven by scenarios. There are two types of scheduled tasks that an app can create, each of which is scheduled and run by the OS, based on certain conditions:

- **Periodic tasks** Periodic tasks run for a brief amount of time on a regular interval—the current configuration is 25 seconds approximately every 30 minutes (as long as the phone is not in Battery Saver mode). They are intended for small tasks which benefit from frequent execution. For example, a weather app might want to fetch the latest forecast from a web service and then update its app tiles.
- **Resource-intensive tasks** Resource-intensive tasks can run for a longer period, but they do not run on a predictable schedule. Because they can have a larger impact on the performance of the device, they only execute when the device is plugged in, nearly fully charged, on Wi-Fi, and not in active use. Resource-intensive agents are intended for more demanding operations such as synchronizing a database with a remote server.

## Continuous background execution for location tracking

In the case of background music playback described earlier, there is very little app code that needs to execute after the initial setup is complete. The built-in audio playback infrastructure handles outputting the actual sound, and the user generally performs tasks such as play, pause, and skip track by using the built-in Universal Volume Control (UVC) rather than reopening the app itself. For the most part, all the app needs to do is provide song URLs and metadata (or streaming audio content) to the audio service.

This is not the case for location tracking and, in particular, turn-by-turn navigation apps. These apps generally need to receive and process up-to-date location information every few seconds to determine whether the user should be turning left or right. They are also likely to offer a rich UX within the app such as a map showing the full route to the destination and the time/distance to go, which will encourage the user to frequently relaunch it. As a result, the audio playback model of using a constrained background task is less suitable in this case. Instead, Windows Phone 8 introduces a concept known as Continuous Background Execution (CBE), which simply refers to the ability of the current app to continue running even if the user navigates away, albeit with a restricted API set.

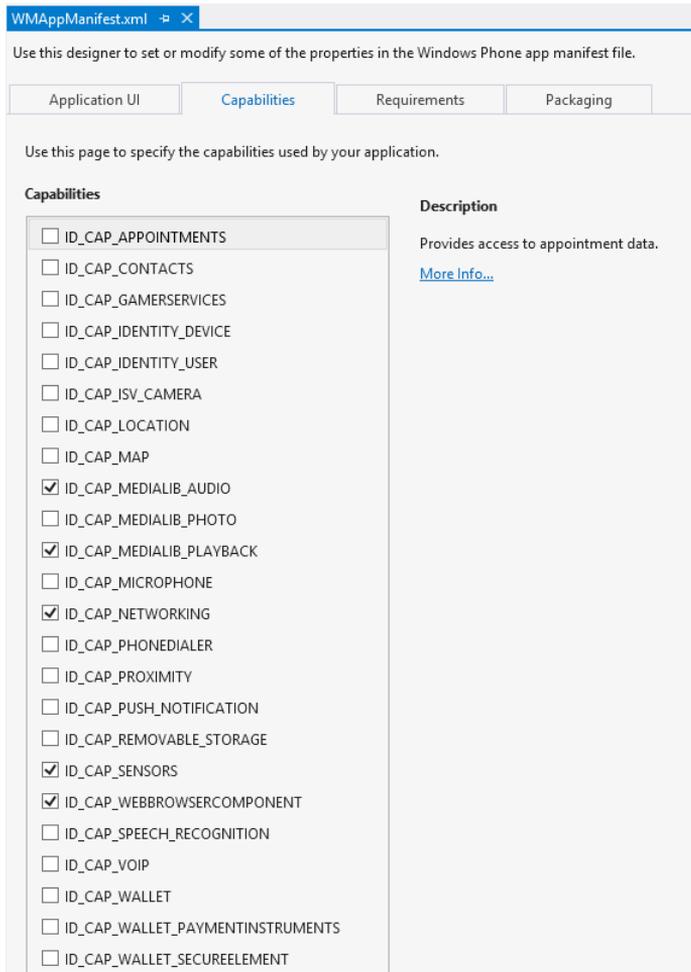
## Security model

Modern smartphones are by far the most personal items that people have ever owned—in the palm of your hand are the names, phone numbers, and addresses of all of your family and friends, thousands of photos, location history, email correspondence, and, increasingly, financial information stored in mobile wallet apps. Ensuring that all of this information remains safe while the phone moves between physical locations and navigates a variety of websites and apps requires a robust security model.

The Windows Phone security model is based on the notion of *security chambers*, which are isolated containers in which processes are created and executed. The chamber is the security principal to which access rights are granted in the system. The system grants those rights based on the long-standing security principle of *least privilege*, which holds that an app should not be granted the rights to do anything beyond what is strictly necessary to perform its stated functions. For example, the email app should not have the ability to arbitrarily start the camera and take a picture, because that is clearly not necessary to perform its core function.

So, how does Windows Phone ensure this principle of least privilege? Every security chamber, whether it contains code owned by Microsoft or by an external software developer, starts out with a limited set of privileges—enough to write a self-contained app such as a calculator or a simple game, but not enough to enable the full range of scenarios consumers expect from a modern smartphone. If an app wants to access resources that reside outside of its chamber, such as sending traffic over the network or reading from the user's contacts, it must be explicitly granted that access via *capabilities*. Capabilities act as a set of access control mechanisms that gate the usage of sensitive resources. The system must explicitly grant capabilities to a chamber.

Windows Phone developers encounter these capabilities directly when building their apps because accessing any privileged resource from your app requires including the appropriate capability in your app manifest. The graphical manifest editor includes a Capabilities tab that lists all of the available options, as shown in Figure 1-4.



**FIGURE 1-4** You select the required capabilities for a chamber in the manifest editor.

Because all of the capabilities listed in the manifest editor are available for Windows Phone Store apps to use, you might ask how the principle of least privilege is being maintained. The answer is that it is the user who decides. The capabilities listed in the manifest are translated into user-readable line items on the Windows Phone Store details page for the app when it's eventually published. The user can then decide whether he feels comfortable installing an app which requires access to a given capability—for example, the user should expect that an app that helps you find nearby coffee shops will need access to your location, but he would probably be suspicious if a calculator app made the same request. Figure 1-5 presents the user-readable capabilities for a weather app. As you can probably guess, “location services” corresponds to ID\_CAP\_LOCATION, and “data services” is the replacement for ID\_CAP\_NETWORKING.



**FIGURE 1-5** Security capabilities are displayed as user-readable strings in an app's details page.

## Capability detection in Windows Phone 8

It's worth mentioning that Windows Phone 8 has introduced a subtle but important change in how capabilities are detected during app ingestion. In Windows Phone 7.x, the capabilities that the app developer included in the manifest that was submitted to the Store were discarded and replaced with a set determined by scanning the APIs used in the app code. In other words, if you included the `ID_CAP_LOCATION` capability in your manifest but never used any of the location APIs in the `System.Device.Location` namespace, that capability would be removed from the final version of your XAP package (XAP [pronounced "zap"] is the file extension for a Silverlight-based application package [.xap]) and the Store details page for your app would not list location as one of the resources it needed. Given this Store ingestion step, there was no reason for a developer to limit the capabilities that her app was requesting during development. Anything that she didn't end up using would simply be discarded as part of her submission.

With the introduction of native code support in Windows Phone 8, this approach is no longer feasible, and developers are now responsible for providing the appropriate list of capabilities in their app manifests. If an app fails to list a capability that is required for the functionality it is providing, the associated API calls will simply fail. On the other hand, if an app requests a capability that it doesn't actually need, it will be listed on its Windows Phone Store details page, potentially giving the user pause about installing it.



**Note** For managed code apps, developers can continue to use the CapDetect tool that ships with the Windows Phone SDK to determine which capabilities they need.

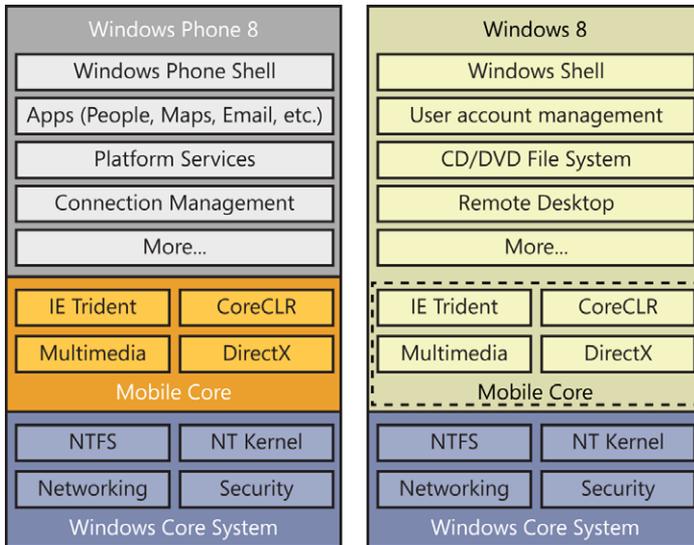
## Windows and Windows Phone: together at last

Even though the distinctive UX described earlier in this chapter did not change significantly between Windows Phone 7 and Windows Phone 8, there have been dramatic shifts happening below the surface. For the first time, Windows Phone is built on the same technology as its PC counterpart. In this section, we describe the two core parts of that change which impact developers: the shared Windows core, and the adoption of the Windows Runtime.

### Shared core

By far the most significant architectural change in Windows Phone 8 is the adoption of a shared core with Windows, but you might be wondering what a "shared core" actually means. In fact, it contains two distinct components. At the very bottom is the Windows Core System, the most basic functions of the Windows OS, including (among other things) the NT kernel, the NT File System (NTFS), and the networking stack. This minimal core is the result of many years of architectural refinement, the goal of which was to provide a common base that could power multiple devices, including smartphones.

Above the Core System is Mobile Core, a set of Windows functionality that is not part of Core System but which is still relevant for a smartphone. This includes components such as multimedia, CoreCLR, and Trident, the rendering engine for Internet Explorer. Figure 1-6 illustrates some of the shared components on which Windows and Windows Phone rely. Note that Mobile Core is only a distinct architectural entity in Windows Phone. Windows contains the same components as Mobile Core, but they are part of a larger set of functionality. This is depicted by a dashed line around the Mobile Core components in the Windows 8 portion of the diagram.



**FIGURE 1-6** Windows 8 and Windows Phone 8 share a common core.

Core System and Mobile Core only represent the alignment of Windows and Windows Phone where the two operating systems are running exactly the same code. There are numerous other areas where APIs and behavior are shared, albeit with slightly different implementations to account for the different environments. For example, the location API in Windows Phone automatically incorporates crowd-sourced data about the position of cell towers and Wi-Fi access points to improve the accuracy of location readings, an optimization which is not part of the Windows 8 location framework.

## Windows Runtime

For consumers, the most radical change in Windows 8 is the new UI. For developers, it is the new programming model and API set, collectively known as the Windows Runtime. Although Microsoft has delivered a variety of new developer technologies on top of Windows over the years (most notably .NET), the core Windows programming model has not changed significantly in decades. The Windows Runtime represents not just a set of new features and capabilities, but a fundamentally different way of building Windows apps and components.

The Windows Runtime platform is based on a version of the Component Object Model (COM) augmented by detailed metadata describing each component. This metadata makes it simple for Windows Runtime methods and types to be “projected” into the various programming environments built on top of it. In Windows Phone, there are two such environments: a CoreCLR-based version of .NET (C# or Visual Basic), and pure native code (C/C++). We will discuss the Windows Runtime throughout the book, covering both consumption of Windows Runtime APIs from your apps as well as creation of new Windows Runtime components.



**Note** Even though the core architecture of the Windows Runtime and many of the APIs are the same for Windows and Windows Phone, the two platforms offer different versions of the API framework which sits on top of it. For instance, Windows Phone does not implement the *Windows.System.RemoteDesktop* class, but does add some phone-specific namespaces such as *Windows.Phone.Networking.Voip*. The term Windows Phone Runtime is sometimes used in documentation and in Visual Studio project templates to highlight this difference. However, since the core technology is the same and the differences are obvious in context, we will use the term Windows Runtime throughout the book.

## Building and delivering apps

---

Now that you understand the fundamentals of Windows Phone, it’s time to start looking at how you can build and deliver apps that run on it.

### Developer tools

Everything you need to get started building Windows Phone 8 apps is available in the Windows Phone 8 SDK, which is available as a free download from the Windows Phone Dev Center at <http://dev.windowsphone.com>. In particular, the Windows Phone 8 SDK includes the following:

- Microsoft Visual Studio 2012 Express for Windows Phone
- Microsoft Blend 2012 Express for Windows Phone
- The Windows Phone device emulator
- Project templates, reference assemblies (for managed code development), and headers/libraries (for native code development)

As with previous versions of the Windows Phone SDK, Visual Studio Express and Blend Express can be installed on top of full versions of Visual Studio and Blend, seamlessly merging all of the phone-specific tools and content directly into your existing tools. Throughout the book, we will refer to Visual Studio Express 2012 for Windows Phone as the primary development environment for Windows Phone 8, but everything we describe will work just as well with any other version of Visual Studio as soon as you have the Windows Phone 8 SDK installed.



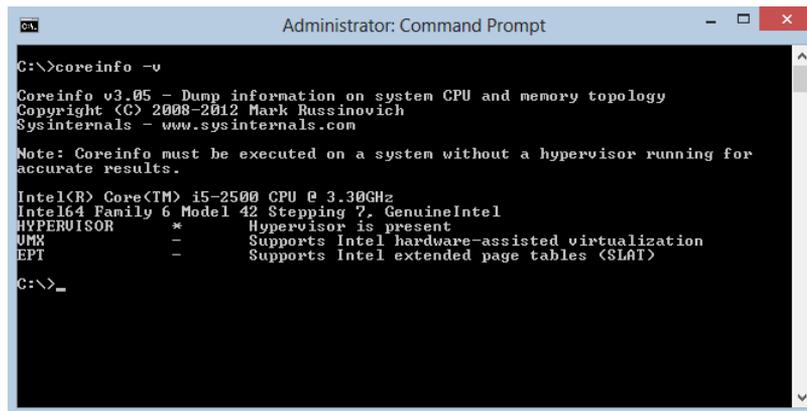
**Note** Visual Studio 2012, including Visual Studio 2012 Express for Windows Phone, can only be installed on Windows 8.

## Windows Phone emulator system requirements

The Windows Phone 8 SDK includes a new version of the Windows Phone emulator for testing apps directly on your desktop. The new emulator is built on the latest version of Microsoft Hyper-V, which requires a 64-bit CPU that includes Second Level Address Translation (SLAT), a memory virtualization technology included in most modern CPUs from Intel and AMD.

To check if your CPU supports SLAT, do the following:

1. Download the Coreinfo tool from <http://technet.microsoft.com/en-us/sysinternals/cc835722>.
2. Open a command prompt as an administrator. From the Start menu, type **cmd** to find the command prompt, right-click it, and then choose Run As Administrator.
3. Navigate to the location where you downloaded Coreinfo and run **CoreInfo -v**.
4. Look for a row labeled EPT (for Intel CPUs) or NP (for AMD). If you see an asterisk, as shown in Figure 1-7, you're all set. If you see a dash, your CPU does not support SLAT and will not be capable of running the new Windows Phone emulator. Note that if you have already activated Hyper-V on your computer, you will see an asterisk in the HYPERVISOR row and dashes elsewhere. In this case, you can safely ignore the dashes because your computer is already prepared to run the Windows Phone Emulator.



```
Administrator: Command Prompt
C:\>coreinfo -v
Coreinfo v3.05 - Dump information on system CPU and memory topology
Copyright (C) 2008-2012 Mark Russinovich
Sysinternals - www.sysinternals.com

Note: Coreinfo must be executed on a system without a hypervisor running for
accurate results.

Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz
Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
HYPERVISOR      *      Hypervisor is present
VMX             -      Supports Intel hardware-assisted virtualization
EPT             -      Supports Intel extended page tables (SLAT)

C:\>_
```

**FIGURE 1-7** Use the free Coreinfo tool to determine if your computer can run the new Windows Phone emulator.



**Note** SLAT is required only to run the Windows Phone emulator. You can still build Windows Phone 8 apps on a non-SLAT computer; you will simply need to deploy and test them on a physical device.

## Building for Windows Phone 7.x and 8.x

Because Windows Phone 8 requires new hardware, it will take some time for the installed base of Windows Phone 8 devices to surpass that of the existing Windows Phone 7.x phones. During that time, you will likely want to deliver two versions of your app, one for Windows Phone 7.x and one for Windows Phone 8.0. The Windows Phone 8 developer tools have full support for this approach.

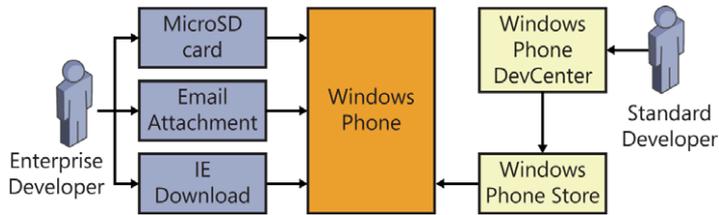
In Visual Studio 2012 Express for Windows Phone, you can create new projects for Windows Phone 7.1 and Windows Phone 8.0, and each will be deployed to the appropriate emulator image for its target platform. You can also run your Windows Phone 7.1 apps on the Windows Phone 8 emulator to ensure that it behaves as expected—even though Windows Phone 8 is backward-compatible with Windows Phone 7.0 and 7.1 apps, it is always worth verifying that there aren't any nuances in the platform behavior for which you might want to account.

### Lighting up a Windows Phone 7.1 app with new tiles

To truly take advantage of the new platform features in Windows Phone 8, you must build a version of your app which explicitly targets Windows Phone 8.0. Because there is some additional overhead to creating and managing a separate XAP for version 8.0, Windows Phone 8 allows Windows Phone 7.1 apps to create and manage the new Live tile templates available in the latest release. This approach is based on reflection and is described in detail in Chapter 13, "Porting to Windows Phone 8 and multitargeting."

## App delivery

Windows Phone 7.x offered a single, broad mechanism for distributing apps: the Windows Phone Store (previously, the Windows Phone Application Marketplace). In Windows Phone 8, the Windows Phone Store will continue to be the primary source of apps for most customers. However, the distribution options have been expanded to include additional channels for distributing enterprise apps—enterprise customers will be able to deliver apps to their employees via the Internet, intranet, email, or by loading them on a microSD card and inserting the card into the phone. The options for app deployment in Windows Phone 8 are depicted in Figure 1-8.



**FIGURE 1-8** Windows Phone 8 adds multiple enterprise deployment options.

If you're familiar with any flavor of .NET technology, you know that building a project doesn't generally convert your code into something that's directly executable by a CPU. Rather, it is converted into Microsoft Intermediate Language (MSIL), a platform-independent instruction set, and packaged into a dynamic-link library (DLL). In the case of Windows Phone, these DLLs are then added to your app package for delivery to the phone, where it remains until the user launches the app. At that point, the just-in-time (JIT) compiler turns those DLLs into native instructions targeting the appropriate platform—ARM for physical devices and x86 for the Windows Phone emulator.

In Windows Phone 8, this process changes, such that all apps are precompiled as part of the Windows Phone Store submission process. This means that when a user downloads an app from the Windows Phone Store, the app package already contains code that is compiled for ARM. Because no "JITing" is required when the app is starting up or running, users should experience faster app load times and improved runtime performance.



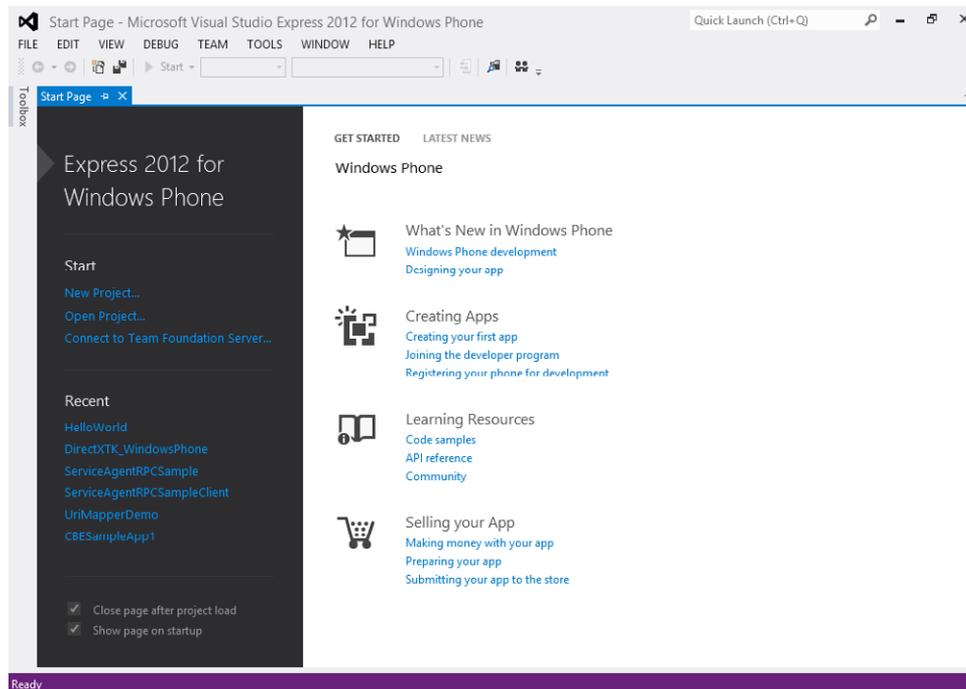
**Note** Existing Windows Phone 7.1 apps are automatically precompiled in the Windows Phone Store. No action is required from the developers of those apps.

## Getting started with "Hello World"

By now, you are well versed in the fundamentals of Windows Phone. Go ahead and file all of that knowledge away, because it's time to get into some code. Those of you who are seasoned Windows Phone developers will no doubt be tempted to skip this section, but you might want to at least ensure that your installation of the Windows Phone Developer Tools is working properly before diving into more advanced topics. In particular, you should try to launch a project in the Windows Phone emulator to ensure that Hyper-V is fully enabled and then navigate to a webpage in Internet Explorer to verify that networking is properly set up.

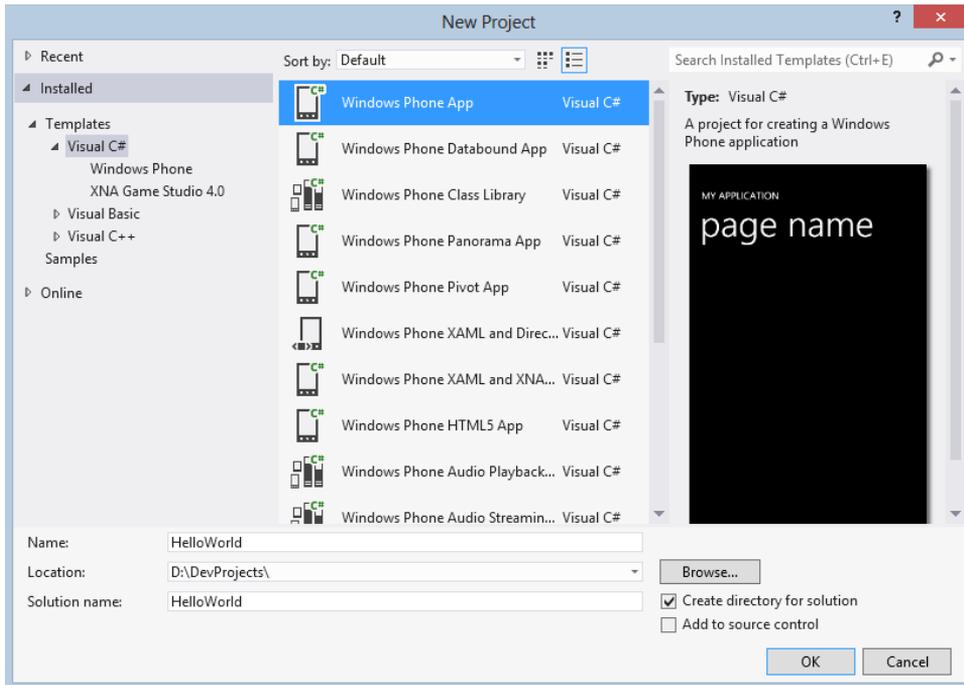
# Creating a project

After you've installed the Windows Phone SDK from the Dev Center, begin by starting Visual Studio. The first screen you see is the Visual Studio Start Page, as demonstrated in Figure 1-9.



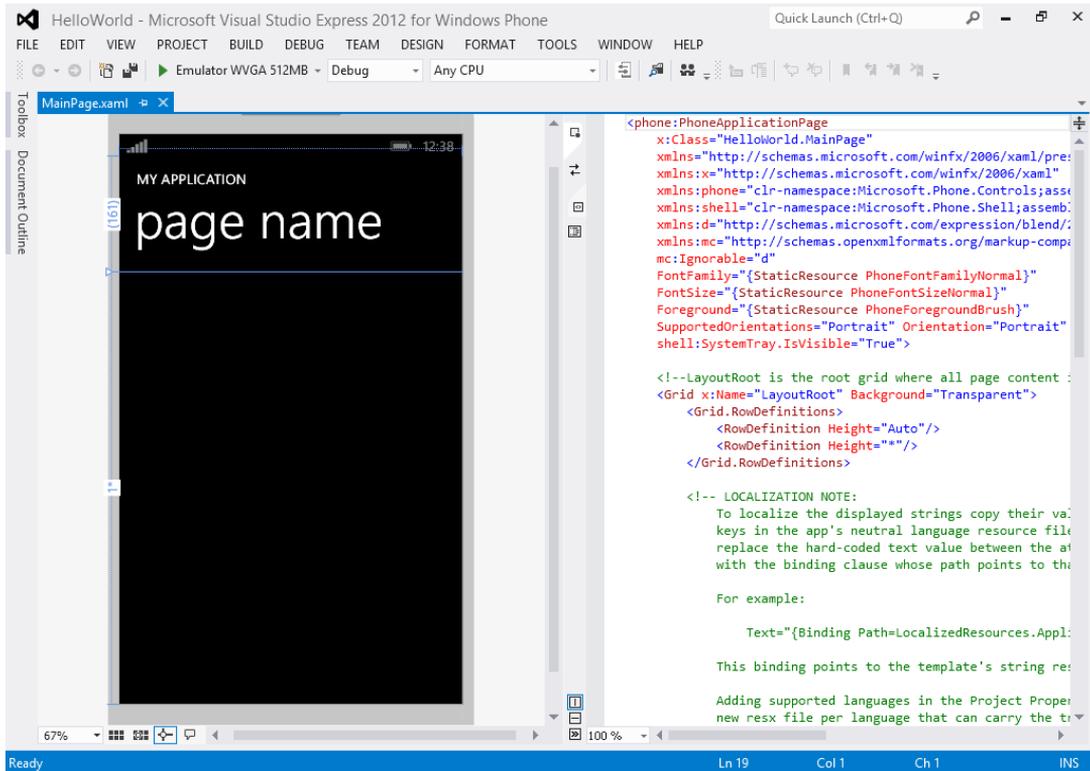
**FIGURE 1-9** The first screen you see upon starting Visual Studio is the Start Page, which offers a quick way to begin a new project.

On the left side of the Start Page, in the navigation pane, click **New Project**. This opens the New Project dialog box in which you can choose the type of project that you want to create and the language in which you want to write it. XAML apps written in C# are the most common type on Windows Phone, so we will start there. Under **Templates**, click **Visual C#**, choose **Windows Phone App**, and then name it **HelloWorld**, as shown in Figure 1-10.



**FIGURE 1-10** The New Project dialog box offers a number of templates for creating new apps, class libraries, background agents, and more. To get started, create a simple Windows Phone App in C#.

If your project was created successfully, you should be looking at a screen that resembles Figure 1-11, with *MainPage.xaml* already opened for you.



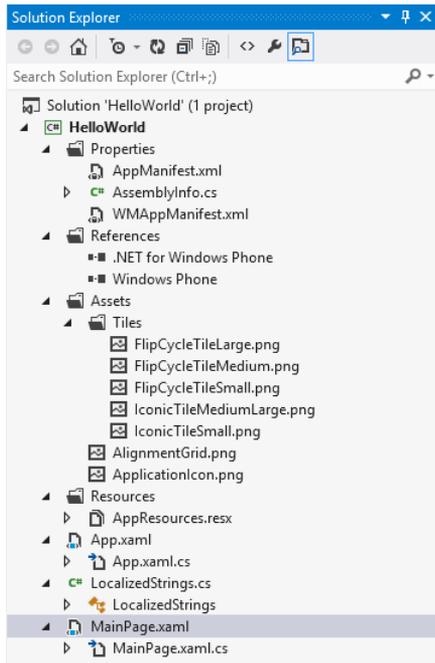
**FIGURE 1-11** By default, for a Windows Phone project in C#, Visual Studio starts on *MainPage.xaml*.

## Understanding the project structure

*MainPage.xaml* is one of a number of folders and files included in the default Windows Phone project template. Some of these have special meaning which might not be obvious at first glance, so it's worth taking a quick tour of the standard project structure while you're building your "Hello World" app. Figure 1-12 shows an expanded view of Solution Explorer for "Hello World."



**Note** The project structure and the list of default files shown in Figure 1-12 is representative of a managed XAML app. The structure of Direct3D apps is discussed in Chapter 25, "Games and Direct3D."



**FIGURE 1-12** Windows Phone project templates include a number of special files to get you started.

The first important file is *WMAAppManifest.xml*, the app manifest. The app manifest contains all of the information that the OS needs to know about the app to surface it properly on the phone. Some elements of the manifest (for example, hardware requirements) are also used during the Windows Phone Store submission process. The manifest includes (among other things) the following:

- The app name
- A reference to its app list icon and default Start tile
- Its supported resolutions (new in Windows Phone 8)
- The list of security capabilities it requires, such as location and photos, and any hardware requirements, such as NFC or a front-facing camera
- Any extensibility points for which the app is registering—for example, as an entry in the Photos share picker

In Visual Studio Express 2012 for Windows Phone, many of these manifest attributes are now configurable through a simple GUI. However, some features, such as registering for extensibility points, still require direct editing of the underlying XML file.



**Tip** By default, Visual Studio always displays the GUI tool when you double-click *WMAppManifest.xml*. To configure additional settings with the raw XML editor, in Solution Explorer, right-click the app manifest file. In the shortcut menu that opens, point to Open With and then click XML (Text) Editor. To return to the GUI tool, double-click the file again.

The Assets folder is provided as a location to include the core images that your app should provide. At the root of the Assets folder is a file called *ApplicationIcon.png*. This is a default icon which is shown for your app in the app list. The Tiles subfolder is prepopulated with a handful of icons for use in the FlipCycle and Iconic tile templates, which are discussed in detail in Chapter 14, “Tiles and notifications”. All of these files are placeholders intended to show you which images need to be provided. You can (and should) change them to something representative of your app before submitting it to the Windows Phone Store or distributing it to others.

Together, *Resources\AppResources.resx* and *LocalizedStrings.cs* provide the initial framework for developing a fully localized app. Localization is beyond the scope of this book, but it is well documented on MSDN. See [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff637522\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff637522(v=vs.105).aspx) for details on building a fully localized Windows Phone app.

*App.xaml* provides a convenient location to store resources that you intend to use throughout your app, such as UI styles. Its code counterpart, *App.xaml.cs*, contains critical startup code that you should generally not modify and some empty handlers for the core app lifetime events—Launching, Activated, Deactivated, and Closing. If you want to take any action when your app is opened, closed, paused, or resumed, you will need to fill these in. This is discussed in more detail in Chapter 2, “App model and navigation.”

*MainPage.xaml* is the default starting point for your app, which we will return to momentarily to make some changes for our “Hello World” app. You can think of pages in Windows Phone as being equivalent to webpages. They contain both the definition of the UI that will be displayed to the user as well as the bridging code between that UI and the rest of the app’s functionality. The role of pages in the Windows Phone navigation model is explored in depth in Chapter 2.



**Tip** Remember that the project templates are just a starting point for your app; you don’t need to be locked in to their structure or content. For instance, if you’re following a Model-View-ViewModel (MVVM) pattern, you might want to consolidate all of your views in a single subfolder. If so, don’t hesitate to move *MainPage.xaml* to that folder or create a new page to act as your main page in that location. Just remember to update the Navigation Page setting in your manifest so that the system knows where to start your app.

## Adding a splash screen

New Windows Phone 7.1 projects also include a file named *SplashScreenImage.jpg*, which, as the name implies, is rendered as a splash screen while the app is loading. Given the improvements in app launch time in Windows Phone 8, it is assumed that most apps will not need a splash screen, so the file has been removed from the default project template. If you believe your app could benefit from a splash screen, simply add a file named *SplashScreenImage.jpg* to the root of the project, with its Build Action set to Content.

## Greeting the world from Windows Phone

Now that you understand what all the files in the default project mean, return to *MainPage.xaml*, which has been waiting patiently for you to bring it to life. By default, Visual Studio displays pages in a split view, with the Visual Studio designer on the left and the XAML markup that defines the UI on the right. You can make changes to your page by manipulating controls in the visual designer or by directly editing the XAML.

Start by using the designer to make changes to the default text that the project template has included at the top of the page. Double-click the words MY APPLICATION and change the entry to **HELLO, WORLD**. Likewise, double-click "page name" and change it to **welcome**.

Now, redirect your attention to the right side of the screen, where the XAML markup for your *MainPage* is shown. You will probably be able to spot where the changes you just made were reflected in the underlying XAML. The `<StackPanel>` element with the name *TitlePanel* should now look like this:

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
  <TextBlock Text="HELLO, WORLD" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
  <TextBlock Text="welcome" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

Directly below the *TitlePanel*, you should find a *Grid* element called *ContentPanel*. Replace this element with the following XAML:

```
<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <TextBlock
    x:Name="helloTextBlock"
    Text="Hello from Windows Phone 8!"
    Foreground="{StaticResource PhoneAccentBrush}"
    Grid.Row="0"
    HorizontalAlignment="Center"/>
  <Button
    x:Name="goodbyeButton"
    Content="Say goodbye!"
    Grid.Row="1" Click="goodbyeButton_Click"/>
</StackPanel>
```

This markup creates a simple *StackPanel* in your app, which is then filled with a *TextBlock* and a *Button*. The *TextBlock* contains the critical greeting, whereas the *Button* suggests that the meeting might not last very long. You use a *StackPanel* in this case because it is a simple and efficient way of displaying a set of visual elements in a horizontal or vertical arrangement.

As mentioned earlier, a page contains both the markup describing how the UI looks and the connective code that bridges the gap between the UI and the rest of the app's logic. In this case, the button acts as your first glimpse into that code. Double-click the *Button* in the Visual Studio designer. This opens *MainPage.xaml.cs*, which is known as a *code-behind* file because, as its name implies, it contains the code behind a given page. A code-behind file is created automatically for every page you create in a managed Windows Phone project.

You will notice that Visual Studio has not only opened the code-behind file, but it has taken the liberty of creating a click event handler (named *goodbyeButton\_Click*) for the button that you added to your page. You will use this event handler to add some code that makes your app actually do something.



**Note** It might seem odd to be handling “click” events in an app built for an exclusively touch-based platform. The reason is that the managed UI framework for Windows Phone is primarily based on Microsoft Silverlight, which was initially built as a web browser plugin. Because a “tap” in a phone app is semantically equivalent to a click on a web page, there was no compelling reason to rename the event.

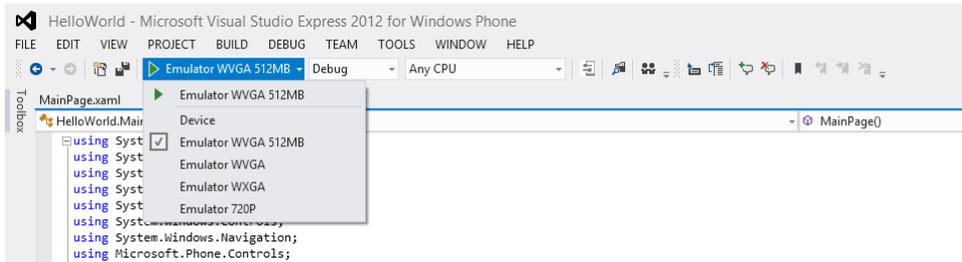
Add the following code to the click event handler:

```
helloTextBlock.Visibility = System.Windows.Visibility.Collapsed;  
goodbyeButton.IsEnabled = false;
```

As you can probably discern, this code does two things: it makes your “Hello from Windows Phone 8!” text disappear, and it disables your button. To be sure, it's time to run the app.

Cross your fingers and press F5.

Within a few seconds, you should see the Windows Phone Emulator starting up. By default, Windows Phone 8 projects target the WVGA 512MB emulator, meaning a virtualized version of a Windows Phone 8 device with a WVGA (800x480) screen and 512 MB of memory. You can easily change this on a drop-down menu on the Visual Studio toolbar, as shown in Figure 1-13.



**FIGURE 1-13** By default, Visual Studio deploys Windows Phone 8 projects to a WVGA 512MB emulator. You can change this target through a drop-down in the toolbar.

If all has gone according to plan, you should see your app running in the emulator, with your *MainPage* displayed and ready. Go ahead and click the Say Goodbye! button and you should see your “Hello from Windows Phone 8!” text disappear and your button become dimmed, indicating that it’s been disabled.

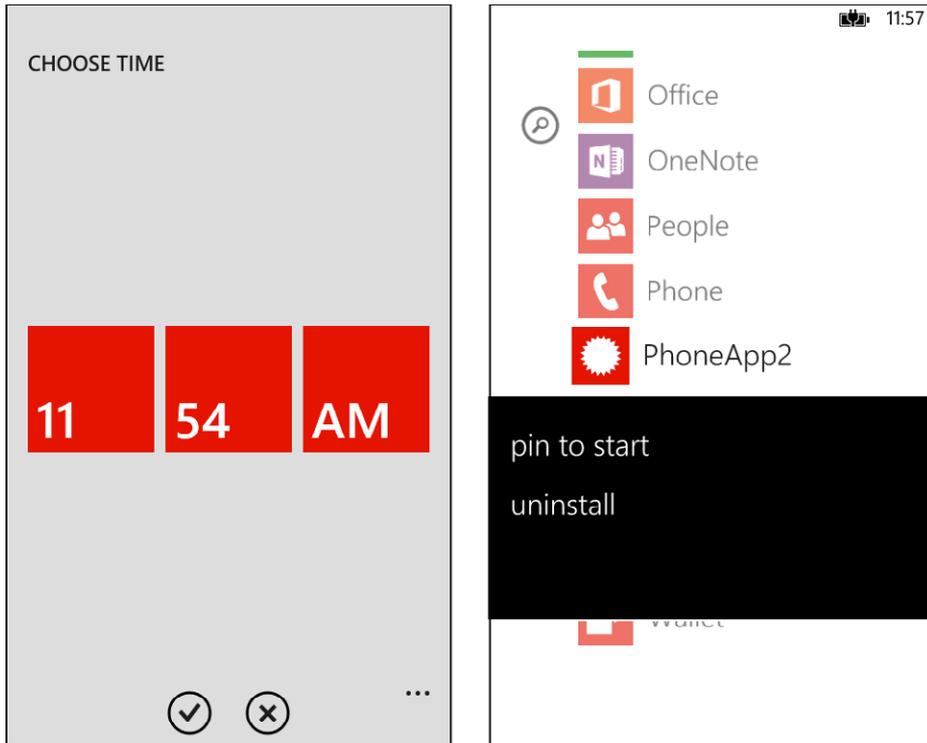
## Deploying to a Windows Phone device

The Windows Phone Emulator is sufficient for most of the app development you will do, especially while learning about the platform, as you’ll be doing throughout this book. After you begin building a real app that you intend to submit to the Windows Phone Store, however, you will want to deploy it to a real device.

The first step in deploying to a device is registering as a developer in the Windows Phone Dev Center at <http://dev.windowsphone.com>. After you’re registered, you can “unlock” your device for deploying apps by using the Windows Developer Registration Tool, which is included in the Windows Phone SDK. Simply connect your device to your computer via USB, run the tool, and then enter your Microsoft Account details. Within a few seconds, your device will be ready for deployment directly from Visual Studio.

## The Windows Phone Toolkit

Windows Phone provides most of the core platform controls, such as *Button*, *TextBox*, *Pivot*, and *Panorama* directly in the SDK. However, some controls and UI behaviors that are commonly found in the built-in apps are not available in the SDK. For example, the SDK does not include controls that mimic the built-in time picker or context menu, which are shown in Figure 1-14.



**FIGURE 1-14** Many standard Windows Phone controls are not included in the SDK.

Instead, many of these controls are shipped separately from the main SDK in the Windows Phone Toolkit, which is built and maintained by the same team that creates the SDK controls. The advantage of this approach is that the toolkit can be updated more frequently than the SDK can, making it possible to fix bugs, improve performance, and add new controls on a regular basis. The toolkit is available via NuGet, a package manager included by default in Visual Studio 2012, which enables you to add libraries directly to your project. To install the Windows Phone Toolkit through NuGet, select Tools | Library Package Manager | Package Manager Console in Visual Studio. When the console has loaded, enter the following command at the prompt:

PM > **Install-Package WPToolkit**

Within a few seconds, the toolkit will be downloaded and added to your project. To begin using the toolkit controls and behaviors within your XAML, you must add a prefix for the toolkit's namespace to your page as shown:

```

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

```

As a simple example of how the toolkit can add some visual polish to your app, we will add the tilt effect to the button in our Hello World solution. The tilt effect, found throughout the Windows Phone user experience, enables clickable controls to tilt in response to the user's touch input. To enable the tilt effect for a particular element in XAML, you simply need to set the *TiltEffect.IsTiltEffectEnabled* attached property to *true* as shown:

```

<Button
  x:Name="goodbyeButton"
  Content="Say goodbye!"
  Grid.Row="1" Click="Button_Click"
  toolkit:TiltEffect.IsTiltEnabled="True"
/>

```



**Tip** In most cases, you should enable the tilt effect across all clickable elements in your app. To do that, simply attach it to your app's *RootFrame* in the *App.xaml.cs* file:

```
TiltEffect.SetIsTiltEnabled(RootFrame, true);
```

That's it! If you've made it this far successfully, you should have everything set up correctly and you should be ready for the more detailed topics in the chapters ahead.

## Summary

In this chapter, you learned about many of the principles driving the development of Windows Phone, including the distinctive UI, the architectural convergence with Windows, and the importance of developers and apps. These principles will act as the foundation as you proceed through the remainder of the book and delve into the details of specific features.

You were also introduced to the Windows Phone developer tools and SDK, so with "Hello World" in the rearview mirror, it's time to move on.

# Phone and media services

For app developers, a key feature of a smartphone—beyond the ability to make and receive cellular calls, of course—is the degree to which you can integrate your app with the built-in phone and media services. Windows Phone provides a set of Launchers and Choosers, which is the way that the app platform repurposes built-in apps, such as the browser, search, calendar, email, and so on. You can take advantage of these features within your own apps.

The platform also provides extensive support for media services, including audio and video playback. You can choose from several different sets of media-related application programming interfaces (APIs), depending on the specific requirements of your app. If you have a simple requirement for media playback, you can use the *MediaPlayerLauncher*. If you need more flexibility, the *MediaElement* type might suit your needs better. If you need more fine-grained control over the media file content, you can use the *MediaStreamSource* API, and so on. In this chapter, we examine the various levels of app support and your choices for integrating with—and extending—standard system features, including built-in phone apps, audio and video services, and search. This is an area where relatively little has changed between Windows Phone 7 and Windows Phone 8, so most of the programming techniques described in this chapter work on both versions of the platform.

## Launchers and Choosers

---

Both Launchers and Choosers are API mechanisms for invoking existing apps and services on the phone. In the API, they're all called "tasks." The difference between Launchers and Choosers is that Launchers start a feature but don't return a value, whereas Choosers launch a feature and *do* return a value. Table 5-1 summarizes these tasks. The phone has a range of very useful built-in apps, exposed for programmatic integration via a very simple set of APIs.

Invoking a Launcher or Chooser causes your app (the invoking app) to be deactivated. Some Choosers run in your app session, so if you press and hold the back key, you don't see your app as a separate instance that you can activate; rather, you see the Chooser screenshot listed for your app. As you would expect, the inter-app navigation behavior is consistent with all other inter-app navigations. Users can return to the original app upon completing the task inside the Launcher/Chooser, or they can use the Back key. If the Launcher/Chooser has multiple pages, the Back key will navigate the user through the previous pages and, finally, back to the calling app. In the same manner, if the user navigates forward through multiple apps, this can result in the original calling app falling off the backstack, as normal. Also, the Chooser is auto-dismissed if the user forward navigates away from it. Table 5-1 lists the full set of available Launchers and Choosers.

**TABLE 5-1** Launchers and Choosers

Type	Task	Description	New in 8
Launchers	<i>BingMapsDirectionsTask</i>	Launches the Maps app, specifying a starting and/or ending location, for which driving or walking directions are displayed (identical to the <i>MapsDirectionsTask</i> ).	
	<i>BingMapsTask</i>	Launches the Maps app centered at the specified or current location (identical to the <i>MapsTask</i> ).	
	<i>ConnectionSettingsTask</i>	Launches a settings dialog with which the user can change the device's connection settings.	
	<i>EmailComposeTask</i>	Composes a new email.	
	<i>MapDownloaderTask</i>	Provides a mechanism for the user to download maps for offline use.	Y
	<i>MapUpdaterTask</i>	Provides a mechanism for the user to update map data he has previously downloaded for offline use.	Y
	<i>MapsDirectionsTask</i>	Launches the Maps app, specifying a starting and/or ending location, for which driving or walking directions are displayed (identical to <i>BingMapsDirectionsTask</i> ).	Y
	<i>MapsTask</i>	Launches the Maps app centered at the specified or current location (identical to <i>BingMapsTask</i> ).	Y
	<i>MarketplaceDetailTask</i>	Launches the Windows Phone Store and displays the details for a specific app.	
	<i>MarketplaceHubTask</i>	Launches the Windows Phone Store and searches for a particular type of content.	
	<i>MarketplaceReviewTask</i>	Displays the Windows Phone Store review page for the current app.	
	<i>MarketplaceSearchTask</i>	Launches the Windows Phone Store and displays the search results from the specified search terms.	
	<i>MediaPlayerLauncher</i>	Launches Media Player, and plays the specified media file.	
	<i>PhoneCallTask</i>	Initiates a phone call to a specified number.	
	<i>SaveAppointmentTask</i>	Provides a mechanism for the user to save an appointment from your app.	Y
	<i>SearchTask</i>	Launches Microsoft Bing Search with a specified search term.	
	<i>ShareLinkTask</i>	Launches a dialog with which the user can share a link on the social networks of her choice. If the user does not have any social networks set up, the Launcher silently fails.	
	<i>ShareMediaTask</i>	Provides a mechanism for your app to share a media item with one of the media-sharing apps on the phone.	Y
<i>ShareStatusTask</i>	Launches a dialog with which the user can share a status message on the social networks of her choice. If the user does not have any social networks set up, the Launcher silently fails.		
<i>SmsComposeTask</i>	Composes a new text message.		
<i>WebBrowserTask</i>	Launches Microsoft Internet Explorer and browses to a specific URI.		

Type	Task	Description	New in 8
Choosers	<i>AddWalletItemTask</i>	Launches the Wallet app and allows the user to add the supplied item to his wallet.	Y
	<i>AddressChooserTask</i>	Launches the Contacts app with which the user can find an address.	
	<i>CameraCaptureTask</i>	Opens the Camera app to take a photo.	
	<i>EmailAddressChooserTask</i>	Provides a mechanism for the user to select an email address from his Contacts List.	
	<i>GameInviteTask</i>	Shows the game invite screen with which the user can invite players to a multiplayer game session.	
	<i>PhoneNumberChooserTask</i>	Provides a mechanism for the user to select a phone number from his Contacts List.	
	<i>PhotoChooserTask</i>	Provides a mechanism for the user to select an image from his Picture Gallery or take a photo.	
	<i>SaveContactTask</i>	Launches the Contacts app with which the user can save a contact.	
	<i>SaveEmailAddressTask</i>	Saves an email address to an existing or new contact.	
	<i>SavePhoneNumberTask</i>	Saves a phone number to an existing or new contact.	
	<i>SaveRingtoneTask</i>	Launches the Ringtones app with which the user can save a ringtone from your app to the system ringtones list.	

Of the new tasks introduced in Windows Phone 8, maps are discussed in Chapter 18, “Location and maps,” appointments are discussed in Chapter 15, “Contacts and calendar,” and wallet is discussed in Chapter 20, “The Wallet.” However, the general pattern for invoking Launchers and Choosers is consistent across all tasks. The *LaunchersAndChoosers* solution in the sample code demonstrates this pattern. Buttons are available to invoke a *PhoneCallTask*, *WebBrowserTask*, *SearchTask*, and *EmailAddress ChooserTask*, as shown in Figure 5-1.

In general, the app code for invoking Launchers and Choosers is very simple. The app platform provides easy-to-use wrappers for all of the app-accessible system tasks on the device. The basic steps for using a Launcher are as follows:

1. Create an instance of the type that represents the specific Launcher feature that you want to use.
2. Set properties on the object as appropriate.
3. Invoke the *Show* method.



**FIGURE 5-1** Many standard tasks are exposed programmatically as Launchers and Choosers.

In this example, when the user taps the Phone button, the app instantiates a *PhoneCallTask* object and calls *Show*, which prompts the user to confirm the outgoing call. This feature is protected by the *ID\_CAP\_PHONEDIALER* capability, so you need to update your app manifest to include this; otherwise, the attempt to invoke *PhoneCallTask.Show* will throw a security exception.

```
PhoneCallTask phone = new PhoneCallTask();  
phone.PhoneNumber = phoneNumber.Text;  
phone.Show();
```

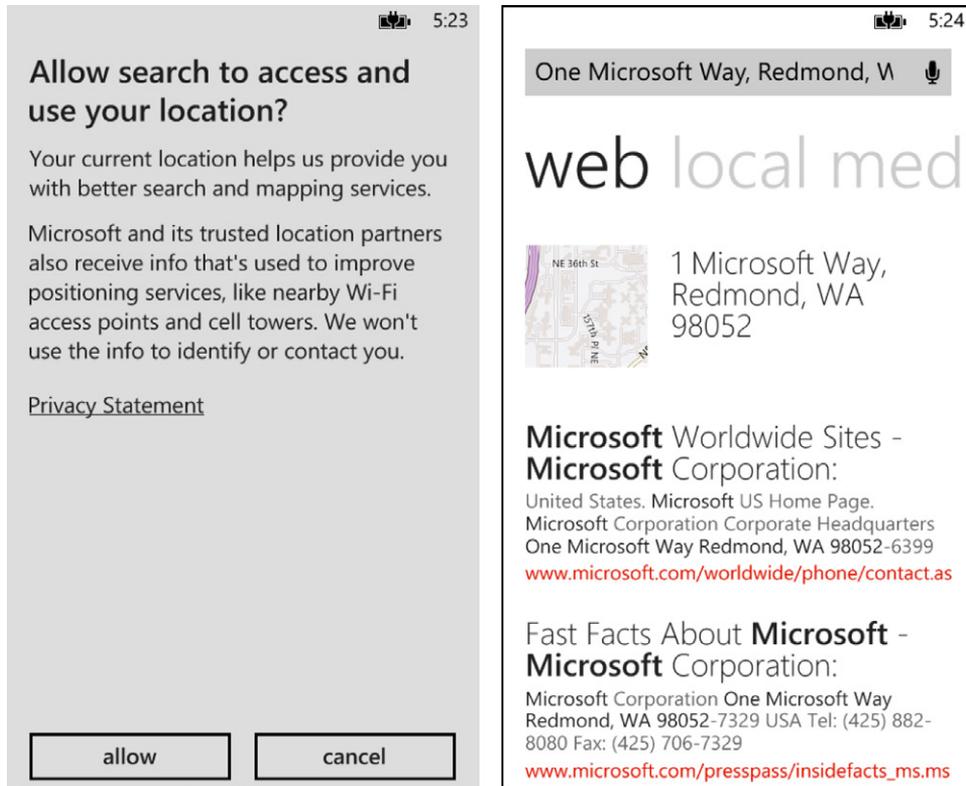
The Search button invokes *SearchTask.Show*; it is equally simple to use, following the exact same pattern.

```
SearchTask search = new SearchTask();  
search.SearchQuery = searchText.Text;  
search.Show();
```

The first time the user selects the search feature, he gets a prompt from the system, as shown in Figure 5-2 (left). This is because the search feature can make use of the user's location, and the user must be given the option to allow access to this information. If and when the user proceeds with the search, the task displays a scrolling list that contains the Bing search results for the specified term, as illustrated in Figure 5-2 (right).



**Note** This happens only once on a real device, but it will happen on the emulator every time you restart it.



**FIGURE 5-2** Invoking a search task for the first time triggers a permission request to the user.

Using the *WebBrowserTask* follows the same pattern. Windows Phone 7 developers might be accustomed to using the *URL* string property of this task, but this is now obsolete, and you should instead use the *Uri* property (of type *Uri*, and named *Uri*).

```
WebBrowserTask browser = new WebBrowserTask();  
browser.Uri = new Uri("http://" + browseText.Text);  
browser.Show();
```

Using the *EmailAddressChooserTask* is only marginally more involved. So far, we've shown how to use Launchers, but the *EmailAddressChooserTask* is a Chooser, which means it returns a value to the calling app. The basic steps for using a Chooser are as follows:

1. Create an instance of the type that represents the specific Chooser feature that you want to use. It is common to declare this as a field in your page class.
2. Hook up the *Completed* event on the object (or provide an inline delegate or lambda), which will be invoked when the Chooser task completes.
3. Set properties on the object as appropriate.
4. Invoke the *Show* method.
5. In your *Completed* event handler or delegate, process the return value from the Chooser.

The code snippet that follows illustrates this pattern. In this example, the email address that the user has selected from the contacts list offered by the Chooser is then pasted into the corresponding *TextBox* back in the sample app. It is important to note that you must declare the Chooser object as a class field in the page where it is used. You must also call the Chooser constructor to set up the field—and also hook up the *Completed* event—within the page constructor. The reason for this is to ensure that your app will be called back correctly after it has been deactivated and then reactivated for situations in which it was tombstoned and then rehydrated on return from the Chooser.

```
private EmailAddressChooserTask emailAddress = new EmailAddressChooserTask();

public MainPage()
{
    InitializeComponent();
    emailAddress.Completed += emailAddress_Completed;
}

private void emailButton_Click(object sender, RoutedEventArgs e)
{
    emailAddress.Show();
}

private void emailAddress_Completed(object sender, EmailResult e)
{
    if (e.TaskResult == TaskResult.OK)
    {
        emailText.Text = e.Email;
    }
}
```

## Search extensibility

---

Another way that your app can communicate with standard phone services is by extending the Bing search experience with custom behavior, integrating your app seamlessly with the search results. There are two ways by which you can extend the Bing search behavior in your apps: App Connect

and App Instant Answer. With both features, you can set up your app so that it shows up in the Bing search results when the user taps the hardware Search button. Table 5-2 summarizes the differences.

**TABLE 5-2** Bing search extensibility

Requirement	App Connect	App Instant Answer
<i>WAppManifest.xml</i>	Requires Extensions entries for each Bing category that you want to extend.	No specific changes required.
<i>Extras.xml</i>	Required. Specifies captions to be used in the search results apps pivot item.	Not used.
<i>UriMapper</i>	Recommended. Allows you to re-route to a specific page on app startup.	Not required.
Target page	You can re-route to multiple different pages, depending on the search item, if you want.	No option. Your app is launched as normal, with its default startup page.
Query string	You should parse the incoming query string for categories and item names for which you want to provide extensions.	You should parse the incoming query string for the <i>bing_query</i> value.
Search connection	Bing includes your app in the search results when the user's search matches the categories for which you registered extensions.	Bing includes your app in the search results, based on whether it thinks the query is relevant to your app.

In both cases, your app is launched with a particular query string, and you are responsible for parsing that query string to get the search context. It is then up to your app to decide what behavior to execute, based on this search context. Both approaches are discussed in more detail in the following sections.

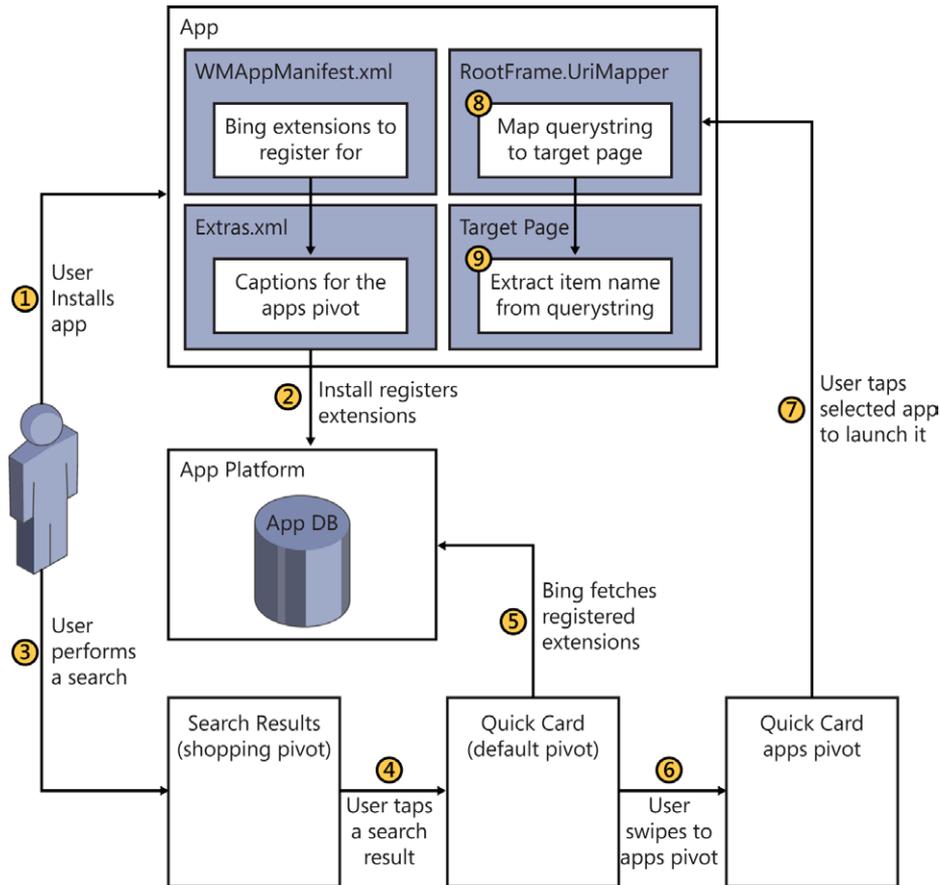
## App Connect

The App Connect approach is the more complex of the two extensibility models. You register your app in a way that gives you more fine-grained control over the criteria that Bing will use to identify it as a suitable extension. You should only register your app for the search categories—or search “extensions”—for which you believe your app has relevance.



**Note** It is important to choose the extensions carefully and to avoid spamming the system by registering for unrelated extensions. Apps that register excessive unrelated extensions will be removed from the Windows Phone Store.

If and when the user chooses to start your app from the search results list, your app is given a richer query string with which you can fine-tune its subsequent behavior. The overall model for App Connect is illustrated in Figure 5-3. In summary, the user initiates a Bing search and then taps one of the items in the search results. This navigates to a system-provided Quick Card, which is a pivot page that offers an “about” pivot that provides basic information, a “reviews” pivot, and an “apps” pivot. Your app can be listed in the apps pivot.



**FIGURE 5-3** The App Connect extensibility model.

The following steps walk through how to create an App Connect search extension. A completed example is in the *SimplestAppConnect* solution in the sample code. Figure 5-4 (left) shows the search results page for a search on the string "coffee." Figure 5-4 (center) shows the Quick Card for one of the selected items from the search results, and Figure 5-4 (right) shows the apps pivot for that Quick Card, in which the sample app's title string and caption strings are displayed alongside the app's icon.

First, create a new Windows Phone app project. Add a second page to the project, named *MyTargetPage*. This will be the target page to which to navigate when the app is launched via App Connect. In the XAML for this page, add a simple *TextBlock*; the app will eventually set the text for this dynamically using the item information in the Bing search results.

```
<TextBlock x:Name="Target" TextWrapping="Wrap" Margin="{StaticResource PhoneHorizontalMargin}"/>
```

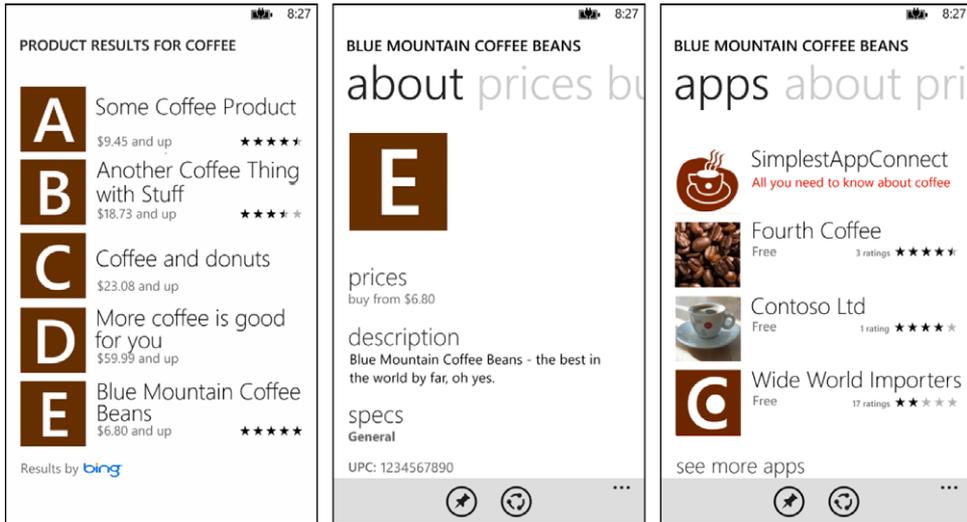


FIGURE 5-4 Search results, Quick Card, and apps list for a search on the term “coffee.”

Then, add an *Extensions* section to your *WMAAppManifest.xml*, within the *App* element, after the *Tokens* section. The *Extension* entry must specify one of the defined extension identifiers, as listed in the Search Registration and Launch Reference for Windows Phone, which you can find at [http://msdn.microsoft.com/en-us/library/hh202958\(VS.92\).aspx](http://msdn.microsoft.com/en-us/library/hh202958(VS.92).aspx). In this example, the app registers for just one extension: *Bing\_Products\_Gourmet\_Food\_and\_Chocolate*. The *ConsumerID* is always the same: *{5B04B775-356B-4AA0-AAF8-6491FFEA5661}*. This specifies that the app is an extension to Bing search. The *TaskID* is always “\_default”, and the *ExtraFile* must be a relative path to the *Extras.xml* file in your project.

```
<Extensions>
  <Extension
    ExtensionName="Bing_Products_Gourmet_Food_and_Chocolate"
    ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFEA5661}"
    TaskID="_default"
    ExtraFile="Extensions\\Extras.xml" />
</Extensions>
```

Now that you’ve referred to the *Extras.xml* file, you should actually create it. To do so, make a new folder in your project named *Extensions*, add a new XML file to this folder, and then name it **Extras.xml**. You should set the Build Action for this to *Content*—in Microsoft Visual Studio 2012, this will be set by default. The *Extras.xml* must be in the *Extensions* folder, but the path you specify in the *ExtraFile* attribute can omit the *Extensions* root path and just specify “Extras.xml.” Either will work, so long as the file itself is in the right place. This file is where you specify the strings to be used in the Bing search results list for your app.

```

<?xml version="1.0" encoding="utf-8" ?>
<ExtrasInfo>
  <AppTitle>
    <default>My Search Extension</default>
  </AppTitle>
  <Consumer ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFE5661}">
    <ExtensionInfo>
      <Extensions>
        <ExtensionName>Bing_Products_Gourmet_Food_and_Chocolate</ExtensionName>
      </Extensions>
      <CaptionString>
        <default>All you need to know about coffee</default>
      </CaptionString>
    </ExtensionInfo>
  </Consumer>
</ExtrasInfo>

```

In *Extras.xml*, you can supply an *AppTitle* with at least one string for the default language. In Windows Phone 7.1, the *AppTitle* element is used to specify the title of the app as you would like it to appear in the apps pivot page of the Quick Card; it is therefore a required field. In Windows Phone 8, however, the system uses the app title from the phone's installed Apps list, instead; so, on this platform, the *AppTitle* is not required. In the *Consumer* section, the *ConsumerID* is again the same Bing search ID. Under this, you list all the *Extensions* that you want to register for Bing search. Again, you must provide at least one default *CaptionString*. The app's icon (the 62x62-pixel image) for an extension app should not use transparency—and, as always, you should test this to ensure that it displays correctly in both light and dark themes.

The next item you need is a custom URI mapper. You need this in order to map the navigation query string that Bing search passes to your app to the correct target page within your app. For this piece, add a new class file to your project and then change the code to derive your class from *UriMapperBase*. For example, map this incoming URI

```
/SearchExtras?ProductName=coffee&Category=Bing_Products_Gourmet_Food_and_Chocolate
```

to this target page, including the original query string parameters:

```
/MyTargetPage.xaml?ProductName=coffee&Category=Bing_Products_Gourmet_Food_and_Chocolate
```

In addition to determining the correct target page, you're free to do whatever other processing you want, including modifying the parameter list according to your requirements before passing it on, if you need to. When the app has been launched via Bing search App Connect, the URI will include the *"/SearchExtras"* substring. So, if you examine the URI and find that it does not include this substring, you should immediately return because this means that the app has been launched normally, not via Bing search.

```

public class MyUriMapper : UriMapperBase
{
    public override Uri MapUri(Uri uri)
    {
        String inputUri = uri.ToString();

```

```

        if (inputUri.Contains("/SearchExtras"))
        {
            if (inputUri.Contains("Bing_Products_Gourmet_Food_and_Chocolate"))
            {
                String outputUri = inputUri.Replace(
                    "/SearchExtras", "/MyTargetPage.xaml");
                return new Uri(outputUri, UriKind.Relative);
            }
        }
        return uri;
    }
}

```

A custom URI mapper must override the one and only virtual method, named *MapUri*. This takes in the search URI, as supplied by Bing. In your implementation, you typically parse the URI, and look first for the */SearchExtras* substring. If this is found, you can then go on to look for the Products category. In this example, the only category of interest is *Bing\_Products\_Gourmet\_Food\_and\_Chocolate*. This app provides only one target page for all search requests. In a sophisticated app, you might have multiple pages; if this is the case, you would need to implement a more complex decision tree to determine which page to return from the URI mapper. You would also typically search for more than one category and one product.

To use the URI mapper in your app, you create an object of this type and then set it as the value of the *UriMapper* property in the *RootFrame*. The best place to do this is at the end of the *InitializePhoneApplication* method in the *App* class.

```
RootFrame.UriMapper = new MyUriMapper();
```

This causes the system to load the specified target page and pass in all incoming URIs to your *UriMapper* so that they can be manipulated before navigation takes place. These incoming URIs include the query string as part of the *NavigationContext* that comes in to the page in the form of a dictionary of key-value pairs.

In the target page, you should override the *OnNavigatedTo* method so that you can examine the incoming query string. Look for an incoming *ProductName*. Having found the corresponding value for the key-value pair—in this case, *coffee*—you can then make a decision as to whether you know anything about this specific product. If so, you can then go on to do whatever domain logic you want based on this value. In this example, the app simply indicates that this is a known product by setting the *TextBlock.Text* value. The value of this string will be the full product name of the product that the user selected. Using the example in Figure 5-4, this would be *Blue Mountain Coffee Beans*.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    String product;
    if (NavigationContext.QueryString.TryGetValue("ProductName", out product))
    {
        if (product.ToLowerInvariant().Contains("coffee"))
        {
            Target.Text = String.Format("We know about {0}.", product);
        }
    }
}

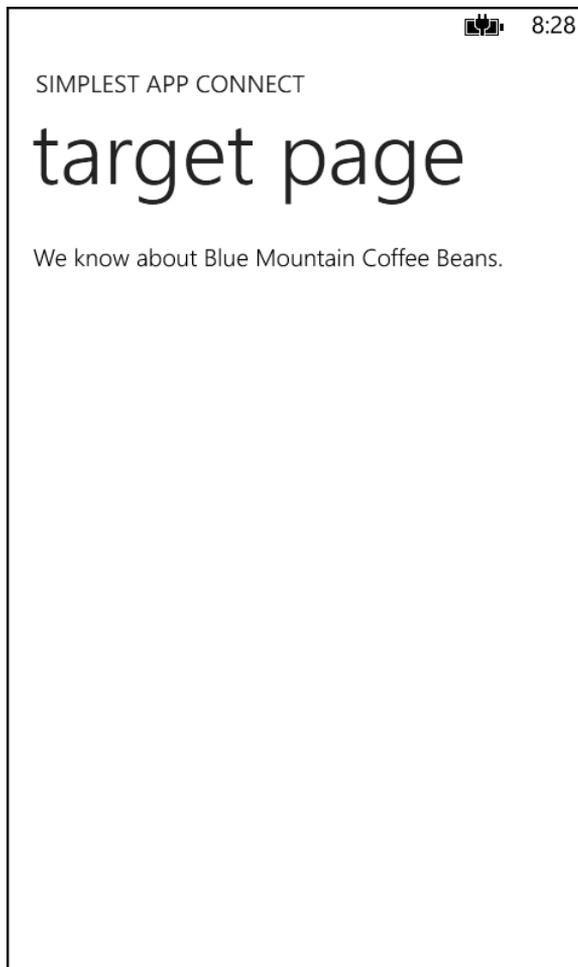
```

```

else
{
    Target.Text = String.Format("We don't know about {0}.", product);
}
}
}

```

With the code complete, you can test this either in the emulator or on a physical device. Tap the Bing search button and then enter **coffee**. In the primary search results, swipe over to the *shopping* pivot item, if it's not already selected, and then scroll down to find the products list. If necessary, tap the See More Products link to get to a coffee product that contains the string "coffee." Tap any such item; this takes you to the Quick Card for that item. In the Quick Card, swipe over to the apps pivot item. Your app should be listed there. When you tap the app to launch it, the URI mapper is invoked, and the app navigates to the target page and updates the text with the Bing search information, as shown in Figure 5-5.



**FIGURE 5-5** When the user taps your app in the apps pivot of the Quick Card, it navigates to your target page.

To test the functionality of the app in a more deterministic way, you can provide a fake launch query string in the *WMAppManifest.xml* file. For example, replace the *DefaultTask* entry with an entry that specifies a *SearchExtras* query string (without a leading slash). You can also test the negative case by providing a query string that should not result in listing your app in the search results.



**Note** Be aware that this technique must be used only for testing. The manifest submitted to the Store for publication must use your default page, without additional parameters.

```
<Tasks>
  <!--<DefaultTask Name="_default" NavigationPage="MainPage.xaml"/>-->
  <DefaultTask Name="_default" NavigationPage="SearchExtras?ProductName=coffee&
    Category=Bing_Products_Gourmet_Food_and_Chocolate"/>
  <!--<DefaultTask Name="_default" NavigationPage="SearchExtras?ProductName=bananas&
    Category=Bing_Products_Gourmet_Food_and_Chocolate"/>-->
</Tasks>
```

Suppose that you want to support more than one extension category and perhaps provide different caption strings for some or all of these. Or, suppose that you want to map the launch URI to one of several different target pages, according to some part of the query string. All of these behaviors are possible (see the *SimpleAppConnect* solution in the sample code). First, consider the requirement to support multiple extension categories. To do this, simply add each additional category in the Extensions section in your *WMAppManifest.xml* file. In the following listing, the app supports one of each of the three major categories, Products, Places, and Movies:

```
<Extensions>
  <Extension ExtensionName="Bing_Products_Gourmet_Food_and_Chocolate"
    ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFE5661}" TaskID="_default"
    ExtraFile="Extensions\Extras.xml" />
  <Extension ExtensionName="Bing_Places_Food_and_Dining"
    ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFE5661}"
    TaskID="_default" ExtraFile="Extensions\Extras.xml" />
  <Extension ExtensionName="Bing_Movies" ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFE5661}"
    TaskID="_default" ExtraFile="Extensions\Extras.xml" />
</Extensions>
```

Be aware that there's a difference between *Bing\_Movies* and *Bing\_Products\_Movies*—the latter will include results for movies that are not found to be showing in theatres; for example, if the user is searching for movies to buy on DVD. In the *Extras.xml*, you could group multiple *ExtensionName* entries to share the same caption strings. You can also divide your supported categories into groups, each with its own caption strings.

```
<ExtensionInfo>
  <Extensions>
    <ExtensionName>Bing_Products_Gourmet_Food_and_Chocolate</ExtensionName>
    <ExtensionName>Bing_Places_Food_and_Dining</ExtensionName>
  </Extensions>
```

```

    <CaptionString>
      <default>All you need to know about coffee</default>
    </CaptionString>
  </ExtensionInfo>

<ExtensionInfo>
  <Extensions>
    <ExtensionName>Bing_Movies</ExtensionName>
  </Extensions>
  <CaptionString>
    <default>Coffee in movies</default>
  </CaptionString>
</ExtensionInfo>

```

You could also enhance your URI mapper to target different pages for the different categories.

```

public override Uri MapUri(Uri uri)
{
    String inputUri = uri.ToString();
    if (inputUri.Contains("/SearchExtras"))
    {
        String targetPageName = "/MainPage.xaml";
        if (inputUri.Contains("Bing_Products"))
        {
            targetPageName = "/ProductTargetPage.xaml";
        }
        else if (inputUri.Contains("Bing_Places"))
        {
            targetPageName = "/PlaceTargetPage.xaml";
        }
        else if (inputUri.Contains("Bing_Movies"))
        {
            targetPageName = "/MovieTargetPage.xaml";
        }

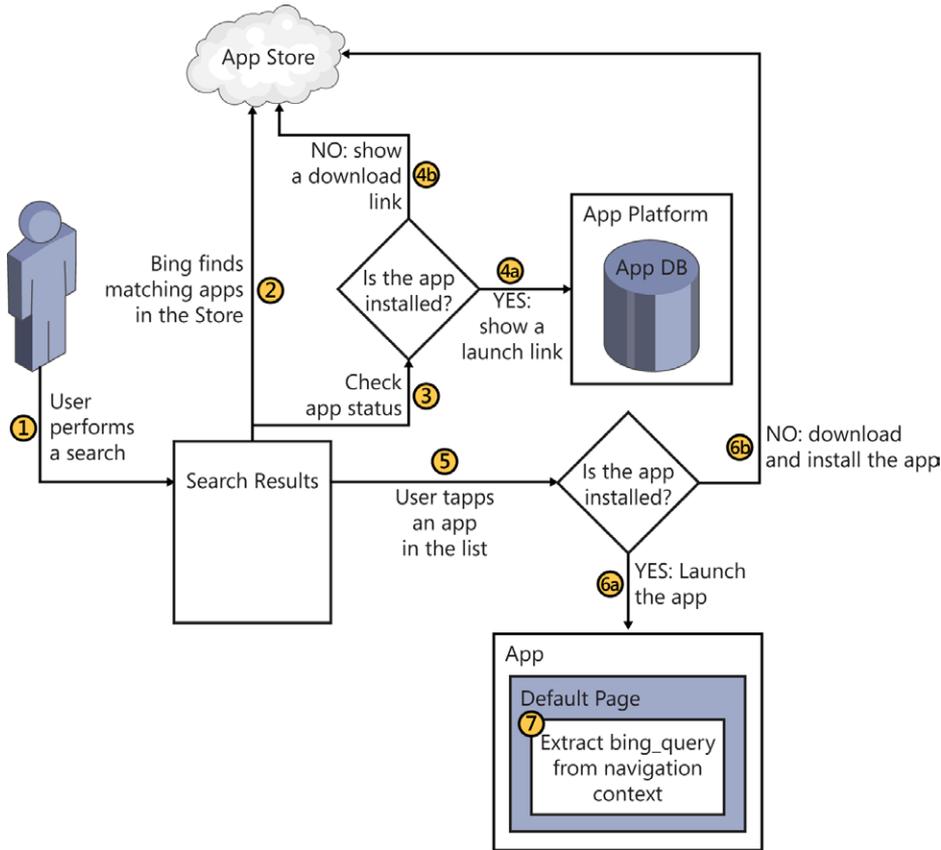
        String outputUri = inputUri.Replace("/SearchExtras", targetPageName);
        return new Uri(outputUri, UriKind.Relative);
    }
    return uri;
}

```

Clearly, you could take this a step further by pivoting your decisions off any of the elements of the query string. To ensure robustness, you should also decode the incoming URI (typically, by using *HttpUtility.UrlDecode*) before processing it and then re-encode it (by using *HttpUtility.UrlEncode*) before returning from your *MapUri* method.

## App Instant Answer

The second search extensibility model is simpler. It requires no special manifest entries, no *Extras.xml*, and no URI mapper. You have no choice about which page to launch based on the search results, and Bing will always launch your app using its default page. The way Bing identifies your app as being suitable for listing in the search results is internal to Bing. The overall model for App Instant Answer is summarized in Figure 5-6.



**FIGURE 5-6** The App Instant Answer extensibility model.

To create an App Instant Answer app, create a Windows Phone app as normal. An example of this is in the *SimpleAppInstantAnswer* solution in the sample code. When the user performs a Bing search, it might include apps in the web pivot. For example, if the user searches for “banana,” and your app name is “Banana Instant Answer,” this will match, and Bing will potentially add your app to the results list. On the other hand, if your app name is “Banoffee Instant Answer,” the match will fail. To set your app name, you set the *Title* attribute of the *App* element in your *WMAppManifest.xml*. Typically you set this in the project properties page in Visual Studio, although you can also edit the manifest manually, if you prefer, as follows:

```
Title="Banana Instant Answer"
```

There are two *Title* entries in the manifest: one is an attribute of the app element, the other is a subelement of the *Tokens* element. The app element’s *Title* attribute is the one that you want here. Also note that even if your app name exactly matches the user’s search term, there’s no guarantee that your app will be listed in the search results.

If you want to allow for the possibility that you’ll be included in search results for App Instant Answers, you should test for the *bing\_query* parameter in the navigation query string.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    String query;
    if (NavigationContext.QueryString.TryGetValue("bing_query", out query))
    {
        // Do something useful with this information.
    }
}
```

As with App Connect, you can test an App Instant Answer app by providing a fake launch URI in your *WMAppManifest.xml* file. But remember, just as before, this must be removed before submitting your app to the Windows Phone Store.

```
<Tasks>
  <!--<DefaultTask Name = "_default" NavigationPage="MainPage.xaml"/>-->
  <DefaultTask Name="_default" NavigationPage="MainPage.xaml?bing_query=Banana" />
</Tasks>
```

Finally, be aware that this is the only way that you can test your app, because even on the emulator, Bing only includes App Instant Answer apps from the published catalog in the Windows Phone Store. If it finds an app in the Windows Phone Store that is already installed on the phone, it will change the link from a Store download link to an installed app link, but it will not include an installed app unless it first finds a match for the app in the Windows Phone Store.

## Audio and video APIs

Windows Phone includes a range of techniques for working with media, both audio and video, in three broad categories that are described in Table 5-3.

**TABLE 5-3** Media techniques for Windows Phone

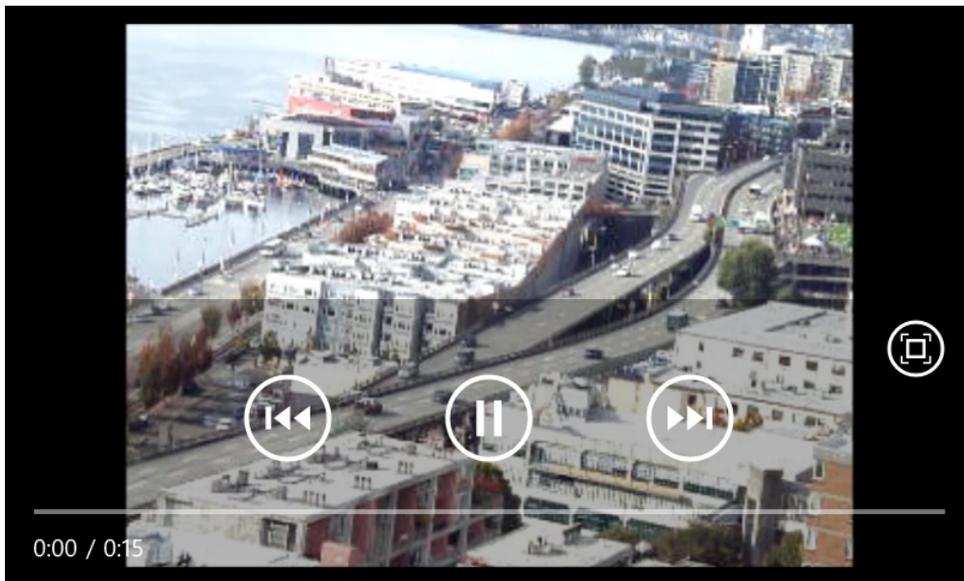
Category	Technique	Description
Media playback	<i>MediaPlayerLauncher</i>	A Launcher for playing audio or video with the built-in player experience. Primarily used for XNA videos.
	<i>MediaElement</i>	The primary wrapper class for audio and/or video files.
	<i>MediaStreamSource</i>	You can use this to work directly with the media pipeline. You use this most often to enable the <i>MediaElement</i> to use a container format not natively supported by the app platform.
	<i>BackgroundAudioPlayer</i>	Background audio agents use this to play music (see Chapter 9, "Background agents.")
Audio input and manipulation	<i>SoundEffect</i> , <i>SoundEffectInstance</i> , <i>DynamicSoundEffect</i>	XNA classes for working with audio content.
	<i>Microphone</i>	The only API for the microphone on the phone is the XNA <i>Microphone</i> class.
Platform integration	<i>MediaHistory</i>	You can use this to integrate your app with the Music + Videos hub.
	<i>MediaLibrary</i>	XNA class that provides access to the user's songs, playlists, and pictures.

## Media playback

The platform provides three main APIs for playing audio and video, each with varying levels of flexibility and control: the *MediaPlayerLauncher*, *MediaElement*, and *MediaStreamSource*. All three are described in the following sections.

### The *MediaPlayerLauncher*

The *MediaPlayerLauncher*, like all Launchers and Choosers provided by the app platform, is very easy to use. It is a simple wrapper that provides access to the underlying media player app without exposing any of the complexity. To use this in your app, you follow the same pattern as for other Launchers. You can see this at work in the *TestMediaPlayer* solution in the sample code (see Figure 5-7). Observe that the media player defaults to landscape mode for videos (you can't change this).



**FIGURE 5-7** You can very quickly add media player support for audio and video playback.

The following listing shows how to invoke the *MediaPlayerLauncher* with a media file that is deployed as *Content* within the app's XAP—the path will be relative to the app's install folder:

```
MediaPlayerLauncher player = new MediaPlayerLauncher();
player.Media = new Uri(@"Assets/Media/ViaductTraffic.mp4", UriKind.Relative);
player.Controls = MediaPlayerControls.Pause | MediaPlayerControls.Stop;
player.Location = MediaLocationType.Install;
player.Show();
```

You can assign the *Controls* property from a flags enum of possible controls. The preceding listing specifies only the *Pause* and *Stop* controls, whereas the listing that follows specifies all available controls (including *Pause*, *Stop*, *Fast Forward*, *Rewind*). The listing also shows how to specify a remote URL for the media file, together with the *MediaLocationType.Data*.

```
MediaPlayerLauncher player = new MediaPlayerLauncher();
player.Media = new Uri(
    @"http://media.ch9.ms/ch9/1eb0/f9621a51-7c01-4394-ae51-b581ab811eb0/DevPlatformDrillDown.wmv",
    UriKind.Absolute);
player.Controls = MediaPlayerControls.All;
player.Location = MediaLocationType.Data;
player.Show();
```

## The *MediaElement* class

The *TestMediaElement* solution in the sample code uses the *MediaElement* control. Unlike the media player, the *MediaElement* class is a *FrameworkElement* type that you can use in your app—superficially, at least—in a similar way as the *Image* type. This means that you get to choose the orientation and size of the element, apply transforms, and so on. You can set up a *MediaElement* in code or in XAML. The *MediaElement* class exposes a set of media-specific properties, including the following:

- **AutoPlay** This property defines whether to start playing the content automatically. The default is *true*, but in many cases you probably want to set it to *false* because of app lifecycle/tombstoning issues.
- **IsMuted** This defines whether sound is on (the default).
- **Volume** This property is set in the range 0 to 1, where 1 (the default) is full volume.
- **Stretch** This is the same property used by an *Image* control to govern how the content fills the control (the default is *Fill*).

The code that follows shows how to set up a *MediaElement* in XAML. This example sets up suitable properties, such as the media source file, whether the audio is muted, the audio volume, and how to render the image within the control.

```
<MediaElement
    x:Name="myVideo" Source="Assets/Media/campus_20111017.wmv" AutoPlay="False"
    IsMuted="False" Volume="0.5" Stretch="UniformToFill"/>
```

All that remains is to invoke methods such as *Play* and *Pause*; in this example, these are triggered in app bar button *Click* handlers.

```
private void appBarPlay_Click(object sender, EventArgs e)
{
    myVideo.Play();
}

private void appBarPause_Click(object sender, EventArgs e)
{
    myVideo.Pause();
}
```

This is enough to get started, but it is a little fragile: there is a very small chance in the app as it stands that the media file is not fully opened at the point when the user taps the play button, and this would raise an exception. To make the app more robust in this scenario, it would be better to have the app bar buttons initially disabled and to handle the *MediaOpened* event on the *MediaElement* object to enable them. You can set up a *MediaOpened* handler in XAML and then implement this in code to enable the app bar buttons.

```
<MediaElement
    x:Name="myVideo" Source="Assets/Media/campus_20111017.wmv" AutoPlay="False"
    IsMuted="False" Volume="0.5" Stretch="UniformToFill"
    MediaOpened="myVideo_MediaOpened" />

private void myVideo_MediaOpened(object sender, System.Windows.RoutedEventArgs e)
{
    ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = true;
    ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = true;
}
```

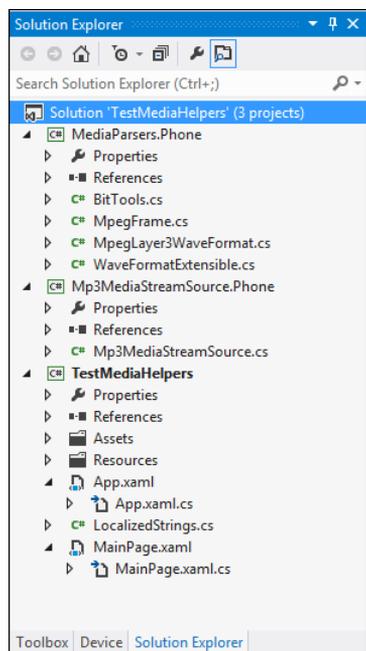
## The *MediaStreamSource* and *ManagedMediaHelpers* classes

The *MediaPlayerLauncher* provides the simplest approach for playing media in your app. Stepping it up a notch, if you need more flexibility, the *MediaElement* class offers a good set of functionality and is suitable for most phone apps. However, if you actually need lower-level access to the media file contents, you can use the *MediaStreamSource* class. This class offers more control over the delivery of content to the media pipeline and is particularly useful if you want to use media files in a format that are not natively supported by *MediaElement*, or for scenarios that are simply not yet supported in the platform, such as RTSP:T protocol support, SHOUTcast protocol support, seamless audio looping, ID3 v1/v2 metadata support, adaptive streaming, or multi-bitrate support.

Unfortunately, the *MediaStreamSource* class is not well documented. Fortunately, Microsoft has made available a set of helper classes, which you can obtain at <https://github.com/loarabia/ManagedMediaHelpers>. These were originally designed for use in Microsoft Silverlight Desktop and Windows Phone 7 apps, but they also work in Windows Phone 8 apps. The classes are provided in source-code format and include library projects and demonstration apps for Silverlight Desktop and Windows Phone. Keep in mind that the library source code is all in the Desktop projects; the phone projects merely reference the Desktop source files. The phone demonstration app is, of course, independent.

Here's how you can use these. First, create a phone app solution, as normal. Then, add the *ManagedMediaHelpers* library projects (either take copies, so that you have all the sources available, or build the library assemblies, and then use *CopyLocal=true* to reference them in your solution). If you add the library phone projects to your solution, you then need to copy across all the source files from the Desktop projects. You need two library projects: the *MediaParsers.Phone* and *Mp3MediaStreamSource.Phone* projects. These projects provide wrapper classes for the MP3 file format. Using this approach, you must copy the four C# files from the *MediaParsers.Desktop* project, and the one C# file from the *Mp3MediaStreamSource.SL4* project. The *Mp3MediaStreamSource.Phone* project has

a reference to the `MediaParsers.Phone` project. Your app needs to have a reference to the `Mp3MediaStreamSource.Phone` project. Figure 5-8 shows this setup, which is the `TestMediaHelpers` solution in the sample code.



**FIGURE 5-8** You can use the `ManagedMediaHelpers` for low-level control of media playback.

Having set up the projects, you can then declare an `Mp3MediaStreamSource` object. The sample app fetches a remote MP3 file by using an `HttpRequest`. When we get the data back, we use it to initialize the `Mp3MediaStreamSource` and set that as the source for a `MediaElement` object, which is declared in XAML.

```
private HttpRequest request;
private Mp3MediaStreamSource mss;
private string mediaFileLocation =
    @"http://media.ch9.ms/ch9/755d/4f893d13-fa05-4871-9123-3eadd2f0755d/
    EightPlatformAnnouncements.mp3";

public MainPage()
{
    InitializeComponent();
    Get = (ApplicationBarIconButton)ApplicationBar.Buttons[0];
    Play = (ApplicationBarIconButton)ApplicationBar.Buttons[1];
    Pause = (ApplicationBarIconButton)ApplicationBar.Buttons[2];
}
```

```

private void Get_Click(object sender, EventArgs e)
{
    request = WebRequest.CreateHttp(mediaFileLocation);
    request.AllowReadStreamBuffering = true;
    request.BeginGetResponse(new AsyncCallback(RequestCallback), null);
}

private void RequestCallback(IAsyncResult asyncResult)
{
    HttpWebResponse response =
        request.EndGetResponse(asyncResult) as HttpWebResponse;
    Stream s = response.GetResponseStream();
    mss = new Mp3MediaStreamSource(s, response.ContentLength);
    Dispatcher.BeginInvoke(() =>
    {
        mp3Element.SetSource(mss);
        Play.IsEnabled = true;
        Get.IsEnabled = false;
    });
}

private void Play_Click(object sender, EventArgs e)
{
    mp3Element.Play();
    Play.IsEnabled = false;
    Pause.IsEnabled = true;
}

private void Pause_Click(object sender, EventArgs e)
{
    mp3Element.Pause();
    Pause.IsEnabled = false;
    Play.IsEnabled = true;
}

```

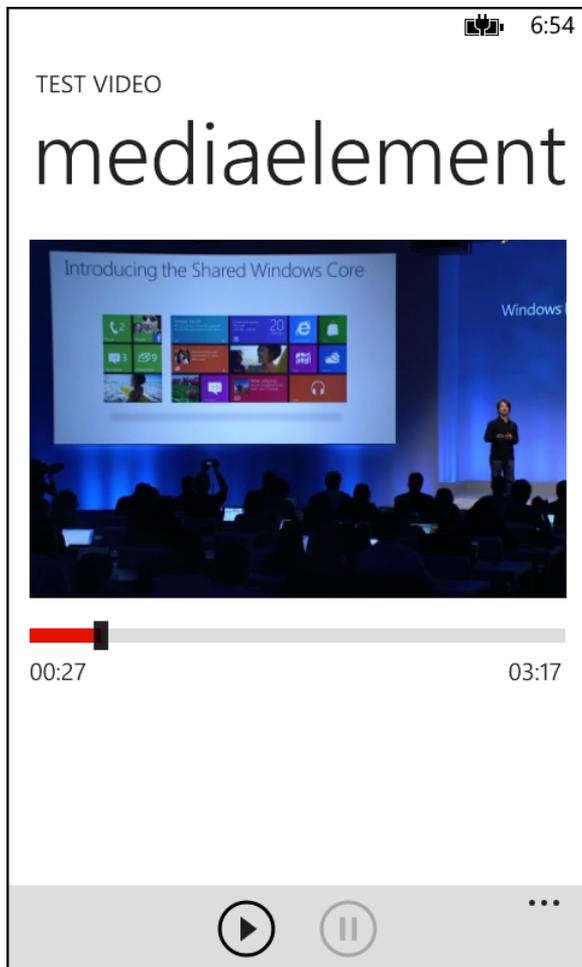
Observe that this code sets the *AllowReadStreamBuffering* property to *true*. If you enable buffering like this, it becomes easier to work with the stream source because all the data is downloaded first. On the other hand, you can't start processing the data until the entire file is downloaded—plus, it uses more memory. The alternative is to use the asynchronous methods and read the stream in the background. This simple example shows you how you can easily use the *MediaStreamSource* type via the *ManagedMediaHelpers*, although it doesn't really show the power of these APIs—by definition, these are advanced scenarios.

## The *MediaElement* controls

When you point a *MediaElement* to a remote media source and start playing, the content is downloaded to the device, and playback starts as soon as there is enough data in the buffer to play. Download and buffering continues in the background while the previously buffered content is playing. If you're interested in the progress of these operations, you can handle the *BufferingChanged* and *DownloadChanged* events exposed by the *MediaElement* class. The standard media player app on

the device, invoked via *MediaPlayerLauncher*, offers a good set of UI controls for starting, stopping, and pausing, as well as a timeline progress bar that tracks the current position in the playback, and a countdown from the total duration of the content. By contrast, the *MediaElement* class does not provide such UI controls; however, you can emulate these features by using the properties exposed from *MediaElement*, notably the *Position* and *NaturalDuration* values.

Figure 5-9 shows the *TestVideo* solution in the sample code. This uses a *MediaElement* combined with a *Slider* and *TextBlock* controls to mirror some of the UI features of the standard media player.



**FIGURE 5-9** You can report media playback progress with custom UI.

The app XAML declares a *MediaElement*, a *Slider*, and a couple of *TextBlock* controls (to represent the playback timer count-up and count-down values).

```

<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <MediaElement
        x:Name="Player" Height="297" Width="443" AutoPlay="False" Stretch="UniformToFill"
        Source="http://media.ch9.ms/ch9/b428/b746df27-e928-4306-9464-4b77c289b428/
SharedWindowsCore.wmv"/>
    <Slider
        x:Name="MediaProgress" Height="90" Margin="-5,0"
        Maximum="1" LargeChange="0.1" ValueChanged="MediaProgress_ValueChanged"/>
</StackPanel>
<TextBlock
    Grid.Row="1" x:Name="ElapsedTime" Text="00:00" IsHitTestVisible="False"
    Width="60" Height="30" Margin="19,180,0,0" HorizontalAlignment="Left" />
<TextBlock
    Grid.Row="1" x:Name="RemainingTime" Text="00:00" IsHitTestVisible="False"
    Width="60" Height="30" Margin="0,180,6,0" HorizontalAlignment="Right"/>

```

The *MediaElement* points to a video file on Channel9 (as it happens, this example is a presentation by Joe Belfiore, Microsoft vice president for Windows Phone). For the *Slider*, the important piece is to handle the *ValueChanged* event. Note that the two *TextBlock* controls are not part of the same *StackPanel*—this gives us the opportunity to specify *Margin* values and effectively overlay them on top of the *Slider*. Because of this, we need to be careful to make the *TextBlock* controls non-hit-testable so that they don't pick up touch gestures intended for the *Slider*.

The *Slider* performs a dual role: the first aspect is a passive role, in which we update it programmatically to synchronize it with the current playback position; the second aspect is an active role, in which the user can click or drag the *Slider* position—we respond to this in the app by setting the *MediaElement.Position* value. In the *MainPage* code-behind, we declare a *TimeSpan* field for the total duration of the video file, and a *bool* to track whether we're updating the *Slider* based on the current playback position.

```

private bool isUpdatingSliderFromMedia;
private TimeSpan totalTime;
private DispatcherTimer timer;

public MainPage()
{
    InitializeComponent();
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(0.5);
    timer.Tick += new EventHandler(timer_Tick);
    timer.Start();

    appBarPlay = ApplicationBar.Buttons[0] as ApplicationBarIconButton;
    appBarPause = ApplicationBar.Buttons[1] as ApplicationBarIconButton;
}

private void Player_MediaOpened(object sender, RoutedEventArgs e)
{
    timer.Start();
}

```

Here's how to implement the first role. We implement a *DispatcherTimer* with a half-second interval, updating the *Slider* on each tick. We wait until the *MediaElement* reports that the media source is successfully opened and then start the timer. When the timer event is raised, the first thing to do is to cache the total duration of the video file—this is a one-off operation. Next, we calculate the time remaining and render this in the corresponding *TextBlock*. Assuming that the playback has actually started (even if it is now paused), we then calculate how much of the video playback is complete and use the resulting value to update the position of the *Slider*. We also need to update the current "elapsed time" value to match the playback position. Throughout this operation, we toggle the *isUpdatingSliderFromMedia* flag. This will be used in another method.

```
private void timer_Tick (object sender, EventArgs e)
{
    if (totalTime == TimeSpan.Zero)
    {
        totalTime = Player.NaturalDuration.TimeSpan;
    }

    TimeSpan remainingTime = totalTime - Player.Position;
    String remainingTimeText = String.Format("{0:00}:{1:00}",
        (remainingTime.Hours * 60) + remainingTime.Minutes, remainingTime.Seconds);
    RemainingTime.Text = remainingTimeText;

    isUpdatingSliderFromMedia = true;
    if (Player.Position.TotalSeconds > 0)
    {
        double fractionComplete = Player.Position.TotalSeconds / totalTime.TotalSeconds;
        MediaProgress.Value = fractionComplete;
        TimeSpan elapsedTime = Player.Position;
        String elapsedTimeText = String.Format("{0:00}:{1:00}",
            (elapsedTime.Hours * 60) + elapsedTime.Minutes, elapsedTime.Seconds);
        ElapsedTime.Text = elapsedTimeText;
        isUpdatingSliderFromMedia = false;
    }
}
```

In the handler for the *ValueChanged* event on the *Slider*, we check first that we're not in this handler as a result of what we did in the previous method. That is, we need to verify that we're not here because we're updating the *Slider* from the media position. The other scenario for which we'd be in this handler is if the user is clicking or dragging the *Slider* position. In this case, assuming that the media content can actually be repositioned (*CanSeek* is *true*), we reset its position based on the *Slider* position. This is the inverse of the normal behavior, for which we set the *Slider* position based on the media position.

```
private void MediaProgress_ValueChanged(
    object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (!isUpdatingSliderFromMedia && Player.CanSeek)
    {
        TimeSpan duration = Player.NaturalDuration.TimeSpan;
        int newPosition = (int)(duration.TotalSeconds * MediaProgress.Value);
        Player.Position = TimeSpan.FromSeconds(newPosition);
    }
}
```

The app bar buttons invoke the *MediaElement Play* and *Pause* methods, each of which is very simple. In the case of *Pause*, we need to first establish that this media content can actually be paused. If you don't check *CanSeek* or *CanPause*, and just go ahead and attempt to set *Position* or call *Pause*, in neither case is an exception thrown. Rather, the method simply does nothing. So, these checks are arguably redundant, except that you should use them to avoid executing unnecessary code.

```
private void appBarPause_Click(object sender, EventArgs e)
{
    if (Player.CanPause)
    {
        Player.Pause();
    }
}
```

## Audio input and manipulation

---

Both *MediaElement* and *MediaStreamSource* give you some ability to manipulate media during playback. For even greater flexibility, you can use the *SoundEffect* and *SoundEffectInstance* classes. You can also use the *DynamicSoundEffectInstance* class in combination with the *Microphone* to work with audio input.

### The *SoundEffect* and *SoundEffectInstance* classes

As an alternative to using *MediaElement*, you can use the XNA *SoundEffect* classes, instead. One of the advantages is that you can play multiple *SoundEffects* at the same time, whereas you cannot play multiple *MediaElements* at the same time. Another advantage is that the *SoundEffect* class offers better performance than *MediaElement*. This is because the *MediaElement* carries with it a lot of UI baggage, relevant for a *Control* type. On the other hand, the *SoundEffect* class is focused purely on audio and has no UI features. The disadvantage of this is that it is an XNA type, so your app needs to pull in XNA libraries and manage the different expectations of the XNA runtime.

The *TestSoundEffect* solution in the sample code shows how to use *SoundEffect*. It also illustrates the *SoundEffectInstance* class, which offers greater flexibility than the *SoundEffect* class. A key difference is that *SoundEffect* has no *Pause* method; the playback is essentially “fire and forget.” The *SoundEffectInstance* also supports looping and 3D audio effects. You can create a *SoundEffectInstance* object from a *SoundEffect*, and this does have a *Pause* method. Also, you can create multiple *SoundEffectInstance* objects from the same *SoundEffect*; they'll all share the same content, but you can control them independently.

The sample app has two sound files, built as *Content* into the XAP (but not into the DLL). In the app, we first need to declare *SoundEffect* and *SoundEffectInstance* fields. Note that this pulls in the *Microsoft.Xna.Frameworks.dll*, and in Visual Studio 2012, you don't need to add this reference manually, because it's done for you. Early in the life of the app, we load the two sound files from the install

folder of the app by using *Application.GetResourceStream*. This can be slightly confusing, because we need to explicitly build the files as *Content* not *Resource*. However, *GetResourceStream* can retrieve a stream for either *Content* or *Resource*. If the sound file is a valid PCM wave file, you can use the *FromStream* method to initialize a *SoundEffect* object. For one of these *SoundEffect* objects, we create a *SoundEffectInstance*.

```
private SoundEffect sound;
private SoundEffectInstance soundInstance;

public MainPage()
{
    InitializeComponent();

    sound = LoadSound("Assets/Media/AfternoonAmbienceSimple_01.wav");
    SoundEffect tmp = LoadSound("Assets/Media/NightAmbienceSimple_02.wav");
    if (tmp != null)
    {
        soundInstance = tmp.CreateInstance();
    }
    InitializeXna();
}

private SoundEffect LoadSound(String streamPath)
{
    SoundEffect s = null;
    try
    {
        StreamResourceInfo streamInfo =
            App.GetResourceStream(new Uri(streamPath, UriKind.Relative));
        s = SoundEffect.FromStream(streamInfo.Stream);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.ToString());
    }
    return s;
}
```

Not only must the file be a valid WAV file, it must also be in the RIFF bitstream format, mono or stereo, 8 or 16 bit, with a sample rate between 8,000 Hz and 48,000 Hz. If the sound file was created on the phone with the same microphone device and saved as a raw audio stream (no file format headers), you could instead work with the stream directly and assume the same sample rate and *AudioChannels* values.

Also, very early in the life of the app, we must do some housekeeping to ensure that any XNA types work correctly. The basic requirement is to simulate the XNA game loop. This is the core architectural model in XNA, and most significant XNA types depend on this. XNA Framework event messages are placed in a queue that is processed by the XNA *FrameworkDispatcher*. In an XNA app, the XNA *Game* class calls the *FrameworkDispatcher.Update* method automatically whenever *Game.Update* is processed. This *FrameworkDispatcher.Update* method causes the XNA Framework to process

the message queue. If you use the XNA Framework from an app that does not implement the *Game* class, you must call the *FrameworkDispatcher.Update* method yourself to process the XNA Framework message queue.

There are various ways to achieve this. The simplest approach here is to set up a *DispatcherTimer* to call *FrameworkDispatcher.Update*. The typical tick rate for processing XNA events is 33 ms. The XNA game loop updates and redraws at 30 frames per second (FPS); that is one frame every 33 ms. It's a good idea to set up timers as class fields rather than local variables. This way, you can start and stop them out of band—such as in *OnNavigatedTo* and *OnNavigatedFrom* overrides.

```
private DispatcherTimer timer;
private void InitializeXna()
{
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromMilliseconds(33);
    timer.Tick += delegate { try { FrameworkDispatcher.Update(); } catch { } };
    timer.Start();
}

protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    timer.Stop();
}

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    timer.Start();
}
```

The app provides three app bar buttons. The *Click* handler for the first one simply plays the *SoundEffect* by invoking the “fire-and-forget” *Play* method. The other two are used to *Start* (that is, *Play*) or *Pause* the *SoundEffectInstance*. If the user taps the Play button to play the *SoundEffect* and then taps the *Start* button to play the *SoundEffectInstance*, she will end up with both audio files playing at the same time.

```
private void appBarPlay_Click(object sender, EventArgs e)
{
    if (sound != null)
    {
        sound.Play();
    }
}

private void appBarStart_Click(object sender, EventArgs e)
{
    if (soundInstance != null)
    {
        soundInstance.Play();
    }
}
```

```
private void appBarPause_Click(object sender, EventArgs e)
{
    if (soundInstance != null)
    {
        soundInstance.Pause();
    }
}
```

## Audio input and the microphone

The only way to work with audio input in a Windows Phone app is to use the XNA *Microphone* class. This provides access to the microphone (or microphones) available on the system. Although you can get the collection of microphones, the collection always contains exactly one microphone, so you would end up working with the default microphone, anyway. All microphones on the device conform to the same basic audio format, and return 16-bit PCM mono audio data, with a sample rate between 8,000 Hz and 48,000 Hz. The low-level audio stack uses an internal circular buffer to collect the input audio from the microphone device. You can configure the size of this buffer by setting the *Microphone.BufferDuration* property. *BufferDuration* is of type *TimeSpan*, so setting a buffer size of 300 ms will result in a buffer of  $2 * 16 * 300 = 9,600$  bytes. *BufferDuration* must be between 100 ms and 1000 ms, in 10-ms increments. The size of the buffer is returned by *GetSampleSizeInBytes*.

There are two different methods for retrieving audio input data:

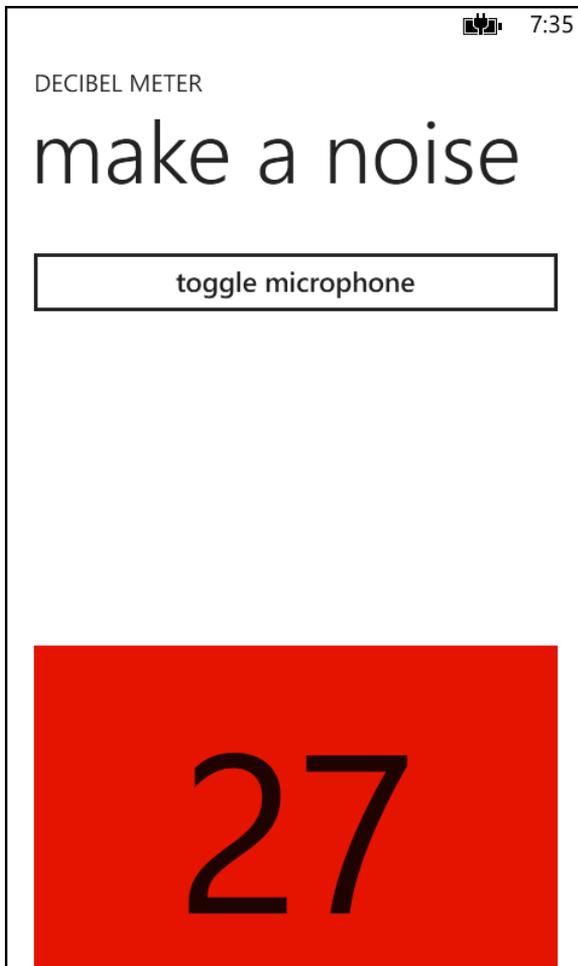
- Handle the *BufferReady* event and process data when there is a *BufferDuration*'s-worth of data received in the buffer. This has a minimum latency of 100 ms.
- Pull the data independently of *BufferReady* events, at whatever time interval you choose, including more frequently than 100 ms.

For a game, it can often be more useful to pull the data so that you can synchronize sound and action in a flexible manner. For a non-game app it is more common to respond to *BufferReady* events. With this approach, the basic steps for working with the microphone are as follows:

1. For convenience, cache a local reference to the default microphone.
2. Specify how large a buffer you want to maintain for audio input and declare a byte array for this data.
3. Hook up the *BufferReady* event, which is raised whenever a buffer's-worth of audio data is ready.
4. In your *BufferReady* event handler, retrieve the audio input data and do something interesting with it.
5. At suitable points, start and stop the microphone to start and stop the buffering of audio input data.

You might wonder what happens if your app is using the microphone to record sound and then a phone call comes in and the user answers it. Is the phone call recorded? The answer is “No,” specifically because this is a privacy issue. So, what happens is that your app keeps recording, but it records silence until the call is finished.

Figure 5-10 shows the *DecibelMeter* solution in the sample code, which illustrates simple use of the microphone. The app takes audio input data, converts it to decibels, and then displays a graphical representation of the decibel level, using both a rectangle and a text value. Note that this requires the *ID\_CAP\_MICROPHONE* capability in the app manifest.



**FIGURE 5-10** You can build a simple decibel meter to exercise the microphone.

The app XAML defines a *Grid* that contains a *Button* and an inner *Grid*. Inside the inner *Grid*, there’s a *Rectangle* and a *TextBlock*. These are both bottom-aligned and overlapping (the control declared last is overlaid on top of the previous one).

```

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <Button x:Name="ToggleMicrophone" Content="toggle microphone"
    Click="ToggleMicrophone_Click"/>
  <Grid Grid.Row="1" Height="535">
    <Rectangle x:Name="LevelRect"
      Height="0" Width="432" VerticalAlignment="Bottom"
      Margin="{StaticResource PhoneHorizontalMargin}" />
    <TextBlock
      Text="0" x:Name="SoundLevel" TextAlignment="Center" Width="432"
      FontSize="{StaticResource PhoneFontSizeHuge}"
      VerticalAlignment="Bottom"/>
  </Grid>
</Grid>

```

First, we declare a byte array for the audio data and a local reference to the default microphone. We then initialize these in the *MainPage* constructor. We specify that we want to maintain a 300-ms buffer for audio input. Whenever the buffer is filled, we'll get a *BufferReady* event. We retrieve the size of the byte array required to hold the specified duration of audio for this microphone object by using *GetSampleSizeInBytes* (this is how we know what size buffer to allocate). The following code also retrieves the current accent brush and sets this as the *Brush* object with which to fill the rectangle:

```

private byte[] soundBuffer;
private Microphone mic;

public MainPage()
{
    InitializeComponent();

    Brush accent = (Brush)Resources["PhoneAccentBrush"];
    LevelRect.Fill = accent;

    mic = Microphone.Default;
    mic.BufferDuration = TimeSpan.FromMilliseconds(300);
    mic.BufferReady += Microphone_BufferReady;
    int bufferSize = mic.GetSampleSizeInBytes(mic.BufferDuration);
    soundBuffer = new byte[bufferSize];
}

```

Whenever a buffer's-worth of audio input data is received, we pull that data from the *Microphone* object and copy it into our private buffer to work on it. We process this data by determining the average sound level in decibels and rendering text and graphics to represent that level. The rectangle height and position are constrained by the height of the containing grid.

```

private void Microphone_BufferReady(object sender, EventArgs e)
{
    int soundDataSize = mic.GetData(soundBuffer);
    if (soundDataSize > 0)

```

```

    {
        SoundLevel.Dispatcher.BeginInvoke(() =>
        {
            int decibels = GetSoundLevel();
            SoundLevel.Text = decibels.ToString();
            LevelRect.Height = Math.Max(0, Math.Min(
                ContentPanel.RowDefinitions[1].ActualHeight, decibels * 10));
        });
    }
}

```

The sound pressure level ratio in decibels is given by  $20 \cdot \log(\text{actual value} / \text{reference value})$ , where the logarithm is to base 10. Realistically, the <reference value> would be determined by calibration. In this example, we use an arbitrary hard-coded calibration value (300), instead. First, we must convert the array of bytes into an array of shorts. Then, we can convert these shorts into decibels.

```

private int GetSoundLevel()
{
    short[] audioData = new short[soundBuffer.Length / 2];
    Buffer.BlockCopy(soundBuffer, 0, audioData, 0, soundBuffer.Length);
    double calibrationZero = 300;
    double waveHeight = Math.Abs(audioData.Max() - audioData.Min());
    double decibels = 20 * Math.Log10(waveHeight / calibrationZero);
    return (int)decibels;
}

```

Finally, we provide a button in the UI so that the user can toggle the microphone on or off:

```

private void ToggleMicrophone_Click(object sender, RoutedEventArgs e)
{
    if (mic.State == MicrophoneState.Started)
    {
        mic.Stop();
    }
    else
    {
        mic.Start();
    }
}

```

As before, we need to ensure that the XNA types work correctly in a Silverlight app. Previously, we took the approach of a *DispatcherTimer* to provide a tick upon which we could invoke *FrameworkDispatcher.Update* in a simple fashion. A variation on this approach is to implement *IApplicationService* and put the *DispatcherTimer* functionality in that implementation. *IApplicationService* represents an extensibility mechanism in Silverlight. The idea is that where you have a need for some global “service” that needs to work across your app, you can register it with the runtime. This interface declares two methods: *StartService* and *StopService*. The Silverlight runtime will call *StartService* during app initialization, and it will call *StopService* just before the app terminates. Effectively, we’re taking the *InitializeXna* custom method from the previous example and reshaping it as an implementation of *IApplicationService*. Then, instead of invoking the method directly, we register the class and leave it to Silverlight to invoke the methods.

Following is the class implementation. As before, we simply set up a *DispatcherTimer* and invoke *FrameworkDispatcher.Update* on each tick.

```
public class XnaFrameworkDispatcherService : IApplicationService
{
    DispatcherTimer timer;

    public XnaFrameworkDispatcherService()
    {
        timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromTicks(333333);
        timer.Tick += OnTimerTick;
        FrameworkDispatcher.Update();
    }

    private void OnTimerTick(object sender, EventArgs args)
    {
        FrameworkDispatcher.Update();
    }

    void IApplicationService.StartService(AppserviceContext context)
    {
        timer.Start();
    }

    void IApplicationService.StopService()
    {
        timer.Stop();
    }
}
```

Registration is a simple matter of updating the *App.xaml* file to include the custom class in the *ApplicationLifetimeObjects* section.

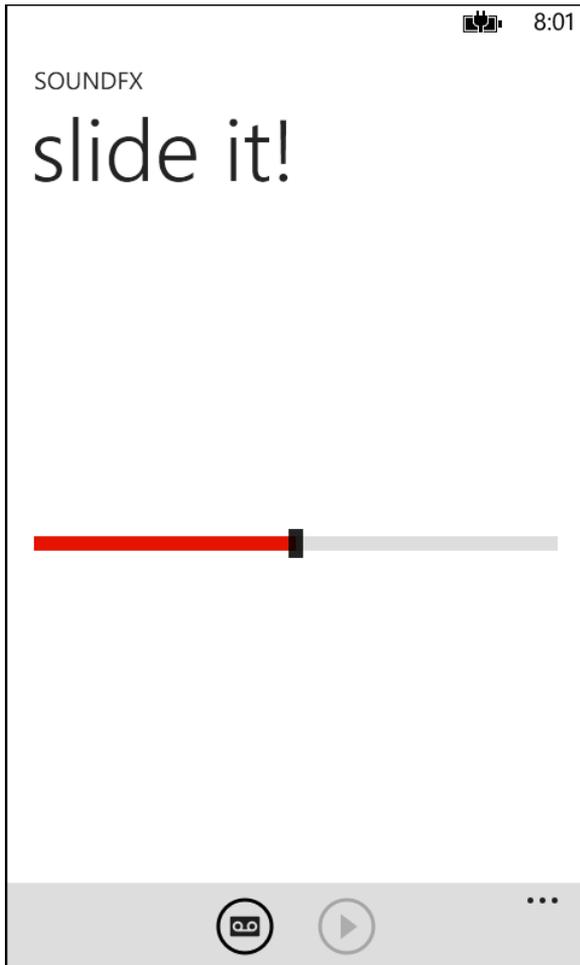
```
<Application
...standard declarations omitted for brevity.
    xmlns:local="clr-namespace:DecibelMeter">

    <Application.ApplicationLifetimeObjects>

        <local:XnaFrameworkDispatcherService />

        <shell:PhoneAppservice
            Launching="Application_Launching" Closing="Application_Closing"
            Activated="Application_Activated" Deactivated="Application_Deactivated"/>
    </Application.ApplicationLifetimeObjects>
</Application>
```

Figure 5-11 shows the *SoundFx* solution in the sample code. This uses the microphone to record sound and then plays back the sound. The app uses a slider to control the sound pitch on playback. This needs the *ID\_CAP\_MICROPHONE* capability in the app manifest.



**FIGURE 5-11** It's very simple to build sound recording and playback features.

In the *MainPage* constructor, we set up the XNA message queue processing, initialize the default microphone (with a 300-ms buffer), and create a private byte array for the audio data, as before. We then set the *SoundEffect.MasterVolume* to 1. This is relative to the volume on the device/emulator itself. You can set the volume in a range of 0 to 1, where 0 approximates silence, and 1 equates to the device volume. You cannot set the volume higher than the volume on the device. Each time the audio input buffer is filled, we get the data in the private byte array and then copy it to a *MemoryStream* for processing. Note that we need to protect the buffer with a *lock* object: this addresses the issue of the user pressing *Stop* while we're writing to the buffer (this would reset the buffer position to zero). The *Uri* fields and the *ButtonState* enum are used to change the images for the app bar buttons, because each one serves a dual purpose.

```

private byte[] soundBuffer;
private Microphone mic;
private MemoryStream stream;
private SoundEffectInstance sound;
private bool isRecording;
private bool isPlaying;
private DispatcherTimer timer;

private Uri recordUri = new Uri("/Assets/record.png", UriKind.Relative);
private Uri stopUri = new Uri("/Assets/stop.png", UriKind.Relative);
private Uri playUri = new Uri("/Assets/play.png", UriKind.Relative);
private enum ButtonState { Recording, ReadyToPlay, Playing };

public MainPage()
{
    InitializeComponent();

    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(0.33);
    timer.Tick += timer_Tick;
    timer.Start();

    mic = Microphone.Default;
    mic.BufferDuration = TimeSpan.FromMilliseconds(300);
    mic.BufferReady += Microphone_BufferReady;
    int bufferSize = mic.GetSampleSizeInBytes(mic.BufferDuration);
    soundBuffer = new byte[bufferSize];
    SoundEffect.MasterVolume = 1.0f;

    appBarRecord = ApplicationBar.Buttons[0] as ApplicationBarIconButton;
    appBarPlay = ApplicationBar.Buttons[1] as ApplicationBarIconButton;
}

private void timer_Tick(object sender, EventArgs e)
{
    FrameworkDispatcher.Update();
    if (isPlaying && sound.State != SoundState.Playing)
    {
        isPlaying = false;
        UpdateAppBarButtons(ButtonState.ReadyToPlay);
    }
}

private void Microphone_BufferReady(object sender, EventArgs e)
{
    lock (this)
    {
        mic.GetData(soundBuffer);
        stream.Write(soundBuffer, 0, soundBuffer.Length);
    }
}

```

Notice that we have to poll the *SoundEffectInstance* to see when its state changes because the class doesn't expose a suitable event for this. The user can tap the app bar buttons to start and stop the recording. We handle these by calling *Microphone.Start* and *Microphone.Stop*. When the user chooses to start a new recording, we close any existing stream and set up a fresh one and then start the microphone. Conversely, when the user asks to stop recording, we stop the microphone and reset the stream pointer to the beginning.

```
private void appBarRecord_Click(object sender, EventArgs e)
{
    if (isRecording)
        StopRecording();
    else
        StartRecording();
}

private void appBarPlay_Click(object sender, EventArgs e)
{
    if (isPlaying)
        StopPlayback();
    else
        StartPlayback();
}

private void StartRecording()
{
    if (stream != null)
        stream.Close();
    stream = new MemoryStream();
    mic.Start();
    isRecording = true;
    UpdateAppBarButtons(ButtonState.Recording);
}

private void StopRecording()
{
    mic.Stop();
    stream.Position = 0;
    isRecording = false;
    UpdateAppBarButtons(ButtonState.ReadyToPlay);
}

private void UpdateAppBarButtons(ButtonState state)
{
    switch (state)
    {
        case ButtonState.Recording:
            appBarRecord.IconUri = stopUri;
            appBarRecord.Text = "stop";
            appBarRecord.IsEnabled = true;
            appBarPlay.IsEnabled = false;
            break;
    }
}
```

```

        case ButtonState.ReadyToPlay:
            appBarRecord.IconUri = recordUri;
            appBarRecord.Text = "record";
            appBarRecord.IsEnabled = true;
            appBarPlay.IconUri = playUri;
            appBarPlay.Text = "play";
            appBarPlay.IsEnabled = true;
            break;
        case ButtonState.Playing:
            appBarRecord.IconUri = recordUri;
            appBarRecord.Text = "record";
            appBarRecord.IsEnabled = false;
            appBarPlay.IconUri = stopUri;
            appBarPlay.Text = "stop";
            appBarPlay.IsEnabled = true;
            break;
    }
}

```

The only other interesting code is starting and stopping playback of the recorded sound. To start playback, we first create a new *SoundEffect* object from the buffer of microphone data. Then, we create a new *SoundEffectInstance* from the *SoundEffect* object, varying the pitch to match the slider value. We also set the Volume to 1.0 relative to the *SoundEffect.MasterVolume*; the net effect is to retain the same volume as the device itself. To stop playback, we simply call *SoundEffectInstance.Stop*, as before.

```

private void StartPlayback()
{
    SoundEffect se = new SoundEffect(stream.ToArray(), mic.SampleRate, AudioChannels.Mono);
    sound = se.CreateInstance();
    sound.Volume = 1.0f;
    sound.Pitch = (float)Frequency.Value;
    sound.Play();
    isPlaying = true;
    UpdateAppBarButtons(ButtonState.Playing);
}

private void StopPlayback()
{
    if (sound != null)
        sound.Stop();
    isPlaying = false;
    UpdateAppBarButtons(ButtonState.ReadyToPlay);
}

```

We can take this one step further by persisting the recorded sound to a file in isolated storage. You can see this at work in the *SoundFx\_Persist* solution in the sample code. To persist the sound, we can add a couple of extra app bar buttons for *Save* and *Load*. To save the data, we simply write out the raw audio data by using the isolated storage APIs. This example uses a .pcm file extension because the data is in fact PCM wave data. However, this is not a WAV file in the normal sense, because it is missing the header information that describes the file format, sample rate, channels, and so on.

```

private const string soundFile = "SoundFx.pcm";

private void appBarSave_Click(object sender, EventArgs e)
{
    using (IsolatedStorageFile storage =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream isoStream =
            storage.OpenFile(soundFile, FileMode.Create, FileAccess.Write))
        {
            byte[] soundData = stream.ToArray();
            isoStream.Write(soundData, 0, soundData.Length);
        }
    }
}

private void appBarLoad_Click(object sender, EventArgs e)
{
    using (IsolatedStorageFile storage =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream isoStream =
            storage.OpenFile(soundFile, FileMode.Open, FileAccess.Read))
        {
            stream = new MemoryStream();

            isoStream.CopyTo(stream, (int)isoStream.Length);
        }
    }
}

```

You've seen already that you can use the *SoundEffect* class to load a conventional WAV file (including header) from disk. There's no support in *SoundEffect*—or indeed any other Silverlight or XNA classes—for saving WAV files with header information. This is not generally a problem on Windows Phone, because if the same app is both recording the data and playing it back, it can precisely control the file contents without the need for a descriptive header. On the other hand, if you need to record audio on the phone and then transmit it externally (for example, via a web service) to a consuming user or app that is using a different device (perhaps a PC, not a phone at all), you need to save a descriptive header in the file along with the audio data.

One solution to this is the NAudio library. NAudio is an open-source Microsoft .NET audio and MIDI library that contains a wide range of useful audio-related classes intended to speed development of audio-based managed apps. NAudio is licensed under the Microsoft Public License (Ms-PL), which means that you can use it in whatever project you like, including commercial projects. It is available at <http://naudio.codeplex.com/>.

## The *DynamicSoundEffectInstance* class

So far, we've used the *SoundEffect* and *SoundEffectInstance* classes to play back audio streams, either from static audio content or from dynamic microphone input. The *DynamicSoundEffectInstance* is derived from *SoundEffectInstance*. The critical difference is that it exposes a *BufferNeeded* event. This

is raised when it needs more audio data to play back. You can provide the audio data from static files or from dynamic microphone input; however, the main strength of this feature is that you can manipulate or compute the audio data before you provide it. Typically, you would modify source data, or even compute the data entirely from scratch.

The *TestDynamicSounds* solution in the sample code does just that: it provides a simple sound based on a sine wave. Sound is the result of a vibrating object creating pressure oscillations—that is, variations in pressure over time—in the air. A variation over time is modeled in mathematical terms as a wave. A wave can be represented by a formula that governs how the amplitude (or height) of the signal varies over time, and the frequency of the oscillations. Given two otherwise identical waves, if one has higher amplitude it will be louder; if one has greater frequency it will have a higher pitch. A wave is continuous, but you need to end up with a buffer full of discrete items of audio data, whereby each datapoint is a value that represents a sample along the wave.

With this basic context, we can get started with dynamic sounds. First, we need to declare fields for the *DynamicSoundEffectInstance*, a sample rate set to the maximum achievable on the device (48,000), and a buffer to hold the sound data. You can get the required buffer size from the *DynamicSoundEffectInstance* object. For the purposes of this example, set the frequency to an arbitrary value of 300.

```
private DynamicSoundEffectInstance dynamicSound;  
private const int sampleRate = 48000;  
private int bufferSize;  
private byte[] soundBuffer;  
private int totalTime = 0;  
private double frequency = 300;
```

At a suitable early point—for example, in the *MainPage* constructor—you would set up your preferred method for pumping the XNA message queue. We want to initialize the *DynamicSoundEffectInstance* early on, but the catch is that the constructor is too early because you won't yet have started pumping the XNA message queue. One solution is to hook up the *Loaded* event on the page and do your initialization of the XNA types there, but there is a possible race condition with that approach. The simplest approach is to just pump the XNA message queue once first, before performing initialization. Apart from the timing aspect, the key functional requirement is to hook up the *BufferNeeded* event. This will be raised every time the audio pipeline needs input data.

```
public MainPage()  
{  
    InitializeComponent();  
  
    timer = new DispatcherTimer();  
    timer.Interval = TimeSpan.FromMilliseconds(33);  
    timer.Tick += delegate { try { FrameworkDispatcher.Update(); } catch { } };  
    timer.Start();  
    FrameworkDispatcher.Update();  
}
```

```

dynamicSound = new DynamicSoundEffectInstance(sampleRate, AudioChannels.Mono);
dynamicSound.BufferNeeded += dynamicSound_BufferNeeded;
dynamicSound.Play();
bufferSize = dynamicSound.GetSampleSizeInBytes(TimeSpan.FromSeconds(1));
soundBuffer = new byte[bufferSize];
}

```

In the handler for the *BufferNeeded* event, the task is to fill in the byte array of sound data. In this example, we fill it with a simple sine wave. The basic formula for a sine wave as a function of time is as follows:

$$y(t) = A \cdot \sin(\omega t + \varphi)$$

Where:

- **A = amplitude** This is the peak deviation of the function from its center position (loudness).
- **$\omega$  = frequency** This is how many oscillations occur per unit of time (pitch).
- **$\varphi$  = phase** This is the point in the cycle at which the oscillation begins.

In this example, for the sake of simplicity, we can default the amplitude to 1 (parity with the volume on the device), and the phase to be zero (oscillation starts at the beginning of the cycle). We loop through the whole buffer, 2 bytes (that is, 16 bits: one sample) at a time. For each sample, we compute the floating-point value of the sine wave and convert it to a short (16 bits). The double value computed from the sine wave formula is in the range  $-1$  to  $1$ , so we multiply by the *MaxValue* for a short in order to get a short equivalent of this.

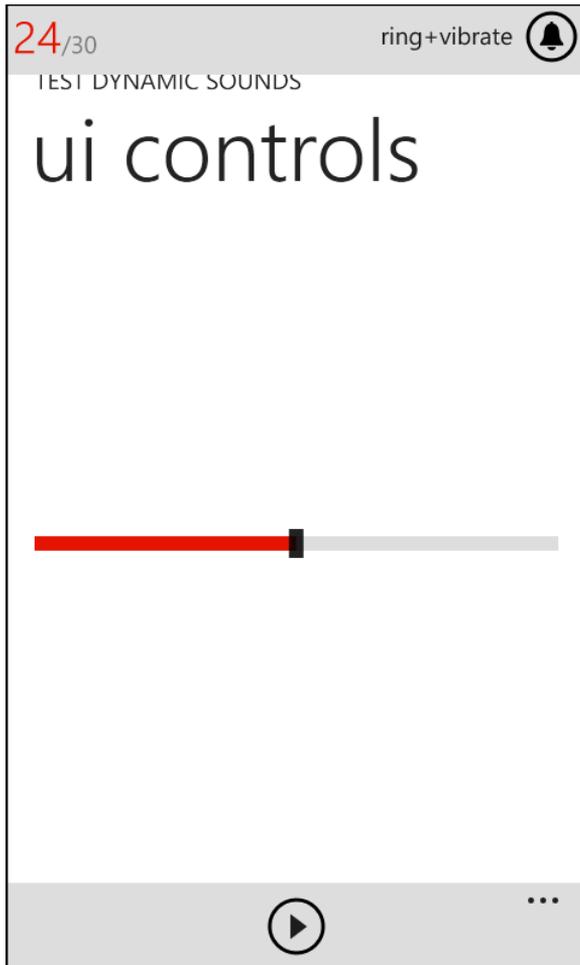
Then, we need to store the short as 2 bytes. The low-order byte of the short is stored as an element in the sample array and then the high-order byte is stored in the next element. We fill the second byte with the low-order byte of the short by bit-shifting 8 bits to the right. Finally, we submit the newly filled buffer to the *DynamicSoundEffectInstance* so that it can play it back.

```

private void dynamicSound_BufferNeeded(object sender, EventArgs e)
{
    for (int i = 0; i < bufferSize - 1; i += 2)
    {
        double time = (double)totalTime / (double)sampleRate;
        short sample =
            (short)(Math.Sin(2 * Math.PI * frequency * time) * (double)short.MaxValue);
        soundBuffer[i] = (byte)sample;
        soundBuffer[i + 1] = (byte)(sample >> 8);
        totalTime++;
    }
    dynamicSound.SubmitBuffer(soundBuffer);
}

```

The result is a continuously oscillating tone. Figure 5-12 shows a variation on this app (*TestDynamicSounds\_Controls* in the sample code), which includes an app bar button to start/stop the playback, and a *Slider* to control the frequency of the wave.



**FIGURE 5-12** You can use *DynamicSoundEffectInstance* to manipulate audio data before playback.

The XAML defines a *Slider*, with its range set at 1.0 to 1000.0, and initial position set at halfway along the range, as demonstrated in the following:

```
<Slider
  Grid.Row="1" Margin="12,0,12,0"
  x:Name="Frequency" Minimum="1.0" Maximum="1000.0" Value="500.0" />
```

The implementation of the *BufferNeeded* event handler is changed slightly to use the *Slider* value instead of the fixed frequency value:

```
short sample =
  (short)(Math.Sin(2 * Math.PI * Frequency.Value * time) * (double)short.MaxValue);
```

The only other work is to respond to button *Click* events to start and stop the playback:

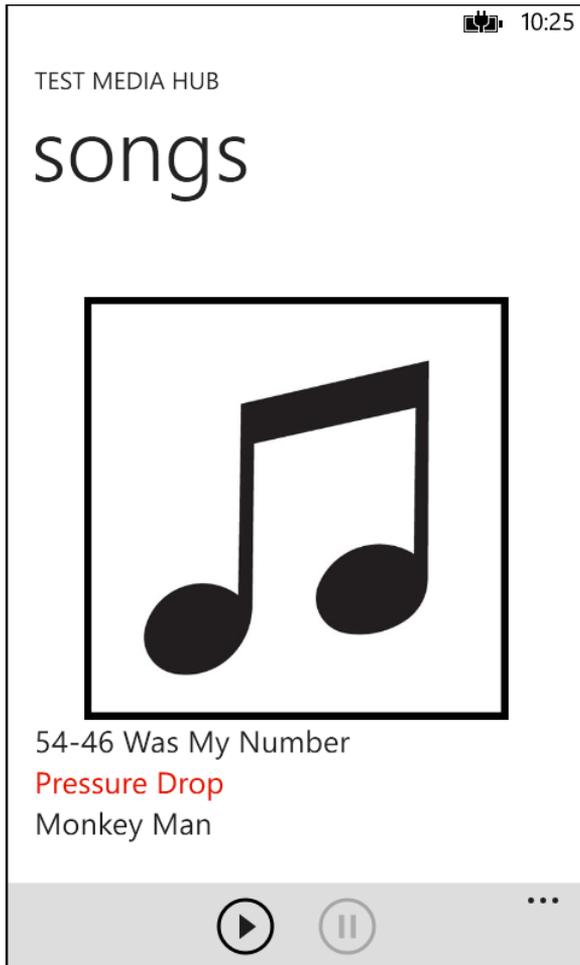
```
private void appBarPlay_Click(object sender, EventArgs e)
{
    if (isPlaying)
    {
        dynamicSound.Stop();
        appBarPlay.IconUri = new Uri("/Assets/play.png", UriKind.Relative);
        appBarPlay.Text = "play";
        isPlaying = false;
    }
    else
    {
        dynamicSound.Play();
        appBarPlay.IconUri = new Uri("/Assets/stop.png", UriKind.Relative);
        appBarPlay.Text = "stop";
        isPlaying = true;
    }
}
```

When this app runs, the user can manipulate the slider to control the data that's fed into the playback buffer. Because we've tied the amplitude to the volume on the device, the user can change the volume of the playback by invoking the universal volume control (UVC), as shown in Figure 5-12. On the emulator, this is invoked by pressing F10 while audio playback is ongoing; press F10 to decrease the volume and F9 to increase it. On the device, this is invoked by the hardware volume controls.

## Music and Videos Hub

---

The Music and Videos Hub on Windows Phone is a centralized location for accessing the phone's music and videos library. Figure 5-13 shows an app that integrates with the Music and Videos Hub to fetch a list of all songs in the library and render them in a *ListBox*. This is the *TestMediaHub* solution in the sample code. When the user selects an item from the *ListBox*, the app fetches the selected song's album art and presents buttons with which he can play/pause the selected song.



**FIGURE 5-13** Your app can integrate with the Music and Videos Hub on the phone.

The *MainPage* class declares fields for the *MediaLibrary* itself and for the current *Song*. As always, you need to ensure that you pump the XNA message queue. Next, we initialize the *MediaLibrary* field and set the collection of *Songs* to be the *ItemsSource* on your *ListBox*. In the XAML, we data-bind the *Text* property on the *ListBox* items to the *Name* property on each *Song*.

```
private MediaLibrary library;  
private Song currentSong;  
  
public MainPage()  
{  
    InitializeComponent();
```

```

Play = ApplicationBar.Buttons[0] as ApplicationBarItemButton;
Pause = ApplicationBar.Buttons[1] as ApplicationBarItemButton;

DispatcherTimer dt = new DispatcherTimer();
dt.Interval = TimeSpan.FromMilliseconds(33);
dt.Tick += delegate { FrameworkDispatcher.Update(); };
dt.Start();

Library = new MediaLibrary();
HistoryList.ItemsSource = library.Songs;
}

```

You can't effectively test this on the emulator, because it has no songs. If you test on a physical device, you would want one that has a representative number of songs. If there are very many songs on the device, initializing the list could be slow—and in this case you could restrict the test to perhaps the first album by using the following syntax:

```
HistoryList.ItemsSource = library.Albums.First().Songs;
```

The app's play and pause operations are more or less self-explanatory, invoking the *MediaPlayer* *Play* or *Pause* methods.

```

private void Play_Click(object sender, EventArgs e)
{
    MediaPlayer.Play(currentSong);
    Pause.IsEnabled = true;
    Play.IsEnabled = false;
}

private void Pause_Click(object sender, EventArgs e)
{
    MediaPlayer.Pause();
    Play.IsEnabled = true;
}

```

The interesting work is done in the *SelectionChanged* handler for the *ListBox*. Here, we fetch the currently selected item and fetch its album art to render in an *Image* control in the app's UI. If we can't find any corresponding album art, we use a default image built in to the app.

```

private void HistoryList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (HistoryList.SelectedIndex == -1)
    {
        return;
    }

    currentSong = HistoryList.SelectedItem as Song;
    if (currentSong != null)
    {
        Play.IsEnabled = true;
    }
}

```

```

Stream albumArtStream = currentSong.Album.GetAlbumArt();
if (albumArtStream == null)
{
    StreamResourceInfo albumArtPlaceholder =
        App.GetResourceStream(new Uri(
            "Assets/AlbumArtPlaceholder.jpg", UriKind.Relative));
    albumArtStream = albumArtPlaceholder.Stream;
}
BitmapImage albumArtImage = new BitmapImage();
albumArtImage.SetSource(albumArtStream);
MediaImage.Source = albumArtImage;
}
}

```

## The Clipboard API

---

Windows Phone 8 includes programmatic support for the clipboard, albeit in a constrained manner. You can set text into the system-wide clipboard, but you cannot extract text from it programmatically. This constraint is for security and privacy reasons, and it ensures that the user is always in control of where the clipboard contents might be sent. Figure 5-14 shows the *TestClipboard* application in the sample code.

This example offers a *RichTextBox* at the top with a *Button* below it, and a regular *TextBox* at the bottom. The *RichTextBox* is populated with some dummy text. The reason for this choice of controls is that you want to get your text from a control that supports selection of its contents, which the *RichTextBox* does. You also need to make an editable text control available into which the user can paste; hence, the *TextBox*. When the user taps the button, you arbitrarily select some or all of the text in the *RichTextBox* and then set this text into the clipboard by using the static *Clipboard.SetText* method.

```

private void copyText_Click(object sender, RoutedEventArgs e)
{
    textSource.SelectAll();
    Clipboard.SetText(textSource.Selection.Text);
}

```

After this, if and when the user chooses to tap the regular *TextBox*, the standard phone UI will show the Soft Input Panel (SIP) and include a paste icon, indicating that there is some text in the clipboard. If the user taps this icon, the clipboard contents are pasted into the *TextBox*. This last operation is outside your control and is handled entirely by the phone platform. Be aware that the clipboard is cleared whenever the phone lock engages.

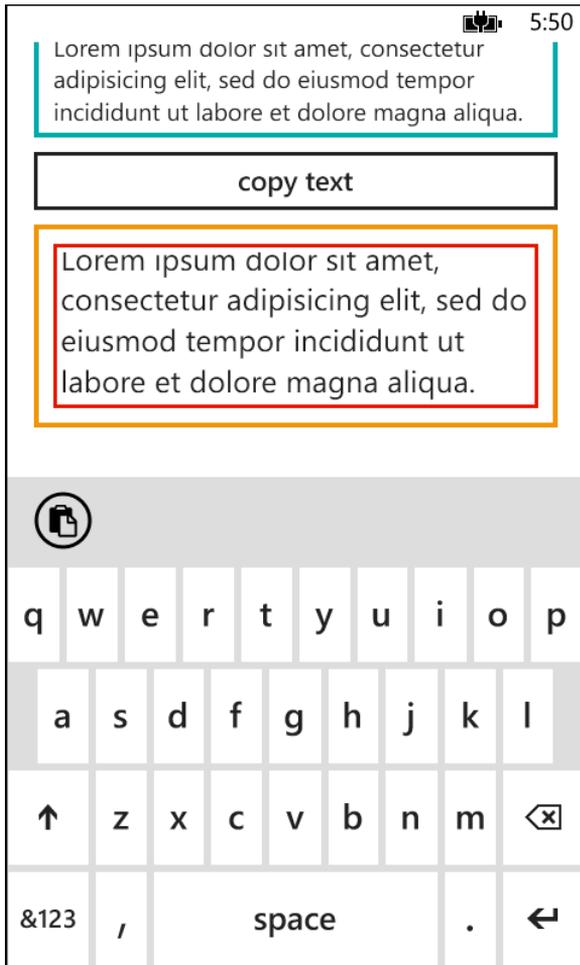


FIGURE 5-14 A simple clipboard application.

## Summary

---

In this chapter, you examined the different levels and types of app platform support for integrating your app with standard features and services on the phone. Built-in apps such as the camera, email, browser, search, and connection to the Windows Phone Store are all exposed by a consistent set of wrapper classes that take care of the complex internal behavior and cross-app hook-ups, all while providing a developer-friendly API surface with which to work. The app platform also provides three broad categories of API support for building audio and video features into your app. There are three main classes for media playback at varying levels of flexibility. Audio input via the microphone and low-level manipulation of audio data is enabled through a second set of classes. Integration with the phone's media hub is enabled through the XNA *MediaLibrary* class. With judicious use of these APIs, you can easily build a very compelling, media-focused user experience into your app. Additional *MediaLibrary* extensibility is covered in Chapter 16, "Camera and photos."

# Index

## Symbols

%20 (space), 761  
/dataservice switch, 327  
" (double quotation marks)  
    &quot; entity and, 146  
/language switch, 327  
.NET Compact Framework (.NET CF), 520  
/ operator (backslash), 146  
/out switch, 327  
/uri switch, 327  
/version switch, 327

## A

absolute screen layout model, 84  
AccelerationHelper class, 245  
Accelerometer class (Windows.Devices.Sensors namespace), 238  
    SensorReadingEventArgs<AccelerometerReading> events, 238  
    TimeBetweenUpdates property, 241  
AccelerometerHelper class  
    Shake Gesture Library and, 252  
    TestAccelerometerHelper\_CurrentValueChanged solution, 249  
    usage, 246–249  
Accelerometer property  
    range of values for, 239  
AccelerometerReading type, 240  
accelerometer sensor, 238–256  
    and Direct3D apps, 960  
    FilteredAccelerometer app, 242  
    gravity and, 238  
    Level Starter Kit, 245–252  
    shake, 252–256  
    Shake Gesture Library, 252  
    SimpleAccelerometer solution, 239  
    testing in emulator, 241  
Access Token (Twitter), 302  
Accounts property (Contact), 597  
accuracy (location)  
    DesiredAccuracyInMeters property, 722  
    performance and, 748–749  
Activated events/handlers, 523  
    IsApplicationInstancePreserved property, 55  
    page creation order, 49  
    resume policy and, 44  
ActiveX controls  
    in browser, 281  
    support, 846  
adapt code, 461  
adaptive streaming, 205  
Ad Control, 811–822  
    design guidelines for, 816–819  
    hardcoded information in, 814  
    maintaining consistency across pages, 817–819  
    Microsoft Advertising SDK, 812–815  
    Microsoft pubCenter and, 819–822  
    panorama control and, 817  
    pivot control and, 817  
    positioning, 816  
    required capabilities for, 813  
AddContactInfo solution, 602  
AddGrammarFromList API, 772–773  
adding tiles vs. upgrading, 517  
Additional Tile Templates, 510  
AddPackageAsync, 862  
AddressChooserTask, 189  
AddSecondaryTile method, 517, 519  
AddWalletItemTask, 189, 799  
ADO.NET Entity Data Model, 325  
AdSupportedApp solution, 816  
AdUnitId attribute, 814

## Advanced Encryption Standard (AES)

- Advanced Encryption Standard (AES), 846
- advertising
  - Ad Control, 811–822
  - categories, specifying, 820
  - Microsoft Advertising SDK, 812
  - pubCenter (Microsoft), 819–822
- AES. *See* Advanced Encryption Standard
- AET. *See* application enrollment token
- AETGenerator tool, 852
- AET installation, 854
- affinity groups, creating, 342
- Alarms API, 11
- alarms (background), 351–354
  - limit on number of, 351
- Albums collection (MediaLibrary), 625
- allocating more memory, 463
- AllowReadStreamBuffering property, 207
- AlternateIdentities property (PeerFinder class), 694
  - Bluetooth devices and, 696
- Android and tap-to-connect, 702
- ANID. *See* Anonymous ID
- animation, 103
- Anonymous ID (ANID) (UserExtendedProperties class), 524
- apartments (COM), 946
- APIs, 297
  - DeviceStatus, 502
  - Facebook, 295
  - Facebook Query Language (FQL), 297
  - Graph, 297
  - overlaps between Windows Phone 7.x and 8, 908
  - Package.Launch, 863
  - REST, 297
- app alias, 446
- app analysis
  - profiling, 493
  - report, 495
- app bar, 78–79, 108–113
  - elements, accessing, 110
- App Connect extensibility model, 193–200
  - creating app with, 194–200
  - multiple extension categories with, 199–200
  - testing, 199
- appdata scheme (DataContext constructor), 430
- AppExtra element, 511
- App Instant Answer extensibility model, 200–202
  - bing\_query parameter (NavigationContext.QueryString), 201
  - SimpleAppInstantAnswer solution, 201
  - testing, 202
- Apptems viewmodel, 861
- Application\_Activated event in trial mode, 826
- Application Analysis (Store Test Kit), 440
- AppBar class, 108
- ApplicationBarIconButton class, 108
- ApplicationBarMenuItem class, 108
- AppBar property, 108
- Application class, 523
  - XAML resources, defining in, 97
- Application\_Closing event handler, 523
- ApplicationCurrentMemoryUsage property, 466, 491
- Application Details (Store Test Kit), 440
- application enrollment token (AET), 850
- applicationhost.config, 318
- ApplicationIcon.png, 26
- ApplicationId attribute, 814
- ApplicationIdleDetectionMode, 47
- Application\_Launching event, 826
- ApplicationMemoryUsageLimit property, 466, 491
- ApplicationPeakMemoryUsage property, 491
- application screenshot 720p, 441
- application screenshot WVGA, 441
- application screenshot WXGA, 441
- Application Single-Threaded Apartment (ASTA), 947
- app lifecycle, 34–47
  - Application.Terminate event, 42
  - Closing event, 35
  - Continuous Background Execution (CBE)
    - feature, 34
  - Deactivated event, 35
  - expected behavior during events, 36
  - Fast app resume, 35
  - Obscured events, 45–47
  - resume policy, 43–45
  - tombstoning, 35, 38–43
  - Unobscured events, 45–47
- app management, 287
- app manifest
  - DeviceLockImageURI element in, 383
  - ExtendedTask element, 392
  - handler registration in, 73–74
  - lock screen BGA requirements, 378
  - Photos Hub extensions and, 651
  - setting to check for required hardware, 258
- app models
  - CoreApplication, 8
  - in Windows 8 vs. Phone 8, 903
  - XAML, 8

- AppointmentMakerControl, 615
  - creating new appointments, 618
- AppointmentMaker solution, 614
- AppointmentStatus enumeration, 618
- App Performance Considerations for Windows Phone (MSDN), 503
- App Responsiveness graph (app analysis report), 497
- apps
  - add/remove accounts in Wallet from, 797–806
  - advertising in, 811–822
  - AllowReadStreamBuffering property, 207
  - ASP.NET, 325
  - AssemblyName, 457
  - automated testing, 452
  - beta testing, 458
  - building/delivering, 18–21
  - building for Windows 8 vs. Phone, 903
  - category, 446
  - category change, 444
  - certification, 439
  - Certification Notes, 452
  - certification requirements, 446
  - clean-and-rebuild, 455
  - compatibility issues from Phone 7 to 8, 509–538. *See also* Windows Phone 7.x apps
  - constant ad location in, 817–819
  - content/experience approach to, 6
  - control, 442
  - debugging, 467–471
  - deep links, 762
  - delivery methods for, 20–21
  - DemoDataSense, 313
  - DemoMobileService cloud, 346
  - deploying through Company Apps feature, 855
  - description, 448
    - limits, 449
  - description change, 444
  - developer tools, 18–20
  - Direct3D app type, 10
  - examining state, 467
  - extensibility points, 468
  - file paths for, 760
    - as file type/URI handlers, 73–76
  - forward navigation support, 48
  - game rating, 444
  - GBAs, limits on, 364
  - hot path, 502
  - icon, 450
  - in-app purchase (IAP), 829–843
  - install folder permissions, 404
  - installing, 20–21
  - key quality metrics, 495
  - languages, 449
  - launching with native code, IV–XXVI
  - launching with NFC tags, 702
  - lifecycle of, 34–47
  - Line of Business (LoB), 847
  - manifest file, 441
  - market distribution, 446
  - Mixed Mode app type, 10
  - Mobile Services, 345
  - monetizing, 811–844
  - multi-targeting, 459
  - native, 466
  - navigation, 52–53
    - cancelling, 59–60
  - network connections, 481
  - NwindODataClient, 326
  - offensive content, 446
  - Page model of design, 47–51
  - passing files to, 75
  - performance, 34–36, 487
  - photo apps picker, 651
  - Photos Hub and, 650–655
  - pivot, 273
  - platform security, 846
  - platform-specific projects for, 528–536. *See also* specific headings
  - pricing, 446
  - profiling, 487–503
  - publication
    - Debug build, 442
    - Release build, 442
    - Store Test Kit, 439
  - publish automatically, 447
  - publishing details, 446
  - push notification clients and, 566–568
  - required artwork, 450
  - resume policy, 43–45
  - screenshot change, 444
  - selective targeting of, 460–466
  - splash screens, 27
  - starting, based on file/URI, 72–73
  - state, 53–56
  - static validation, 452
  - structure of, with Direct3D, 940–954
  - subcategory, 446
  - submission
    - app alias, 446
    - category, 446

## apps pivot (Photos Hub)

- pricing, 446
  - subcategory, 446
- submission tool, 446
- targeting based on device capabilities, 460–462
- TechnicalException, 451
- testing, 472–487
- TestMobileServices, 346
- Title attribute of, 201
- title change, 444
- trial mode, 823–829
- TwitterWriter, 304
- types supported by Windows 8, 902
- unhandled exceptions, 453
- updates, sending, 454–458
- updating Wallet Deals in, 795–796
- Upload page, 451
- user interface and, 6–7
- versioning, 459
- Wallet, unlinking cards from, 805
- Win32/COM APIs, calling from, 8
- WMAppManifest.xml, 25
- XAML app type, 10
- XNA app type, 10
- apps pivot (Photos Hub), 650–651
- app state
  - categories of, 52
  - IsApplicationInstancePreserved property, 55
  - LifecycleState solution, 53
  - PhoneApplicationService.State property, 53
  - storage limit on, 59
- App.xaml, 26
  - RunningInBackground event, 744
- App.xaml.cs, 26
- architecture, 8–18
  - apps, types of, 9–10
  - background processing, 11–12
  - of location/maps, 707–708
  - platform stack, 8–9
  - requisite block diagram of, 8
  - security, 13–16
  - Windows vs. Windows Phone, 16–18
- AreTilesEnhanced property, 514
- ArrangeOverride methods, 90–92
- artwork, updating, 454
- ASP.NET app, 325
- AssemblyName, 457
- Assert class, 474
- AssertFailedException, 478
- Assets folder
  - ApplicationIcon.png, 26
  - Tiles subfolder, 26
- associations (file type/URI), 72–75. *See also* URIs
- Association attribute, 420
- asynchronous callback, 277
- asynchronous call pattern, 278
- asynchronous code
  - in C++, 888–896
  - cancelling operations, 892–896
  - consuming with C++, 896–898
  - progress updates, providing, 890–892
- asynchronous methods, 323
  - BeginGetResponse, 276
  - GetAsync, 289
- Async Pack, 278–280
  - alternative, 280
- AsyncStatus, 859
- AsyncStatus.Completed, 859, 861
- AsyncStatus.Error, 861
- attached properties, 104–107
  - CustomPanel\_AttachableProperty solution, 104
- attributes
  - TestClass, 474
  - TestMethod, 474
- audio
  - agent, 392–394
  - application, 388–391
  - background agent, 384–394
  - handled by Windows 8 vs. Phone, 910–932
  - implementing, 388
  - player state changes, checking for, 390
  - Universal Volume Control (UVC), 12
- audio input
  - DecibelMeter solution, 215
  - DynamicSoundEffectInstance class, 223–227
  - manipulation of, 211–227
  - methods for retrieving, 214
  - microphone, 214–223
  - NAudio library, 223
  - persisting recorded sound, 222
  - phone calls, behavior during, 215
  - required references for, 215
  - SoundEffect class, 211–214
  - SoundEffectInstance class, 211–214
  - SoundFx\_Persist solution, 222
  - SoundFx solution, 218
  - XNA Microphone class, 214–223
- AudioReverb CLSID, 899
- AudioVideoCapture class, 649

AudioVideoCaptureDevice class, 644  
 AudioVolumeMeter CLSID, 899  
 augmented reality, 655  
 automated testing, 452  
 Automated Tests (Store Test Kit), 440  
 automatic resizing, 515  
 AutoPlay property (MediaElement class), 204  
 AutoRotationPreferences property, XIV  
 auto-scaling (screen resolution), 78  
 auto-upload apps, 662–664  
 Averaging (AccelerometerHelper class), 246  
 "awaitable" methods, 278  
 await keyword, 403  
     behavior of, 406  
 await mechanism, 279

## B

Back button  
     backwards navigation and, 48  
     Closing events and, 37  
     Direct3D games and, XXII–XXVI  
     home buttons and, 555–556  
     multiple apps and, 48  
     native code and, XXII–XXVI  
     normal termination and, 37  
     page constructors and, 53  
     pinned tiles and, 554  
     tombstoning and, 38  
 background  
     location tracking, continuous, 12  
     OS services, 11  
     tasks, scheduled, 12  
 BackgroundAgentDemo solution, 366  
 background agents, 349–394  
     alarms/reminders as, 350–357  
     audio, 384–394  
     BTS, 358–362  
     generic, 362–384  
     lock screen, modifying with, 378–384  
     profiling support, 502  
     tasks, 349–350  
     testing, 369–371  
     tiles, updating with, 373–377  
     Wallet, 806–809  
 BackgroundAudioPlayer class, 384–394  
 BackgroundColor property (SystemTray), 111  
 BackgroundCreation, 506  
 BackgroundExecution element (WMAAppManifest.xml), 743  
 background processes, 11–12  
     audio, 11–12  
     auto-uploading apps and, 664  
     in Windows 8 vs. Phone 8, 909–911  
     Music & Video app, 11  
 background services  
     Background Transfer Service, 11  
     Continuous Background Execution (CBE), 12  
 background task control panel, 741  
 background tasks, 349–350  
 BackgroundTransferDemo app, 358  
 BackgroundTransferRequest API, 358  
 background transfers  
     handled by Windows 8 vs. Phone, 909  
 Background Transfer Service, 11  
 Background Transfer Service (BTS), 358–362  
     TransferPreferences property, 358  
 BackKeyPress event, 120  
     native code and, XXII–XXVI  
 backslash (/), escaping special characters with, 146  
 backstack  
     Back button and, 61  
     CBEs and, 746  
     deep links and, 762–763  
     Launchers/Choosers and, 187  
     limits on number of apps in, 40  
     management, 60, 60–63  
     OnRemovedFromJournal virtual method, 60  
     RemoveBackEntry method, 60  
     resume policy and, 43–45  
 BackStack property (NavigationService class), 60  
 backward compatibility, 520  
     testing, 521–538  
 Baer, Matthias, 785  
 Bank Identification Number (BIN), 806  
 BapApp solution, 386  
 bar code readers, 789  
 base class library (BCL), 274  
 BasedOn attribute (Style resources), 100  
 Basic authentication, 339  
 basicHttpBinding, 322  
 BasicTimer.h, 942  
 battery  
     accelerometer and, 239  
     GPS radio and, 748  
 BCL. *See* base class library

## BeginGetResponse

- BeginGetResponse, 276
  - asynchronous method, 276
- behavior
  - Slider control, 527
  - validating, 482
- Belfiore, Joe, 209
- best practices, 503–507
  - locations, 748–749
  - UI thread and, 504–505
  - user engagement, 504
  - visual tree and, 505
- beta release, 841
- beta testing, 458
- “beta-to-release” upgrade path, 458
- big-endian ordering, 679
- binary compatibility, 913
- binding, 322
  - basicHttpBinding, 322
  - data, 322
- <binding> element, 318
- {Binding} syntax, 142
  - SetBinding method vs., 144
- BindingValidationErrorEvent ({Binding} syntax), 166
  - Validation.Errors collection, 166
- BindingValidation\_Info solution, 169
- BindingValidation solution, 165
- BindToShellTile API, 567
- BindToShellToast API, 567
- BingMapsDirectionsTask (Bing Maps launcher), 188, 720
  - target location, setting, 721
- BingMapsLocation solution, 714
- Bing Maps service, 312, 323
  - account requirements for, 712
  - Bing Map control, 712–716
  - Bing Maps web service, 716–720
  - deals and, 787
  - developer accounts, 712
  - launchers, 720–721
  - Phone 7 implementation of, 712–721
  - pushpins, 714–716
  - REST implementation of, 719–720
  - SimpleBingMaps solution, 713
  - SOAP services, list of, 716–720
  - TestGeocodeService solution, 716
  - as web service, 716–720
  - ZoomLevel, 714
- BingMapsTask (Bing Maps launcher), 188, 720
- Bing\_Products\_Gourmet\_Food\_and\_Chocolate Bing extension, 195
- BingPushpins solution, 714
- bing\_query parameter (NavigationContextQuery String), 201
- BitmapImage
  - in HttpRequest, 277
  - in WebClient, 277
- BitmapSource, 506
- blocking page constructors, 504
- Bluetooth
  - accessories, 451
  - devices, connecting to, 695–696
  - finding peers through, 683–686
  - handled by Windows 8 vs. Phone, 906
  - PeerFinder class and, 681
- Boolean properties, 514
  - IsCurrentVersionEnabledForEnhancedTiles, 519
- BoolToStringConverter, 462
- bool values, 462
- Boost (C++ library), 869
- bottlenecks on performance, 487
- Bounds property, XVI
- browsers
  - general-purpose, 284
  - Internet Explorer Mobile 10, 281
  - in Windows Phone 8, 281
  - links, 282
  - scroll manipulations, 282
  - security for, 846
  - zoom manipulations, 282
- Brush control, 95
- BTS. *See* Background Transfer Service
- BufferDuration property (Microphone), 214
- BufferDuration property (Microphone class), 214
- buffering audio input, 216
- BufferingChanged event (MediaElement class), 207
- BufferNeeded event (DynamicSoundEffectInstance), 223–227
- BuildLocalizedApplicationBar method, 109
- BuildMDILXap.ps1 Windows PowerShell script, 853
- builds
  - Debug, 442
  - Release, 442
  - Release-build XAP, 439
- Button control
  - Run Slow Method, 495
  - visual tree and, 82
- ByteOrder property (DataReader class), 680
- ByteOrder property (DataWriter class), 680

## C

## C#

- 4.0 vs. 5.0, 278
- compiler, 516
- lambda functions in, 873
- Windows Runtime components and, 885–887

## C++, 869–878

- asynchronous code in, 888–896
- C++ Component Extensions (C++/CX), 875–878
- consuming asynchronous code with, 896–898
- enums, 874–875
- foreach loops, 872
- lambda functions, 872–874
- moving synchronous code to background, 889–890
- shared pointers, 870
- smart pointers in, 869–872
- type inference, 871–872
- Unique pointers, 870
- Windows Runtime API and, 878–888

caching data resources, 95

calendar, 614–619

- AppointmentMakerControl, 615
- AppointmentMaker solution, 614
- AppointmentStatus enumeration, 618
- new appointments, creating, 618–619
- People Hub information available to, 617
- querying, 614–617
- required references for, 616
- SaveAppointmentTask, 618
- standard vs. custom reminders, 354

Caller/Callee view, 503

calls

- GetQotd, 315
- WebClient.DownloadStringAsync, 300

camera, 621–666

- button (hardware), 637
- emulating in lenses, 661
- extensibility, 656
- focusing on specific points, 637–639
- in emulator, 487
- launching directly into, 624
- lenses, 655–661
- MediaLibrary API, 624–633
- orientation and, 636
- PhotoCamera object, 634–639
- PhotoCaptureDevice class, 639–649
- photos, capturing, 633–649
- Photos Hub, 649–655

- preparing, 640–643
- preview feed, displaying, 635
- Windows Phone 7.1 and, 636

CameraButtons class, 637

CameraCaptureFrame, 642

CameraCaptureTask, 189

CameraChooserTask, 621–624

Cancel method (IAsyncOperation interface), 893

Cancel property (NavigatingCancelEventArgs), 59

CanExecuteChanged event (ICommand class), 154–155

CanExecute method (ICommand class), 154–155

Canvas control (Panel class), 84, 88

capabilities

- declaring, 460
- detection tool, 282
- ID\_CAP\_NETWORKING, 282
- ID\_CAP\_WEBBROWSERCOMPONENT, 282
- optional, 460, 461

Capabilities Detection tool, 282

Capabilities (WMAppManifest.xml), 13–16

- ID\_CAP\_LOCATION, 711

capability tests, 462

CaptureImageAvailable event (PhotoCamera class), 635

CapturePhoto method, 648

CaptureSource class (System.Windows.Media namespace), 633, 649

CAs. *See* certificate authorities

CBE (Continuous Background Execution), 741–747

- non-location tracking apps and, 747

- RunningInBackground event, 742

- SimpleCbe\_MultiPage solution, 746

- SimpleCbe solution, 742

C++ Common Language Infrastructure (C++/CLI), 881

C++ Component Extensions (C++/CX), 875–878

CD3D11\_BUFFER\_DESC object, 945

CD3D11\_TEXTURE2D\_DESC object, 945

CD3D11\_VIEWPORT object, 945

certificate authorities (CAs), 339

certification

- apps, 439
- requirements for, 442, 446

Certification Notes, 452

ChangeConflictException (System.Data.DataCon-  
text), 433

CheckAccess method (Dispatcher class), 583

CheckForResetNavigation method, 746

## CheckSpecializedType method (NdefRecord class)

- CheckSpecializedType method (NdefRecord class), 701
- child windows, 113
- ChildWindow (Silverlight Toolkit), 114, 117–118
  - backstack and, 63
- Choosers, 187–192, 442
  - contacts, single-contact, 590
  - listed, 189
  - usage, 192
- CityFactBook\_LinkedFiles solution, 920
- CityFactBook\_Partial solution, 923
- CityFactBook\_PCL solution, 915–916
- CityFactBook solution, 914
- CityFactBook\_UserControls solution, 927
- CityFactBook\_WindowsRuntime solution, 925
- CivicAddressResolver class, 712
- class IDs (CLSIDs), 899
- clean-and-rebuild, 455
- ClearBackStackAfterReset method, 746
- ClearRenderTargetView function (Direct3D), XII
- ClearRenderTargetView (ID3D11DeviceContext), 951
- Click handlers, 376
  - ICommand vs., 154
  - pinning tiles and, 553
- click-through rate (CTR), 820
- client-side API, 346
- Clipboard API, 230–231
  - TestClipboard application, 230
- Closing events, 35
  - data caching and, 336
  - IsolatedStorageSettings.ApplicationSettings collection, 55
  - MessageBox and, 523
- CloudQotdService, 341
- cloud services, 340
- CLR. *See* Common Language Runtime
- CLSID\_FreeThreadedXMLHTTP60 CLSID, 899
- CLSID\_MFMediaEngineClassFactory CLSID, 899
- CoCreateInstanceEx COM API, 899
- CoCreateInstanceFromApp function, 899
- code paths, alternative, 461
- code sharing
  - between Windows 8/Phone 8, 911–932
  - conditional compilation, 919–922
  - cost/benefits of, 911–913
  - linked files, 919–922
  - managed code, 922–924
  - MVVM and, 913–914
  - partial classes, 922–924
  - PCL project and, 914–919
  - Phone UI as snapped view, 931–932
  - UserControl, 930–931
  - user controls, 927–932
  - XAML-based UI, 927–932
- CollectionBinding\_DTD solution, 173
- CollectionBinding\_Resource solution, 157
- CollectionBinding solution, 147
- CollectionBinding\_XAML solution, 171
- Collection property (NotificationEventArgs parameter), 568
- collections, 147–161
  - CollectionBinding\_Resource solution, 157
  - CollectionBinding\_XAML solution, 171
  - data binding and large collections, 149
  - DynamicCollectionBinding solution, 150
  - dynamic, data binding for, 150–152
  - GroupBinding solution, 158
  - grouping, 158–161
  - sorting, 158–161
- collection types, 320
- CollectionViewSource class, 158
  - View type, converting to LongListSelector.ItemsSource, 183
- color properties (SystemTray), 111
- command binding, 153–157
  - TestCommandBinding solution, 153
- Command={Binding} syntax, 156
- Command element (VCD schema), 758
- CommandPrefix element (VCD schema), 758
- CommandSet element (VCD schema), 758
  - PhraseLists and, 764
  - size limit on, 767
- Common Language Runtime (CLR), 274, 522
- Community Technology Preview (CTP), 442
- COM objects. *See* Component Object Model objects
- Company Apps feature, 855
  - implementing, 856–863
  - managed phones and, 847–849
  - unmanaged phones and, 849–854
- company hub app, building, 856–863
- Compass class, 256–261
  - HeadingAccuracy property, 260
  - IsSupported method, 487
- compass (sensor), 256–261
  - calibrating, 237, 260–261
  - checking for readings, 259
  - in emulator, 487
  - SimpleCompass solution, 257

- compatibility, 913
  - backward, 520
  - breaks, 521
- compiled shader object (CSO) files, 934
- compilers, 516
  - partial classes and, 923
- Completed event, 859, 860
  - TPL and, 280
- Component Object Model (COM), 899–900
  - custom libraries, using, 900
  - Direct3D, 946–949
  - DirectX and, 946–949
  - hybrid SDK and, 522
  - Windows Runtime platform and, 18
- composition thread frame rate, 489
- conditional compilation, 919–922
- configuration axes
  - network speed, 481
  - signal strength, 481
- ConfigurationComplete event handler, 676
- ConnectAsync method
  - .NET sockets and, 676
  - NFC and, 685
- ConnectionCost, 313
- ConnectionReceived event, 671
- ConnectionRequested event, 683
  - NFC and, 685
- ConnectionSettingsTask, 188
- connectivity, 312
  - web, 273–314
- constraints
  - adapt code to accommodate, 461
  - asynchronous call pattern, 278
  - HttpRequest, 278
  - WebClient, 278
- Consumer Key (Twitter API), 302
  - obscuring, 306
- Consumer Secret (Twitter API), 302, 307
- contact card user interface (UI), 588
- Contact class
  - Accounts property, 597
- ContactQueryResult class, 610
  - DeleteContactAsync method, 612
  - EditContactPage method, 611
  - GetExtendedPropertiesAsync, 610
- contacts, 587–613
  - AddContactInfo solution, 602
  - adding, 602–604
  - checking for stale data, 601
  - contact card user interface (UI), 588
  - CustomContactStore solution, 605, 608, 611
  - custom store, creating, 605–613
  - custom stores, displaying, 608–613
  - GetPropertiesAsync, 610
  - LINQ, querying with, 600–601
  - People Hub, understanding, 587–590
  - querying on device, 590–601
  - querying programmatically, 594–620
  - query return format, 597
  - revisions, records of, 613
  - SaveContactTask, 602
  - SaveEmailAddressTask, 603
  - SavePhoneNumberTask, 603
  - SearchAsync method, 595
  - single-contact choosers, 590–594
  - SingleContactChoosers solution, 590, 593
  - StoredContact class, 606
  - syncing with back-end service, 612
  - theme-aware UserControls, 598
  - troubleshooting, 601
  - UserData APIs, querying with, 594
  - Windows.Phone.UserInformation
    - namespace, 605
  - ContactsPopulator solution, 592
  - ContactsQuery solution, 595
    - theme, matching for visibility, 598
  - ContactStoreApplicationAccessMode, 605
  - ContactStore class, 605
    - FindContactByRemoteldAsync method, 612
  - ContactStoreSystemAccessMode, 605
  - ContentIdentifier, 458
  - ContentPanel\_BindingValidationError method, 169
  - ContentPanel grid, 816
  - Content property, 474
  - Continuation object, 336
  - Continuation property and data paging, 332
  - Continuous Background Execution (CBE), 12, 741–747
    - app lifecycle and, 34
  - ContinuousLocation solution, 724
  - Cortana solution, 796, 807
  - controls
    - auto-scaling and, 78
    - Mouse events, 132
    - pivot, 507
    - ProgressBar, 504
    - ProgressIndicator, 504
    - re-templating, 92–94
    - testing, 442
    - TextBlock, 326

## Controls property (MediaPlayerLauncher)

- TextBox, 472
- UserControls vs. custom, 89–92
- Controls property (MediaPlayerLauncher), 204
- ConversionTests, 475
- ConvertAlternatingCharsToUpper method (WindowsRuntimeStringManipulator), 884
- ConvertAlternatingUtf8CharactersToUpper function (WindowsRuntimeStringManipulator), 884
- ConvertBack method (IValueConverter class), 162
- Convert method (IValueConverter class), 162
- ConvertTemperatureAndUpdate method, 472
- ConvertViewportPointToGeoCoordinate type (Map controls), 734
- CopySharedFileAsync method (SharedStorage AccessManager class), 75
- Copy To Output Directory property (audio file), 389
- CoreApplication app model, 8
  - Direct3D for UI, 8
- CoreApplication objects
  - Application Single-Threaded Apartment (ASTA) and, 947
  - Direct3D apps and, 940–941
  - handled by Windows 8 vs. Phone, 907
- Core Common Language Runtime (Core CLR), 17
  - .NET Compact Framework vs., 520
- Core System
  - Mobile Core and, 16–17
  - NT File System (NTFS), 16
  - NT kernel, 16
- CoreWindow class
  - Direct3D apps and, 940–941
  - handled by Windows 8 vs. Phone, 907
- cost per thousand (CPM), 811
- COUNT query, 299
- CountryOrRegion property (AdControl), 815
- CouponCutter solution, 788
- CPM (cost per thousand), 811
- CPU usage, 497
  - performance data, 493
- crash (app)
  - memory usage and, 463
  - unhandled exceptions and, 453
- create\_async method and native code, 889–890
- CreateBuffer (ID3D11Device), 945
- CreateCaptureSequence method, 642
- CreateDepthStencilView (ID3D11Device), 945
- CreateDeviceResources (CubeRenderer), XX
- CreateDeviceResources method (Direct3DBase), 945
  - device objects, creating/caching, 948–949
  - theme color, querying, XII
- CreateFileAsync method, 405
- CreateFX function, 899
- CreateInputLayout (ID3D11Device), 945
- Create method (HttpWebRequest), 276
- CreateOptions property, 506
- CreateOrOpenAsync (ContactStore), 605
- CreatePixelShader (ID3D11Device), 945
- CreateSwapChainForCoreWindow method, 948
- CreateTestPage method, 474
- CreateVertexShader (ID3D11Device), 945
- CreationCollisionOption enumeration, 405
- CredentialsProvider, 713
- cross-targeting, 557–558
- CSS stylesheet
  - Visual Studio, 284
- CTP. *See* Community Technology Preview
- CubeRenderer class, XX
- CubeRender.h/cpp, 942
  - m\_constantBufferData member, 950
- CurrencyAmount InputScope, 135
- CurrentApp class
  - consumable content and, 841
  - durable content and, 837
- CurrentAppSimulator class, 832
- CurrentValueChanged event (SensorBase<T> base class), 237
  - filtering, 243–245
  - observable collections, storing in, 243–245
  - ReadingChanged events vs., 248
- CustomContactStore solution, 605, 608, 611
- customErrors element, 343
- CustomMessageBox class (Windows Phone Toolkit), 114, 118
- custom methods, 297
  - Facebook Query Language (FQL), 297
  - REST, 297
- CustomPanel\_AttachableProperty solution, 104
- CustomPanel solution, 89
- CustomProperties dictionary (WalletItem base class), 790, 791
- CustomWalletProperty objects, 791
- CycleTileData (ShellTileData derived class), 541, 549
- cycle tiles, 541
  - CycleTileData object, 541, 549

## D

- D3D11CreateDevice, 945
- D3D11\_PRIMITIVE\_TOPOLOGY enum, 952
- D3DInputApp solution, 959

- databases. *See also* specific headings
  - LINQ to SQL, 411–434
  - SQLite, 434–436
  - SQL Server Compact 3.5 database, 427
- DatabaseSchemaUpdater class, 424
- DatabaseSchemaVersion property, 426
- data binding, 139–186
  - {Binding} syntax, 142
  - CollectionBinding\_DTD solution, 173
  - CollectionBinding\_Resource solution, 157
  - CollectionBinding solution, 147
  - collections, 147–161
  - command binding, 153–157
  - Databound Application template (Visual Studio), 175–180
  - ElementBinding solution, 164
  - elements, 164–165
  - goals of, 139–140
  - ICommand interface, 153–157
  - INotifyPropertyChanged, 139–147
  - IValueConverter interface and, 598
  - large collections and, 149
  - Model-View-ViewModel (MVVM) pattern, 174–175
  - Resources section (.xaml files), 157
  - separation of concerns, 171–185
  - SimpleDataBinding\_Format solution, 145
  - SimpleDataBinding solution, 140
  - Style resources and, 103
  - templates as resources, 157
  - type/value converters, 161–163
  - validation, 165–170
- Databound Application template (Visual Studio), 175–180
  - improving, 184–185
- DataBoundApp\_modified solution, 184
- DataBoundApp solution, 175
- DataContext constructor, 528
  - in Phone 8 vs. Phone 7, 528
- DataContext instance, 528
- DataContext property (FrameworkElement), 142
- DataContractJsonSerializer, 338, 375, 400, 569
- DataContractSerializer
  - IsolatedStorageFile, 400
  - IsolatedStorageSettings class and, 398
- DataContract XML serializer, 375
- DataPackage type (DataRequested event), II
- Data protection, 846
- DataReader object (Streams namespace), 671
- DataRequested event (DataTransferManager class), II
- data resources, 94–95
- Data Sense API, 312–314
- DataServiceCollection<T> class
  - vs. DataServiceQuery<T>, 331
  - Windows Communications Foundation (WCF) Data Services, 161
- DataServiceContext object, 328, 334
- DataServiceQuery<T>
  - filtered queries and, 330
  - OData clients and, 328
  - vs. DataServiceCollection<T>, 331
- DataServiceState object, 334
- DataSvcUtil tool, 326, 337
- DataTransferManager class, II
- data usage, 907
- data validation, 165–170
  - BindingValidationError event, 166
  - BindingValidation\_Info solution, 169
  - BindingValidation solution, 165
  - ContentPanel\_BindingValidationError method, 169
  - INotifyDataErrorInfo, 169
  - PhoneApplicationPage\_BindingValidationError handler, 169
- DataWriter object (Streams namespace), 671
- DatePicker (Windows Phone Toolkit), 356
- DDLs. *See* Dynamic-Link Libraries
- DeactivatedEventArgs (Reason property), 745
- Deactivated events, 35
  - IsolatedStorageSettings.ApplicationSettings collection, 55
  - MessageBoxes and, 523
  - paged data and, 336
  - PhoneApplicationService.State collection and, 55
  - RunningInBackground event vs., 744
- Deal class (WalletItem), 787, 790–791
- deals, managing, 787–796
- Debug build, 442
- debugging, 467–471
  - device configurations, targeting, 467–468
  - with Fiddler, 468–471
  - mixed-mode projects, 888
  - pinned apps, 557
  - startup, 468
  - TestLifecycle solution, 42
- Debug Location toolbar (Visual Studio), 373
- Debug tab (Visual Studio)
  - Tombstone Upon Deactivation While Debugging setting, 43
- Debug.WriteLine statements, 42

## DecibelMeter solution

- DecibelMeter solution, 215
- DecodePixelHeight property (BitmapSource), 506
- DecodePixelWidth property (BitmapSource), 506
- decoupling data, 171–174
- deep-linking content, 702–705
- deep links, 762–763
- default constructor, 528
- DefaultTask object, 44
- deferred loading, 422
- DeleteContactAsync method (ContactQueryResult class), 612
- DeleteOnSubmit method (System.Data.DataContext), 417
- DELETE operation, 323
- DemoDataSense app, 313
- DemoMobileService cloud app, 346
- DemoWebRequests solution, 273
- Dependency Injection (DI), 174–175
- DependencyObject class, 101
- DependencyProperty, 101–104
  - data binding and, 142
- DependencyProps sample, 101
- deploying apps
  - to cloud, 340
  - Company Apps feature and, 855
- deployment
  - production, 344
  - staging, 344
- depth buffering, 934
- depth stencil buffer, 934
- description (Windows Phone Store), 448
  - limits on, 449
  - update, 454
- design principles, 4–6
  - content-centered, 5
  - motion, 5
  - simplicity, 5
  - skeuomorphic elements, lack of, 5
  - typography, 5
- DesiredAccuracy property (GeoCoordinateWatcher class), 709
  - Geolocator class and, 722
- Detailed Analysis view, 498
- Dev Center, 452–454
- developer accounts (Bing Maps), 712
- developer tools, 18–20
  - Phone 7/8, building support for, 20
  - Windows Phone 8 SDK, 18
  - Windows Phone emulator, 19–20
- DeviceAcceleration property (SensorReading), 270
- DeviceExtendedProperties class, 491
  - vs. DeviceStatus, 491
- DeviceLockImageURI element (app manifest), 383
- device management client, 847
- DeviceNetworkInformation.IsCellularDataEnabled, 462
- DeviceOrientationHelper class, 245
  - updating with CurrentValueChanged events, 251
  - usage, 249–252
- DeviceRotationRate (Motion class), 269
- devices
  - memory limits, targeting apps based on, 463
  - targeting apps based on capabilities of, 460–462
  - testing on, vs. emulator, 487
- device state
  - idle timeouts, disabling, XVI–XXVI
  - integrating with native code, X–XXVI
  - screen orientation, XIII–XXVI
  - themes, integrating with native code, XI–XXVI
  - user default language and, X–XXVI
- DeviceStatus API
  - background agents and, 502
  - memory and, 466
  - Memory APIs in, 491
  - vs. DeviceExtendedProperties, 491
- DeviceStatus.KeyboardDeployedChanged event, 135
- DeviceUniqueId property, 524
- diagnostic analysis
  - capture, 478
  - frame rate issues, 495
  - responsiveness issues, 495
  - share, 478
- dictation grammar, 771–772
- Direct2D, 935, 961–964
- Direct3D, 933–964
  - apps, structure of, 940–954
  - COM and, 946–949
  - device state and, X–XXVI
  - Direct2D and, 961–964
  - DirectXTK and, 961–964
  - initializing, 945
  - manipulating camera preview buffer with, 643–647
  - profiling, 502
  - rendering, 950–954
  - Update method, 950–954
  - Visual Studio project types and, 936–937
  - Windows 8 vs. Phone 8, 909
  - on Windows Phone, 935–936
  - XAML projects and, 937–939

- Direct3D apps, 940–954
  - call sequence of, 943–944
  - CoreApplication class and, 940–941
  - CoreApplication class and, 940–941
  - handled by Windows 8 vs. Phone, 906
  - initializing, 945
  - native code and, 941–944
  - theme colors and, XI–XXVI
  - user input, handling, XVII
  - user input in, 959–961
  - web connectivity and, XXIV–XXVI
- Direct3DBase.h/.cpp, 942
- Direct3DInterop class, 644
- Direction dependency property, 101–103
- DirectWrite (Direct3D), 936
- DirectX
  - COM and, 946–949
  - hardware level features, 949–950
- DirectXHelper.h, 942
- DirectXTex library, 936
- DirectXTK\_Desktop\_2012 solution, 963
- DirectXTK library, 936, 961–964, XVIII
- DirectXTK toolkit, 645
- DirectXTK\_WindowsPhone8 project, 962
- DirectX Toolkit (DirectXTK), 961
- disable features, 461
- DiscardView (ID3D11DeviceContext1), 953
- Dismissed events (CustomMessageBox), 118
- Dismissing events (CustomMessageBox), 118
- Dispatcher class (DependencyObject), 582
- DispatcherTimer object, 501
- DisplayAmount property (Wallet), 799
- DisplayAvailableBalance property (Wallet), 799
- DisplayName filter (SearchAsync), 596
- DisplayName property (PeerInformation class), 685
- DisplayName property (WalletItem), 787
- DisplayProperties class ( Windows::Graphics::Display), XVI
- DisplayRequest class (Windows::System::Display), XVI
- Dispose pattern, 371
- DistanceSpeaker solution, 896
- domain network security, 317
- Donald Knuth, 488
- DotNetSocketClient project, 675, 676
- dot notation, 516
- DownloadChanged event (MediaElement class), 207
- DownloadStringAsync method, 274
- DownloadStringCompleted event, 274, 275
- DownloadStringTaskAsync method, 278

- dragging (ManipulationDelta event), 130
- DrawingSurfaceBackgroundGrid-based app, 937
- DrawingSurfaceBackgroundGrid XAML element, 938–939
- DrawingSurface element, 937–939
- DrawString method (SpriteFont), 963
- dual-core processors, 460
- duplicating existing projects, 529
- DynamicCollectionBinding solution, 150
- dynamic-discovery techniques, 461
- Dynamic-Link Libraries (DLLs), 288
  - background processing in Windows Phone and, 909
- dynamic screen layout model, 84
- DynamicSoundEffectInstance class, 223–227
  - BufferNeeded event, 223–227
  - sine wave, 225
  - SoundEffectInstance and, 223
  - TestDynamicSounds\_Controls solution, 225
  - TestDynamicSounds solution, 224

## E

- EbookStore solution, 834, 835, 838
- EditContactPage method (ContactQueryResult class), 611
- EDM. *See* Entity Data Model
- ElementBinding solution, 164
- elements
  - AppExtra, 511
  - <binding>, 318
  - BindingOperations.SetBinding method and, 144
  - customErrors, 343
  - data binding, naming and, 142
- EmailAddressChooserTask, 189
- EmailAddress filter (SearchAsync), 596
- Email apps, III
- EmailComposeTask, 188
  - native code and, V
- embedded WebBrowser control
  - create apps that use, 281
- emphasis element (SSML), 779
- EmptyTypes array, 516
- emulator (Microsoft Visual Studio)
  - accelerometer, testing in, 241
  - camera behavior in, 487
  - changes between Windows Phone 7 and 8, 316
  - compass behavior in, 487
  - constraints, 487

## enable features

- debugging lock screen BGAs in, 382
- forced tombstoning, testing with, 591
- gyroscope behavior in, 487
- Hyper-V Virtual Machine, 487
- images, 467, 537
- keyboard shortcuts, hardware equivalents for, 486–508
- locking, 481
- MicroSD cards behavior in, 487
- multitouch behavior in, 487
- Music and Video Hub apps, testing, 229
- Near Field Communication (NFC) behavior in, 487
- options for, 537
- simulating app update on, 519
- simulating OS upgrade on, 519
- social network integration behavior in, 487
- State Persistence behavior in, 487
- testing web services in, 317–321
- unlocking, 481
- UVC, displaying in, 385
- UVC, manipulating in, 227
- enable features, 461
- EnableRedrawRegions flag, 490
- EndGetResponse method, 483
- endianness, 679
- enrollment client, 847
- enrollment in Company Apps
  - of managed phones, 847–849
  - of unmanaged phones, 850–854
- Enrollment method, 850
- enterprise apps, 845–864
  - Company Apps feature and, 855
  - managed phones, 847–849
  - unmanaged phones, 849–854
- enterprise-focused support, 845
- Entity Data Model (EDM), 325
- EntityRef type, 421
- EntitySet collection, 421
  - IsForeignKey property, 421
- enums (C++), 874–875
- ErrorAdditionalData property (ErrorOccurred event), 577
- error handling
  - data validation errors, 165–170
  - Geolocator and, 722
- ErrorOccurred event, 576–578
- ErrorType property (ErrorOccurred event), 577
- event handlers
  - App.xaml.cs and, 26
  - OnGetLiveData, 290
  - in visual tree, 122
- events, 274
  - Activated, 523
  - Application\_Closing, 523
  - Closing, 336, 523
  - Completed, 860
  - Deactivated, 336, 523
  - DownloadStringCompleted, 274, 275
  - FrameReported, 133–134
  - GetCompleted, 292
  - GetLoginUrl, 296
  - GetQotdCompleted, 323
  - in C++/CX, 875–878
  - KeyDown, 526
  - KeyUp, 472
  - Launching, 523
  - LayoutUpdated, 526
  - LoadCompleted, 310, 332
  - manipulation, 126–132
  - Mouse, 132–133
  - in native code, 875–878
  - Navigated, 310
  - NetworkStatusChanged, 313
  - Obscured, 481
  - OnBackPressed, 523
  - OnNavigatedFrom, 336, 523
  - OpenReadCompleted, 274
  - Progress, 860
  - routed, 121–124
  - ScriptNotify, 286
  - SessionChanged, 289
  - TextInput, 526
  - TextInputStart, 526
  - TextInputUpdate, 526
  - Unobscured, 481
  - user-input, 497
  - WebBrowser.ScriptNotify, 284
- examining app state, 467
- ExampleText property (SpeechRecognizerUI class), 769
- exceptions, unhandled, 478
- Excessive Allocations, 500
- Exchange ActiveSync, 846
- exchange vs. MDM servers, 847
- Execute method (ICommand class), 154–155
- Execution Manager platform service, 9
- execution, profiling, 493
- ExpandingEllipse solution, 490
- ExpirationTime (background agents), 369

- Export Stack Traces, 454
- ExposureCompensation property, 641
- Expression Blend
  - data-binding collections in, 149
  - separation concerns and, 173
- ExtendedTask element, 372, 392
- ExtendedTask entry (lock-screen BGA), 378
- extensibility models (Bing search), 192–202
  - App Connect, 193–200
  - App Instant Answer, 200–202
  - Extras.xml, 195–196
  - requirements for, 193
  - search extensions, registering, 193
  - SimpleAppConnect solution, 199
  - SimplestAppConnect solution, 194
- extensibility points, 7, 468
- Extension element (Wallet Hub), 803
- Extension entry (lock screen BGA), 378
- Extension methods, 516
- ExtensionName attribute (Wallet), 804
- Extensions section (WMApManifest.xml), 195
- External Events, 497
- Extras.xml, 195–196
- extreme programming (XP), 472

## F

- Facebook, 273
  - API documentation, 295
  - code, 294
  - COUNT query, 299
  - DetailsPage, 294
  - friend count, 299
  - Graph API Explorer, 298
  - LoginPage, 294
  - MainPage, 294
  - MediaLibrary and, 625
  - NuGet package, 294
  - OAuth 2.0 protocol, 300
  - pages, 294–314
  - permissions, 295
  - support, 294
  - XAML, 294
- FacebookClient object
  - GetAsync method, 296
  - GetLoginUrl method, 295
  - Graph API and, 298
- Facebook C# SDK, 294–300
- FacebookOAuthResult.IsSuccess property, 296
- Facebook Query Language (FQL), 297
- FallbackValue attribute ({Binding} syntax), 147
- fast app resume, 35
  - IsApplicationInstancePreserved property (Application.Activated), 40
- fast app switching, 37–38
- Feedback element (VCD schema), 759
- FibonacciPrimes solution, 878, 889, 893
- Fiddler
  - configure, 469
  - emulator and, 468–471
  - HTTP debugging tools, 468
  - proxy, 469
  - use with Wi-Fi, 470
- FidMe, 804
- File Association contracts, 906
- FileOpenPicker API, VIII–XXVI
  - handled by Windows 8 vs. Phone, 906
- file type associations, 72–75
  - app activation, handling, 74–75
  - handlers, apps as, 73–76
  - passing files to apps, 75
  - restrictions on, 74
  - starting apps based on, 72–73
- fill rate counter, 489
- FilteredAccelerometer app, 242
- FindAllPeersAsync method, 683–684
  - Bluetooth devices and, 696
- FindContactByRemoteldAsync method (Contact-Store), 612
- FindPackagesForCurrentPublisher, 861, 862
- FindTimeForAppointment (AppointmentMakerControl), 615
- firewalls and push notifications
  - , 561
- flick gesture, 131
- FlipTileData (ShellTileData derived class)
  - Phone 7 apps, lighting up with, 515
  - tile properties, updating, 549
  - updating tiles and, 540
- flip tiles, 539–540
  - FlipTileData class, 549
  - FlipTileData object, 540
  - ShellTile.Update method, 540
  - WideBackContent, 540
- FMRadio class, 524
- FM radio hardware support, 524
- Fong, Jeff, 4
- foreach loops (C++), 872
- foreground agents, 807

## ForegroundColor property (SystemTray)

- ForegroundColor property (SystemTray), 111
- Forms authentication, 339
- FPS. *See* Frames Per Second
- FQL. *See* Facebook Query Language
- Fragment, 65–68
  - NavigationParameters solution, 65
- Frame, 78
- FramePageSizes app sample, 79–81
- Frame Rate, 497
  - diagnosing problems with, 495
  - performance data, 493
- frame rate counters, 488–489
  - composition thread, 489
  - fill, 489
  - intermediate surface, 489
  - surface, 489
  - Texture memory usage, 489
  - UI thread, 489
- FrameReported event, 133–134
- frames (Direct3D), 934
- Frames Per Second (FPS), 497
- FrameworkElement type, 527
  - data-binding and, 142
  - XAML resources, defining in, 97
- FunctionalCapabilities section, 464
- functionality, platform-specific, 532
- FxC.exe, 934
- FXEcho CLSID, 899
- FXEQ CLSID, 899
- FXMasteringLimiter CLSID, 899
- FXReverb CLSID, 899

## G

- GameInviteTask, 189
- games, 903, 933–964
  - initializing, 945
  - phases of, 940
  - rating, 444
  - theme colors and, XI–XXVI
  - Update method, 950–954
  - user input, handling, XVII
- garbage collector, 500
- GBA. *See* generic background agents (GBA)
- GC events, 498
- GC Roots view, 501
- general-purpose browser, 284
- generate
  - memory report, 495
  - proxy code, 327
- GenerateBarCodeImage method (Wallet), 789
- generic background agents (GBA), 362–384
  - components of, 366–373
  - expiry of, 363
  - lock-screen background, updating with, 378–382
  - lock-screen notifications and, 382–384
  - memory limits on, 364
  - prohibited/permitted operations in, 364
  - setting up, 365
  - tiles, updating with, 373–377
- GeocodeQuery API, 734–739
- GeocodeService, configuring, 717
- GeoCoordinateWatcher class, 367, 708–712
  - accuracy setting in constructor for, 709
  - as background agent, 371
  - Geolocator class vs., 722
  - stopping/starting, 710–712
- Geolocator class (Phone 8), 722
  - GetGeopositionAsync method, 722
  - MovementThreshold properties, 725
  - PositionChanged events, 722
  - ReportInterval property, 725
  - SimpleGeoLocator app, 722
- Geoposition<T>, 710
- Get Access Token, 298
- GetActions method (ScheduledActionService), 352
- GetAsync method, 289
- GetAvailablePreviewResolutions method (PhotoCaptureDevice class), 640
- GetChild method (VisualTreeHelper class), 82
- GetCompleted event, 292–293
- GetConstructor method, 515, 517
- GetDefault factory method, 697
- GetExtendedPropertiesAsync (ContactQueryResult class), 610
- GetForCurrentThread (CoreWindow), 940
- GetGeopositionAsync method, 722
  - ContinuousLocation solution, 724
- GetLogicalChildrenByType<T> extension method, 84
- GetLoginUrl event, 296
- GetMethod method, 517
- GetNextTrack method (BackgroundAudioPlayer), 392
- GET operation, 323
- GetParent method (VisualTreeHelper class), 82, 947
- GetPath extension method (PhoneExtensions namespace), 658
- GetPreviousTrack method (BackgroundAudioPlayer), 392
- GetPrimaryTouchPoint method, 134

- GetProductReceiptAsync method (CurrentApp class), 837
  - GetPropertiesAsync (ContactQueryResult class), 610
  - GetQotd, 315
  - GetQotdAsync, 323
  - GetQotdCompleted event, 323
  - GetResourceStream method
    - SoundEffect/SoundEffectInstance classes and, 212
  - GetSampleSizeInBytes (Microphone), 214
  - GetSharedFileName method (SharedStorageAccess-Manager class), 75
  - GetSupportedPropertyRange (PhotoCaptureDevice), 641
  - GetSupportedPropertyValues (PhotoCaptureDevice), 641
  - GetUserPreferredUILanguages function, X
  - GetValue method (DependencyObject), 101
  - GetXXXPoint method, 134
  - GlobalElementChange solution, 83
  - Global Speech Experience. *See* GSE (Global Speech Experience)
  - GoBack method, 282, 284
  - GoForward method, 282, 284
  - GPUs
    - on desktop computers, 949
    - redraw regions and, 490–491
    - shader code and, 934
  - grammars, 770–776
    - disabling, 772
    - predefined, 771–772
    - preloading, 776
    - simple lists for, 772–773
    - SRGS, using, 773–776
  - Graph API, 297
  - graphical manifest editor (Visual Studio), 711
  - gravity, effect of on accelerometer, 238
  - Gravity property (SensorReading), 270
  - grayscale, 631
  - GrayscalePhotoEditor solution, 629
  - Grid app template (Visual Studio), 932
  - Grid control (Panel class), 84, 87
    - Column property, 87
    - Row property, 87
    - StackPanel vs., 505
    - in visual tree, 82
  - GroupBinding solution, 158
  - GroupDescription property (CollectionView Source), 158, 160
  - Groupon, 787
  - GSE (Global Speech Experience), 754–756
    - usage, 754–756
  - GUID, 324
    - file type/URI associations and, 75
  - gyroscope
    - in emulator, 487
    - SimpleGyroscope solution, 262
  - Gyroscope class, 263–265
  - Gyroscope.IsGyroscopeSupported method, 487
  - Gyroscope.IsSupported, 462
  - gyroscope sensor, 261–265
- ## H
- HandleDeviceLost method (Direct3DBase), 954
  - hardware
    - camera, 621–666
      - camera button, 637
      - chassis requirements for, 236–237
      - DirectX and, 949–950
      - events, simulating in emulator, 486
      - native code and, XIII–XXVI
    - hardware buttons, cancelling navigation with, 59
    - hardware keyboard, 135
    - headers, precompiled, 880
      - sharing, 926
      - turning off, 926
    - HeadingAccuracy property (Compass class), 260
    - Heading property (Map control), 729
    - heap summary, 500
    - HelloWorld solution, 961
    - High-Level Shader Language (HLSL), 934
    - high memory, 464
    - HitTest method, XIX
    - HLSL, 934
    - home buttons
      - Back-button behavior and, 555
      - disabling and pinned tiles, 556
      - pinned tiles and, 554
    - HorizontalCenterElement, 527
    - HorizontalThumb, 527
    - HorizontalTrackLargeChangeDecreaseRepeatButton, 527
    - hot path, 502
    - HRESULTS (COM), 946
    - HTC 8X, 482
    - HTML5
      - support, 282
      - <video> tag, 282

## HTML-based app support in Phone 8

HTML-based app support in Phone 8, 281

HTML pages

    invoke script, 284

    receiving data, 284

HTML parsing, 457

HTTP

    activation, 320

    client libraries, 280

    debugging tools, 468

    native code and, XXIV–XXVI

    WebExceptions and, 469

HttpNotificationReceived events, 567

HttpRequest.h/cpp, XXV

HttpRequest class, 273–281

    asynchronous call patterns, 278

    Async Pack, 278–280

    BitmapImage, 277

    BTS vs., 358

    create, 276

    Data Sense API and, 312

    equivalent, 277

    JSON-formatted data and, 337

    REST web services vs., 324

    Task Parallel Library (TPL), 280–281

    vs. WebClient, 273

    Windows Phone Store requests and, 456

    WritableBitmap, 277

Hybrid SDK, 522

HyperlinkButton objects, 63

Hyper-V, 317

Hyper-V Virtual Machine, 317, 487

## I

IANA. *See* Internet Assigned Names Authority

IAP. *See* in-app-purchase (IAP)

IApplicationBar interface, 111

IASetIndexBuffer (ID3D11DeviceContext), 952

IASetPrimitiveTopology, 952

IASetVertexBuffers (ID3D11DeviceContext), 952

IAsyncOperation interface, 889–890

IAsyncOperationWithProgress type (Windows Runtime), 890

IAsyncResult callback, 331

IAudioVideoCaptureDeviceNative interface, 649

ICameraCaptureDeviceNative interface, 643–644

ICollectionView, 347

ICommand interface, 153–157

    TestCommandBinding solution, 153

IconicTileData objects, 518, 543

IconicTileData (ShellTileData derived class), 549

IconicTileData template, 376

iconic tiles, 542

    IconicTileData class, 549

    IconicTileData object, 543

icons

    apps, 450

    for associated files, 73

ID3 v1/v2 metadata support, 205

ID\_CAP\_APPOINTMENTS capability, 616

ID\_CAP\_CAMERA capability, 649

ID\_CAP\_CONTACTS capability, 460, 595

ID\_CAP\_IDENTITY\_USER capability, 813

ID\_CAP\_LOCATION capability, 813

    CBEs and, 743

    routes/directions and, 734

ID\_CAP\_MAP capability, 727

    CBEs and, 743

    routes/directions and, 734

ID\_CAP\_MEDIALIB capability, 625, 813

    photo apps picker and, 653

ID\_CAP\_MEDIALIB\_PHOTO capability, 625, 813

ID\_CAP\_MICROPHONE capability, 215, 649, 768

ID\_CAP\_NETWORKING capability, 282, 813

    predefined grammars and, 771

ID\_CAP\_PHONEDIALER capability, 594, 813

    PhoneCallTask object, 190

ID\_CAP\_PROXIMITY capability, 681

ID\_CAP\_PUSH\_NOTIFICATION capability, 566

ID\_CAP\_SPEECH\_RECOGNITION capability, 768

ID\_CAP\_WALLET capability, 787

ID\_CAP\_WEBBROWSERCOMPONENT capability, 282, 813

ID\_FUNCCAP\_EXTEND\_MEM capability, 464, 466

ID\_HW\_FRONTCAMERA capability, 636

idle timeouts, disabling, XVI–XXVI

Id property (WalletItem), 787

ID\_REQ\_FRONTCAMERA capability, 636

ID\_REQ\_MEMORY\_90 capability, 466

ID\_REQ\_MEMORY\_300 capability, 464, 466

ID\_REQ\_REARCAMERA capability, 636

IEnumerable collection object, 147

IFrameworkView, 940–941

IFrameworkViewSource interface, 942

IIS. *See* Internet Information Services (IIS)

IIS Express, 317, 318

Image control, 127–132

Image loads, 498

- image requirements (Windows Phone Store), 441
  - Store tile, 441
- images
  - auto-scaling and, 78
  - performance and, 506
- immutable interfaces, 905
- implicit styles, 99–101
- in-app purchase (IAP), 829–843
  - consumable content, 840–843
  - content, types of, 831–832
  - durable content, 835–839
  - handled by Windows 8 vs. Phone, 907
  - implementation of, 830–831
  - license checks on durable content, 836
  - MockIAPLib library, 832–835
  - testing, 832
  - validating receipts for, 839
- Information Rights Management (IRM), 846
- Initialized event (PhotoCamera class), 635
- INotifyDataErrorInfo interface, 169
- INotifyPropertyChanged, 139–147, 290, 328
  - dynamic data bound collection and, 150–152
  - propagating changes with, 143–144
- INotifyPropertyChanging interface
  - LINQ-to-SQL memory optimization and, 432
- Input Method Editor (IME), 135
- InputPane class (Windows::UI::ViewManagement), XX
- InputScope property (SIP), 135
- InputToApp function (JavaScript), 284
- InsertOnSubmit method (System.Data.DataContext), 417
- InstallApp method, 862
- InstallationManager.GetPendingPackageInstalls, 859
- InstallationManager type, 862
- InstallCommandSetsFromFileAsync Windows Runtime method, 759
- instance methods, 516
- Intellisense behavior, 531
- inter-app backstack, 47
- interfaces (COM), 946
- intermediate surface counter, 489
- Internet Assigned Names Authority (IANA), 313
- Internet ConnectionProfile, 313
- Internet Explorer Mobile 10, 281
- Internet Information Services (IIS), 340
  - full version, 320
  - mock web server, 835
- Internet Protocol security, 317
- Internet Protocol security (IPsec), 471
- IntervalTraining app, 351
- IPsec. *See* Internet Protocol security
- IPv4, 670
- IQotdService, 321
- IQueryable<T>, 330
- IRM. *See* Information Rights Management
- IsApplicationInstancePreserved property (Application.Activated), 40
  - Activated event handler, 55
- IsAutoCollapseEnabled property (AdControl), 815
- IsAutoRefreshEnabled property (AdControl), 815
- IsCameraTypeSupported method, 636
- IsCancelable property (NavigatingCancelEventArgs), 59
- IsCurrentVersionEnabledForEnhancedTiles Boolean property, 519
- IsDataValid property (SensorBase<T> base class), 259
- ISE (Isolated Storage Explorer), 401
  - IsolatedStore directory, 402
  - tool for, 455
- IsForeignKey property, 421
- IsKeyboardInputEnabled property (CoreApplication), XIX
- IsLocked property (ObscuredEventArgs object), 46
- IsMuted property (MediaElement class), 204
- IsNavigationInitiator property (Navigating CancelEventArgs), 69
- IsNavigationInitiator property (NavigationEventArgs), 69
- isolated storage, 455
  - downloaded files and, 359
  - SoundFx\_Persist solution, 222
- isolated storage APIs, 395–411
  - files, 399–402
  - settings for, 395–398
- Isolated Storage Explorer. *See* ISE (Isolated Storage Explorer)
- IsolatedStorageFile class, 399–402
  - DataContractJsonSerializer, 400
  - DataContractSerializer and, 400
  - Win32 APIs vs., 399
  - XmlSerializer and, 400
- IsolatedStorageFileReaderWriter solution, 399
- isolated storage files, 399–402
- IsolatedStorageSettings class, 395–398
  - format of storage, 398
  - simultaneous access and data corruption, 398
- IsolatedStorageSettingsReaderWriter solution, 396
- IsolatedStore directory, 402
- isostore scheme, 430

## IsScriptEnabled property

- IsScriptEnabled property, 294
- IsShutterSoundEnabledByUser property (Known-CameraGeneralProperties), 641
- IsShutterSoundRequiredForRegion property (KnownCameraGeneralProperties), 641
- IsSupported method (Compass), 487
- IStorageFile interface, 402
- IStorageFolder interface, 402
- IsTrial property (LicenseInformation object), 826
- isUpdatingSliderFromMedia flag (MediaElement class), 210
- ItemsControl elements (data binding), 147
- ItemsSource property (ItemsControl element), 322
  - data binding and, 147
- IValueConverter interface, 598
  - data binding and, 162–163
- IVector type (Windows Runtime), 881
- IXMLHttpRequest2Callback interface, XXV
- IXMLHttpRequest2 (IXHR2) interface, XXIV–XXV

## J

- JavaScript
  - integrating with WebBrowser control, 284–286
  - interoperate bi-directionally, 284
  - support, 846
- JavaScript Object Notation (JSON), 324
  - DataContractJsonSerializer, 400
- JIT. *See* just-in-time compile
- JournalEntry object
  - backstack and, 60
- JPEG format, 630
  - System.Windows.Media.Imaging namespace and, 547
  - tiles and, 546
- JSON. *See* JavaScript Object Notation
- JSON-formatted data, 325
  - data services and, 337–338
  - vs. XML-formatted data, 337–338
- JsonNwndClient solution, 337
- just-in-time (JIT) compiling, 21, 36, 445

## K

- keyboard (hardware) and orientation, 233
- keyboard inputs, 135–137
  - receiving in native-code apps, XVII
- KeyDown event, 526
- key quality metrics, 495

- keys
  - access, 343
  - Consumer, 311
  - HMACSHA1, 311
  - manage, 343
  - “oauth\_token\_secret”, 309
  - “oauth\_token”, 309
- KeyUp event, 472
- Key value (resource), 96–98
  - MergedDictionaries and, 99
- keywords
  - update, 454
  - Windows Phone Store, 449
- KnownCameraGeneralProperties class, 641
- KnownCameraPhotoProperties class, 641
- KnownFolders class (Windows.Storage), 624

## L

- lambda functions (C++), 872–874
- LandmarksEnabled flag (Map controls), 732
- landscape modes (orientation), 234
- Language-Integrated Query (LINQ) framework, 412
- Language-Integrated Query syntax. *See* LINQ (Language-Integrated Query) syntax
- languages
  - apps, 449
  - user default, X–XXVI
- LastExitReason property (background agents), 363
- LastOperation property (SocketAsyncEventArgs class), 676
- LastUpdatedTime property (WalletItem), 787
- Latitude property (AdControl), 815
- LaunchApp method, 862
- LaunchApp tag, 702
- Launcher class (Windows.System namespace), 72
  - enabling Bluetooth with, 684
- launchers, 187–192, 442, 455
  - Bing Maps, 720–721
  - handled by Windows 8 vs. Phone, 907
  - in Windows 8 vs. Phone, 903
  - listed, 188
  - maps, 739–740
  - MediaPlayerLauncher, 203–204
  - usage, 189–190
- LaunchersAndChoosers solution, 189
- LaunchFileAsync method (Launcher class), 72–73
- LaunchForTest (ScheduledActionService), 365, 369, 371

- Launching event handler, 523
- LaunchUriAsync method (Launcher class), 72
  - built-in URI schemes for, VI–XXVI
- LayoutUpdated event, 526
  - state management and, 53
- LayoutUpdated event (Phone 7), 526
- legacy code
  - Win32 APIs and, 407
- lenses, 655–661
  - emulating camera in, 661
  - launching from camera, 656–659
  - rich media editing, 659–661
- Level Starter Kit, 245–252
  - ReadingChanged event, 247
- license checks on durable content, 836
- LicenseInformation class (Microsoft.Phone.Marketplace), 826
- LifecycleState solution, 53
- lighting up (Windows 7.x apps), 509–519
  - OS version number, checking, 512–513
  - tiles, 510–520
  - upgrading tiles from Phone 7.x apps, 513–517
- “light-up” scenario, 459
- limits
  - beta testing, 458
  - on data, 484
  - on garbage collection, 500
  - memory, 463, 500
  - memory usage, 491
  - upgrades, 515
- Line of Business (LoB), 847
- linked files and code sharing, 919–922
- links (browser), 282
- LINQ (Language-Integrated Query) syntax, 330
  - contacts, querying with, 594
  - lambdas in, 872
  - querying contacts with, 600–601
  - SearchAsync vs. (performance), 599
- LINQ-to-SQL, 411–434
  - associations, 418–423
  - batch updates, speeding up, 433–434
  - create items, 416–418
  - database encryption, 431
  - DatabaseSchemaUpdater class, 424
  - databases, pre-populating, 427–431
  - defining the database, 412–416
  - delete items, 416–418
  - EntityRef type, 421
  - EntitySet collection, 421
  - isostore scheme, 430
  - memory costs with, 412
  - memory usage, minimizing, 432
  - OtherKey property, 421
  - performance issues, 431–434
  - phone vs. desktop versions, 423
  - Plain Old CLR Objects (POCOs), 415
  - queries, performing, 416
  - schema changes, handling, 423–427
  - SQL Server Date type and, 423
  - System.Data.DataContext database schema, 412
  - update items, 416–418
- LinqToSQL\_BookManager solution, 412–416
- LinqToSQL\_LendingLibrary solution, 418
- LinqToSQL\_Performance solution, 433
- LinqToSQL\_ReferenceDBConsumer solution, 427
- LinqToSQL\_ReferenceDBCcreator solution, 427
- LinqToSQL\_UpdatedBookManager solution, 424
- ListBox control
  - LongListSelector vs., 506–507
  - performance issues, 506
  - user interface (UI), 276
- ListenFor element (VCD schema), 759
- ListenText property (SpeechRecognizerUI class), 769
- little-endian ordering, 679
- LiveConnectClient object, 289
- LiveConnectSession field, 292
- Live SDK, 287–294
  - connecting to SkyDrive using, 290–294
  - download, 288
- LoadAsync/LoadCompleted model, 331
- LoadCompleted event
  - data caching and, 334, 336
  - HTML content and, 310
  - paging data and, 332
- LoadDeviceListFromFile method, 572
- Loaded event, 53
- LoadNextPartialSetAsync, 332
- load time performance, 95
- LoadViewModel helper, 514
- localhost, 317
- LocalizedStrings.cs, 26
- local storage, 395–411
  - isolated storage APIs, 395–402
  - Win32 APIs, 407–411
  - Windows Runtime storage API, 402–407
- local webpages, 282–284
- location, 707–750
  - CBE tracking apps, requirements for, 741–742
  - CivicAddressResolver class, 712
  - determining in Phone 7, 708–712

## LocationLens solution

- determining in Phone 8, 722–727
  - handled by Windows 8 vs. Phone, 906
  - Maps API, 727–740
  - SimpleGeoWatcher solution, 708
  - StatusChanged events, 712
  - tracking in Windows 8 vs. Phone 8, 911
  - LocationLens solution, 655, 657, 660
  - locking the emulator, 481
  - lock screen
    - background, updating with GBAs, 378–382
    - DeviceLockImageURI element, 383
    - notifications, GBAs and, 382–384
    - Obscured/Unobscured events and, 46
    - providers, adding to list of, 378
    - specifying location of image for, 380
    - status, 382
  - LockScreenDemo app, 378, 382
  - logos (Wallet), 801
  - Longitude property (AdControl), 815
  - LongListSelector control
    - ItemsSource converting to CollectionViewSource.View, 183
    - ListBox vs., 506–507
    - Windows Phone 7 and, 528
  - LoopingSelector (Windows Phone Toolkit), 352
  - low memory, 464
  - Low-Pass Filtering (AccelerometerHelper class), 246
  - loyalty cards, managing with Wallet Hub, 796
- ## M
- magnetic sensor, 256–261
  - magnetometer device driver, 256–261
  - MainApp.Tests project, 473
    - limitations, 473
  - MainPage code-behind, 328
  - MainPage.xaml, 26
    - code, 286
    - editing, 27–31
  - MainViewModel property (App class), 53, 472
  - MakeSpriteFont utility, 963
  - managed code
    - capturing video in, 649
    - native code vs., 867
    - photo manipulation and, 643
    - sharing, with partial classes, 922–924
    - SQLite, using with, 436
  - ManagedMediaHelpers class, 205–207
    - MediaParsers.Desktop, 205
    - MediaParsers.Phone, 205
    - Mp3MediaStreamSource.Phone, 205
    - usage, 205–207
  - managed phones, 847–849
    - enrollment, 847–849
    - MDM and, 847
  - manifest editor, 544
  - manifest entry
    - ID\_FUNCCAP\_EXTEND\_MEM, 464
    - ID\_REQ\_MEMORY\_300, 464
  - manifest file, 441
  - ManipulationCompleted event, 129
  - ManipulationDelta event, 129
  - ManipulationDemo app, 126
  - manipulation events, 126–132
  - ManipulationStarted event, 130
  - ManipulationXXX events, 126–132
    - MouseXXX events and, 133
  - manual memory management, 869
  - Manual Tests tab (Store Test Kit), 440, 442
  - MapAnimationKind values (Map controls), 732
  - MapCartographicMode values (Map control), 729
  - MapColorMode property (Map controls), 729
  - Map controls
    - ConvertViewportPointTo
      - GeoCoordinate method, 734
    - LandmarksEnabled flag, 732
    - MapDemo\_SetView solution, 731
    - MapLayerDemo app, 733
    - PedestrianFeaturesEnabled flag, 732
    - SetView method, 731
  - MapDemo\_SetView solution, 731
  - MapDemo solution, 727
  - MapDirections app, 734
  - MapDownloaderTask, 188
  - MapDownloaderTask (map launcher), 739
  - MapLayerDemo app, 733
  - MapOverlay objects, 732–734
  - MappedPhotoAppsPicker solution, 653
  - MappedUri property, 71
  - MapRoute API (Map controls), 734–739
  - MapRoute object, 737
  - maps, 707–750
    - offline caching/preloading of data for, 708
  - Maps API, 727–740
    - launchers, 739–740
    - location simulator quirks, 723
  - Map control, 727–734
  - MapDemo solution, 727
  - Microsoft.Phone.Maps.Controls namespace, 727
  - namespace locations of, 737

- routes/directions, 734–739
  - TestMapsTasks solution, 739
- MapsDirectionsTask (map launcher), 188, 734–739, 739
- MapsTask (map launcher), 188, 739
- MapUpdaterTask (map launcher), 188, 739
- MapUri method (UriMapperBase), 197
- Map.ViewportPointToLocation method, 715
- MarketplaceDetailsTask, 188, 455, 458
- MarketplaceHubTask, 188, 455
- MarketplaceReviewTask, 188, 455
- MarketplaceSearchTask, 188, 455
- market requirements, 460
- MAX\_CAP, 463
- maximumAge parameter (GetGeopositionAsync method), 724
- m\_constantBufferData member (CubeRenderer.h), 950
- MDLXAPCompile tool, 853
- MDM. *See* Mobile Device Management
- MDM vs. exchange servers, 847
- “me” identifier, 289
- MeasureOverride method, 90–91, 106
- MediaElement class, 204–205
  - BTS and, 360–361
  - controls, 207–211
  - isUpdatingSliderFromMedia flag, 210
  - vs. SoundEffects, 211
  - TestMediaElement solution, 204
  - TestVideo solution, 208
- MediaLibrary API, 228, 624–633
  - adding photos to, 629–633
  - handled by Windows 8 vs. Phone, 909
  - SavePictureToCameraRoll method, 648
- Media namespace (System.Windows), 649
- MediaOpened event, 361
- MediaParsers.Desktop (ManagedMediaHelpers class), 205
- MediaParsers.Phone (ManagedMediaHelpers class), 205
- media playback, 203–211
  - ManagedMediaHelpers class, 205–207
  - MediaElement class, 204–205
  - MediaElement controls, 207–211
  - MediaPlayerLauncher, 203–204
  - MediaStreamSource class, 205–207
- MediaPlayerLauncher, 188, 203–204
  - Controls property, 204
  - TestMediaPlayer solution, 203
- media services, 187–232
  - audio input/manipulation, 211–227
  - audio/video APIs, listed, 202
  - choosing orientation/size of, 204–205
  - media playback, 203–211
  - Music and Videos Hub, 227–230
- MediaStreamSource class, 205–207
  - adaptive streaming, 205
  - helper classes for, 205–207
  - ID3 v1/v2 metadata support, 205
  - multi-bitrate support, 205
  - RTSP:T protocol support, 205
  - SHOUTcast protocol support, 205
  - TestMediaHelpers solution, 206
- membership cards, managing with Wallet Hub, 786, 796
- memory
  - allocate more, 463
  - consumption, 463
  - generate report, 495
  - limits, 463, 500
  - MAX\_CAP, 463
  - MIN\_CAP, 463
  - profiling, 493
  - targeting apps based on device, 463–466
- memory analysis report, 500–502
- Memory API (DeviceStatus API), 491–492
- memory cap values
  - default for non-XNA Managed apps, 464
  - default for XNA/Native apps, 464
  - optional higher cap for all apps, 464
- MemoryDiagnosticsHelper, 492
- memory leaks, 500
- memory usage
  - limits, 491
  - monitor change, 492
  - performance data, 493
- Memory usage MB, 497
- MergedDictionaries, 99
- MessageBox control, 113
  - providing feedback to, 305
  - in Windows 8 vs. Windows 7, 523–524
- Messaging apps, III
- metadata, 455
  - app versions, 459
- Microphone class
  - DecibelMeter solution, 215
  - GetSampleSizeInBytes, 214
  - Start/Stop methods, 221

## microphones

- microphones
  - audio format of, 214
- MicroSD cards in emulator, 487
- Microsoft Account, 712
- Microsoft Ad Exchange, 811
- Microsoft Advertising control. *See* Ad Control
- Microsoft Advertising SDK, 812–815
- Microsoft ASP.NET, 315
- Microsoft.Bcl.Async NuGet package, 278
  - in Microsoft Visual Studio, 278
- Microsoft Blend 2012 Express for Windows Phone, 18
- Microsoft.Devices.Sensors namespace, 238
- Microsoft Hyper-V, 19
- Microsoft Intermediate Language (MSIL), 21
- Microsoft Management Console (MMC), 851
- Microsoft namespace, 904
- Microsoft.Phone.Controls.Maps.dll, 713
- Microsoft.Phone.Maps.Controls namespace, 727
- Microsoft.Phone.Reactive.dll, 242
- Microsoft.Phone.UserData namespace
  - calendar query APIs, 614
  - contacts, querying and, 594
- Microsoft Photosynth app, 659
- Microsoft pubCenter, 819–822
- Microsoft pubCenter Publisher Agreement, 822
- Microsoft Push Notification Service (MPNS), 558
- Microsoft Silverlight, 903
- Microsoft Silverlight WebBrowser, 273
- Microsoft Visual C++ 2012, 867
- Microsoft Visual Studio 2012
  - Ad Controls, adding in, 812
  - adding IXHR2 support to, XXV
  - App.xaml, 26
  - App.xaml.cs and, 26, 744
  - Assets folder, 26
  - async pack and, 278
  - Bing Maps and, 713
  - BingMapsLocation solution, 714
  - CheckAccess method (Dispatcher class), 583
  - creating new projects in, 22–24
  - CSS stylesheet, 284
  - data-binding collections in, 149
  - Databound Application template, 175–180
  - data resources, including, 94
  - Debug Location toolbar, 373
  - default UnhandledException handler for, 42
  - Direct3DBase class in, 945
  - Direct3D project types in, 936–937
  - DirectXTK\_WindowsPhone8 project, adding, 962
  - event handlers, templates for, 35
  - Express edition of, 902
  - ExtendedTask element, 372
  - full vs. express versions, 18
  - GeocodeService, configuring, 717
  - graphical manifest editor, 711
  - Grid/Split app templates, 932
  - linked files and, 919–920
  - localhost in, 317
  - LocalizedStrings.cs, 26
  - location sensor simulator, 747–748
  - MainPage.xaml, 26
  - Resources\AppResources.resx, 26
  - Scheduled Task Agent project, 365
  - SQLite, installation instructions, 435
  - starter HTML page, 284
  - Stop Debugging menu option, 373
  - Store Test Kit and, 439
  - tiles, generic placeholders for, 545
  - WebBrowser control declaration, 284
  - Windows Phone apps, 284
  - Windows Phone Audio Playback Agent project, 384
  - Windows Phone Runtime component, 925
  - Windows Runtime components and, 885
- Microsoft Visual Studio 2012 Express
  - PCL projects and, 915
  - VCD template files, 757–759
- Microsoft Visual Studio 2012 Express for Windows Phone, 18
  - Phone 7 and 8, support for, 20
- Microsoft Visual Studio 2012 Professional, 317
  - PCL projects and, 915
- Microsoft Xbox Live
  - avatar, 273
- Microsoft XML Core Services 6 (MSXML6)
  - library, XXV
- MIN\_CAP, 463
- Mixed Mode app type, 10
- MMC. *See* Microsoft Management Console
- MO. *See* mobile operator
- Mobile Core
  - CoreCLR, 17
  - Core System and, 16–17
  - Trident rendering engine, 17
- Mobile Device Management (MDM), 847
- mobile operator (MO), 845
- mobileoptimized tag, 286
- MobileServiceCollectionView class, 347

Mobile Services app, 345  
     create, 345  
 mock IAP, 832–835  
 MockIAPLib library, 832–835  
 Model-View-ViewModel (MVVM) pattern, 174–175  
     code sharing and, 913–914  
     implementing, 175–180  
     LINQ-to-SQL databases and, 415  
     performance with, 185  
     Phone 7.1 apps and, 530  
     pivot apps and, 180–181  
     PivotApp solution, 180  
 Mode property (AppBar), 108  
 modification  
     ProductID, 455  
     Refresh method, 330  
 Motion APIs, 266–271  
     navigation and, 5  
     sensor configurations of, 268  
     SimpleMotion app, 266  
     SimpleMotion\_More solution, 270  
 Motto property ({Binding} syntax), 147  
 MouseButtonEventArgs, 123  
 Mouse events, 132–133  
 MouseLeftButtonDown event, 121  
 MouseXXX events, 132  
 MovementThreshold property (GeoCoordinate-  
 Watcher), 749  
 MovementThreshold property (Geolocator  
 class), 725  
     performance and, 749  
 Mp3MediaStreamSource.Phone (ManagedMedia-  
 Helpers class), 205  
 Mp3MediaStreamSource.SL4 (ManagedMediaHelp-  
 ers class), 205  
 ms-appdata URI scheme, 405, 430  
 ms-appx URI scheme, 403, 430, 760  
 MSDN  
     documentation available in, 904–905  
     requirements, 442  
 ms\_nfp\_launchargs query string parameter (OnNavi-  
 gatedTo event handler), 688  
 MTAThread (Platform), 946  
 multi-bitrate support, 205  
 MultiByteToWideChar function, 884, 885  
 multi-targeting, 459  
 multithreaded apartment (MTA), 946  
 multitouch gestures in emulator, 487

Music and Videos Hub, 227–230  
     MediaLibrary class, 228  
     TestMediaHub solution, 227  
 Music & Video app, 11  
 Mutex, 368  
 MVVM. *See* Model-View-ViewModel (MVVM)  
 pattern

## N

namespaces, 904–905  
 Name value (resource), 96  
 native apps, 466  
 NativeBackButton solution, XXII  
 native code, 867–900  
     asynchronous code and, 888–896  
     back key press handling, XXII–XXVI  
     capturing video with, 649  
     C++ Component Extensions (C++/CX), 875–878  
     choosing photos with, VIII–XXVI  
     component object model and, 899–900  
     consuming asynchronous code in, 896–898  
     debugging mixed-mode projects, 888  
     device state and, X–XXVI  
     Direct3D and, 937  
     HTTP networking and, XXIV–XXVI  
     integrating, I–XXVI  
     launchers and, IV–XXVI  
     managed-code vs., 867  
     profiling, 502–503  
     screen orientation and, XIII–XXVI  
     screen resolution and, XV–XXVI  
     share contracts and, I–XXVI  
     user default language and, X–XXVI  
     user input and, XVII–XXVI  
     web connectivity and, XXIV–XXVI  
     and Win32 API, 898–900  
     Windows Runtime API and, 878–888  
     write apps using, 502  
 NativeEmailCompose solution, V  
 NativeHttpNetworking solution, XXV  
 NativeOrientationChanger solution, XIII  
 NativePhotoPicker solution, IX  
 native profiling  
     report views, 503  
     summary report, 502  
 NativeSharing solution, II  
 NativeTextInput solution, XVII, XX  
 NaturalDuration value (MediaElement class), 208

## Natural Language Support (NLS)

- Natural Language Support (NLS), 431
- NAudio library, 223
- Navigate element (VCD schema), 310, 759
- Navigate To Event Handler, 120
- NavigateUri property (HyperlinkButton), 63, 64
- NavigatingCancelEventArgs (OnNavigatingFrom method), 59
  - IsNavigationInitiator property, 69
- navigation (apps), 52–53
  - backstack management, 60–63
  - cancelling, 59–60
  - deep links and, 762–763
  - file type associations, 72–75
  - forward navigation support, 48
  - Fragment, 65–68
  - NavigateUri, 64
  - options for, 63–72
  - QueryString, 65–68
  - rerouting, 70–72
  - ReRouting solution, 70
  - resume policy, 43–45
  - separate assemblies and, 64–65
  - TestNavigating solution, 59
  - URI associations, 72–75
  - URI Mapping, 70–72
  - voice commands for, 753
- NavigationEventArgs object
  - IsNavigationInitiator property, 69
  - NavigationMode property, 68
- NavigationMode property
  - resume policies and, 44
  - values for, 68
- NavigationPage attribute (DefaultTask), 44
- NavigationParameters solution, 65
- Navigation Server platform service, 9
- NavigationService class
  - BackStack property, 60
  - OnRemovedFromJournal virtual method, 60
  - RemoveBackEntry method, 60
- NavigationService.GoBack method, 62
  - NavigateUri vs., 64
  - page constructors and, 53
- NavigationService.Navigate
  - page constructors and, 53
- NavigateUri property, 516
- NavUriFragment property (URI), 73
- NDEF Library for Proximity APIs, 701
- NDEF messages, 700–701
- NdefRecord class, 701
- Near Field Communication (NFC), 487, 696–705. *See also* proximity
  - in emulator, 487
  - NFC tags, 697–702
  - PeerFinder class and, 681
  - tap-to-connect, 686
  - Wallet and, 785–786
- .NET
  - C++/CX vs., 875–878
  - mapping to Windows Runtime types, 887
  - PCL projects and, 914–919
- NETFX\_CORE symbol, 921
- .NET sockets, 667, 675–680
- network connections, 481
- Network data transfer Mbps, 497
- network emulation driver, 481
- networking
  - Bluetooth devices and, 695
  - Bluetooth, finding peers with, 683–686
  - endianness and, 679
  - NFC and, 696–705
  - peer-to-peer socket communication, 680–695
  - proximity and, 680–695
  - sockets, 667–680
- NetworkStatusChanged event, 313
- NetworkToHostOrder method (IPAddress class), 680
- New Staging Deployment, 344
- NFC. *See* Near Field Communication
- NFC Data Exchange Format (NDEF), 697
- nfcdeeplink solution, 703
- nfcdeeplink URI scheme, 704
- NFC Forum, 700
- NfcReaderWriter solution, 697
- NFC tags, 697–702
  - apps, launching with, 702
  - message types, 699
  - NDEF messages, writing, 700–701
  - writing URIs to, 699–700
- Nokia location web service, 708
- Nokia Lumia, 482
- Nokia maps platform, 707–708
- Nonce, 307
- No Network setting, 483
- non-XNA, 466
- NorthwindEntities.cs, 327
- NotificationEventArgs parameter, 568
- NotifyComplete (background agents), 363–364
- NotifyEventArgs parameter, 284
- NotifyOnValidationError property ({Binding} syntax), 166

NotifyPropertyChanged method, 144  
 NotInstalled, 859  
 NotSupportedException, 528  
 NTLM. *See* Windows authentication (NTLM)  
 NuGet package, 280  
     Facebook, 294  
 NuGet Package Manager Console, 528  
 NwindODataClient app, 326  
 NwindODataClient\_Cached, 333  
 NwindODataClient\_Collection solution, 331  
 NwindODataClient\_Filtered solution, 329  
 NwindODataClient\_Paged, 332

## O

“oauth\_token\_secret” keys, 309  
 OAuth 1.0a protocol, 300  
     Twitter, 300  
 OAuth 2.0 protocol  
     Facebook, 300  
 oauth\_consumer\_key, 307  
 oauth\_nonce, 307  
 OAuth security headers, 302  
 oauth\_signature, 307  
 oauth\_signature\_method, 307  
 oauth\_timestamp, 307  
 oauth\_token, 307  
 oauth\_version, 307  
 object-oriented programming  
     platform-specific projects, 534–536  
 object-relational mapping API. *See* ORM (object-relational mapping) API  
 objects  
     Continuation, 336  
     DataServiceContext, 334  
     DataServiceState, 334  
     DispatcherTimer, 501  
     FacebookClient, 296, 298  
     HttpRequest, 324  
     LiveConnectClient, 289  
     ObservableCollection, 482  
     OptionalFeatures, 462  
     Quotation, 324  
     StandardTileData, 519  
     TaskCompletionSource, 306  
     Thumb, 527  
     TodoItem, 347  
     Type, 517  
 ObjectTrackingEnabled property, 432  
 ObscuredEventArgs object, 46  
 Obscured events, 45–47, 481  
     ApplicationIdleDetectionMode, 47  
     TestObscured solution, 45  
 ObservableCollection objects, 482  
     of SkyDrivePhoto objects, 292  
 ObservableCollection<Quotation>, 322, 324  
 ObservableCollection<T>, 320, 331  
 OData client tools, 326  
 OEM. *See* original equipment manufacturer  
 offensive content in apps, 446  
 OfferWebsite property (Deal), 791  
 OMSetRenderTarget (ID3D11DeviceContext), 951  
 OnBackPressed event handler  
     Direct3D games and, XXIV  
     native code and, XXIV  
 OnBackPressed event handler, 523  
 OneWay mode ({Binding} syntax), 142  
 OnGetLiveData  
     callback, 293  
     event handler, 290  
 OnInputPaneHiding event handler, XX  
 OnInputShowing event handler, XX  
 OnlinePaymentInstrument class, 806  
 OnManipulationCompleted method  
     overriding, 129  
 OnNavigatedFrom event, 336  
     backstack management and, 60  
     handling, 523  
     notifications, 52  
 OnNavigatedTo event, 328, 548  
     handling, 504  
     ms\_nfp\_launchargs query string parameter, 688  
     notifications, 52  
     overriding, 332  
     overriding, and BTS apps, 361  
     pinning tiles and, 553  
     reminders and, 357  
 OnNavigatingFrom method, 59  
 OnPlayStateChanged event (OnUserAction method), 394  
 OnPlayStateChanged override (AudioPlayerAgent), 386  
 OnPointerPressed event handler, XX  
 OnRemovedFromJournal virtual method (NavigationService class), 60  
 On-Screen Keyboard (OSK), 135–137  
 OnUserAction override (AudioPlayerAgent), 386  
 opacity of app bar, setting, 108  
 OpenAsync method (PhotoCaptureDevice class), 640

## Open Data (OData)

- Open Data (OData), 325
    - clients, creating, 326–329
    - vs. SOAP, 325
    - WCF data services and, 325–338
  - OpenReadAsync method
    - retrieving non-string data and, 276
    - SOAP vs. REST and, 324
    - WebClient and, 274
  - OpenReadCompleted events, 274, 324
  - OpenTransactedWriteAsync method (StorageFile class), 405
  - Optimal Filtering (AccelerometerHelper class), 246
  - optimization
    - caching data, 333
    - paging the data, 332
    - performance, 503
  - OptionalFeatures object, 462
  - orientation, 233–236
    - camera and, 636
    - modes of, 234
    - WrapOrientation solution, 234
  - OrientationChanged event (DeviceOrientationHelper class), 234, 251
    - exposed orientation values, 251
    - native code and, XIV
    - WrapPanel control vs., 236
  - original equipment manufacturer (OEM), 845
  - OriginalSource property, 123
  - ORM (object-relational mapping) API, 412
  - OS versions, targeting, 510
  - OtherKey property, 421
  - “oauth\_token” key, 309
  - OutOfMemoryException, 463
- P**
- Package Family Name (PFN), 694
  - Package.Launch API, 863
  - Package Manager platform service, 9
  - packages
    - manager, 278
    - Microsoft.Bcl.Async NuGet, 278
    - NuGet, 280
    - WPToolkitTestFx, 473
  - page constructors
    - blocking, 504
    - navigation and, 53
  - PageCreationOrder solution, 49
  - page model
    - creation order, 49
    - inter-app backstack, 47
    - PageCreationOrder solution, 49
  - Page model (app design), 47–51
  - PageNavigationState solution, 56
  - page state
    - categories of, 52
    - PageNavigationState solution, 56
    - PhoneApplicationPage.State property, 56
    - storage limit on, 59
  - page-to-page animations, 69
  - paging
    - per-table, 332
    - per-view, 332
  - PaidClicks solution, 840
  - Panorama control
    - Ad Controls and, 817
    - performance and, 507
    - re-templating, 92–94
    - vs. pivot, 507
  - Panoramaltem, 857
  - Parallel Patterns Library (PPL), 889, XXV
  - partial classes, 535
    - sharing in managed code, 922–924
  - partial keyword, 922
  - Pause method (SoundEffect class), 211
  - PaymentInstrument class, 806
  - pch.h/cpp, 942
  - PCL projects (Visual Studio), 914–919
    - CityFactBook\_PCL solution, 915
    - creating, 915–916
    - platform differences, handling, 916–919
  - PedestrianFeaturesEnabled flag (Map controls), 732
  - PeerFinder class (Windows.Networking.Proximity namespace), 681
    - AlternateIdentities property, 694
    - TriggeredConnectionStateChanged event, 688
  - PeerInformation class, 685
  - peer-to-peer socket communication, 680–695
    - sending URIs across devices, 702–705
    - tap-to-connect, 686–695
  - People Hub, 587–590
    - contact card user interface (UI), 588
    - custom contact stores, 605–613
    - data types by type of account, 601
    - extended properties and, 606
    - social networking and, 588
    - updating information in, 589

- performance
  - analysis sessions, 493
  - apps, considerations for, 34–36
  - best practices for, 503–507
  - bottlenecks, 487
  - DataBoundApp\_modified solution, 184
  - detailed report, 496
  - frame rate counters and, 488
  - goals, 488
  - image loading and, 506
  - LINQ-to-SQL
    - batch updates, speeding up, 433–434
    - memory usage, minimizing, 432
  - ListBox control issues, 506
  - LongListSelector vs. ListBox controls, 506–507
  - maintaining with Windows Runtime types, 883
  - MVVM pattern and, 185
  - native code and, 868
  - optimization, 503
  - panorama controls and, 507
  - perception, 504
  - profiling, 487–503
  - screen resolution/image quality and, 544
  - UI thread and, 504–505
  - user engagement, 504
  - warnings, 500
- PerformanceProgressBar, 504
- Periodic GBAs, 362
- PeriodicTask class, 362
- periodic tasks, 12
- persistent state, 52
- persisting data and larger datasets
  - larger datasets and, 407
- personal information exchange (PFX), 850
- Personal Information Management (PIM), 587
- PFX. *See* personal information exchange
- Phone 8 emulator
  - Fiddler and, 468–471
  - testing PhotoChooserTask in, 621
- PhoneApplicationFrame (Root Visual property), 81
  - AdControl and, 818–819
- PhoneApplicationPage\_BindingValidationError handler, 169
- PhoneApplicationPage class, 108
  - page state and, 56
  - persisting data with, 58
  - State property, 53
- PhoneApplicationService.State property
  - LifecycleState solution, 53
- PhoneCallTask object, 188
  - ID\_CAP\_PHONEDIALER capability, 190
  - required references for, 594
- phone chassis requirements, 236–237
- PhoneDarkThemeVisibility, 599
- PhoneExtensions namespace (Microsoft.Xna.Framework.Media), 658
- PhoneLightThemeVisibility, 599
- PhoneNumberChooserTask, 189, 592
- PhoneNumber filter (SearchAsync), 596
- PhoneNumberResult object, 593
- phone services, 187–232
  - Choosers, 187–192
  - Clipboard API, 230–231
  - extensibility models, 192–202
  - Launchers, 187
- Photo Apps Picker extension (Phone 7), 525, 651–653
  - vs. Photo Edit Picker, 525
- PhotoAppsPicker solution, 653
- PhotoCamera object, 634–639
  - manipulating preview buffer with, 643–647
- PhotoCaptureDevice class (Windows.Phone.Media.Capture namespace), 639–649
  - capturing photos with, 647–651
  - manipulating preview buffer with, 643–647
  - preparing capture sequence, 640–643
  - preview buffer, 643–647
- PhotoChooserTask, 189, 621–624, VIII
  - return value of, 622
- PhotoEditPicker solution, 655
- Photo Edit Picker (Windows Phone 8), 653–655
  - vs. Photo Apps Picker, 525
- photos, 621–666
  - adding to collection, 629–633
  - aquiring, 621–624
  - auto-uploading, 662–664
  - capturing, 633–649, 647–651
  - choosing with native code, VIII–XXVI
  - focusing on specific points, 637–639
  - lenses, 655–661
  - MediaLibrary API, 624–633
  - photo edit picker, 653–655
  - Photos Hub, 649–655
  - picker feature, 290
  - reading, 624–628
  - scopes, 292
  - sharing, 661–664
  - SkyDrive, 290, 293
  - transforming, 631

## Photos\_Extra\_Share extension

- Photos\_Extra\_Share extension, 661
- Photos\_Extra\_Viewer extension, 653
- Photos Hub, 649–655
  - apps pivot in, 650–651
  - photo apps picker, 651–653
  - photo edit picker, 653–655
- PhotoSlideshow solution, 625
- Photos\_Rich\_Media\_Edit extension, 659
- Photosynth app (Microsoft), 659
- PhotoUploader solution, 663
- PhraseLists
  - CommandSet and, 764
  - empty/missing values and, 765
  - implementing, 762–765
  - updating at runtime, 766–767
- PickSingleFileAsync method, IX
- PictureAlbum collection (MediaLibrary), 625
- Pictures property (MediaLibrary), 624
- PIM (Personal Information Management), 587
- pinch and stretch gestures, 130
- pinned tiles, 550–557
  - home buttons and, 554
- PinnedToStart filter (SearchAsync), 596
- PinTiles\_Home solution, 554
- PinTiles solution, 550
- pitch, 262
- Pitch property (Map control), 728
- Pivot Application project (Visual Studio), 180
- Pivot apps, 273
  - column filtering in, 181
  - MVVM in, 180–181
  - row filtering in, 182–185
- PivotApp solution, 180
- pivot control, 482, 507
  - Ad Controls and, 817
  - Photos Hub and, 650–651
  - slideshows and, 627–628
  - vs. panorama, 507
- PivotFilter app, 182
- pivot items, 482
- PixelShaderInput, 953
- Plain Old CLR Objects (POCOs), 415
- PlaneProjection property (Image control), 128
- platform abstraction layer (PAL), 924
- platform service, 9
- platform-specific functionality, 532
- platform-specific projects, 528–536
  - creating, 529–530
  - object-oriented programming and, 534–536
  - Phone 7.1 libraries, sharing, 530–531
  - source code files, linking, 531–534
- platform stack, 8–9
- PlayStateChanged event, 393
- PNG image files
  - tiles and, 546
  - WriteableBitmapEx library and, 548
- pointers
  - circular references to, 871
  - shared, 870
  - smart, 869–872
  - unique, 870
- polling, 367
- PopulateAPIItemsFromXml method (MockIAP), 834
- Popup controls, 113
- PopupControl transient panel, 63
- pop-up windows, 113
- portability
  - maintaining with Windows Runtime types, 883
  - native code and, 868
- Portable Class Library (PCL) project, 913
- PortraitFlipped value (DisplayOrientations enumeration), XV
- portrait modes (orientation), 234
- PositionAccuracy, simulator quirks for, 723
- PositionChanged events, 709–710
  - Geocator class and, 722
- positioning
  - of Ad Controls, 816
  - auto-scaling and, 78
- Position property (GeoCoordinateWatcher object), 710
- Position value (MediaElement class), 208
- PostalCode property (AdControl), 815
- POST operation, 323
  - native code and, XXVI
- PowerLevelChanged event, 577
- predefined grammars, 771–772
- PreloadGrammarsAsync method (SpeechRecognizer class), 776
- premium downloadable content (PDLC), 829–843
- Present method (Direct3DBase class), 952
- preview feed, displaying, 635
- primary tile
  - upgrade, 513
  - WMAppManifest graphical editor, 543
- processor speeds, 460
- ProductID (app), 458
  - modification, 455
  - updating after submission to Dev Center, 842

- ProductListing elements (IAP), 834
- profiling, 487–503
  - app analysis, 493
  - background agents support, 502
  - DeviceStatus Memory API, 491–492
  - Direct3D, 502
  - execution, 493
  - frame rate counters, 488–489
  - memory, 493
  - native code, 502–503
  - redraw regions and, 490–491
  - Windows Phone Performance Analysis Tool, 493–502
  - XAML app, 502
- ProgressBar control, 504
  - performance issues, 504
- Progress event, 860
- Progress handler, 860
- ProgressIndicator control (SystemTray), 112, 504
- progress\_reporter<TProgress>, 890
- progress UI, 484
- projects, 21–31
  - ApplicationIcon.png, 26
  - App.xaml, 26
  - App.xaml.cs, 26
  - Assets folder, 26
  - code, entering and testing, 27–31
  - creating, 22–24
  - duplicate existing, 529
  - LocalizedStrings.cs, 26
  - MainApp.Tests, 473
  - MainPage.xaml, 26
  - Resources\AppResources.resx, 26
  - structure of, 24–27
  - Windows Phone Class Library, 531
  - WMAppManifest.xml, 25
- promises (JavaScript), 898
- properties (class), 101–107
  - attached, 104–107
  - dependency, 101–104
- Properties window (Expression Blend), 173
- PropertyChangedEventHandler, 143
- PropertyChanging event, 432
- Protocol contracts, 906
- proxies and push notifications, 561
- proximity. *See also* Near Field Communication (NFC)
  - handled by Windows 8 vs. Phone, 907
- ProximityDevice class (Proximity namespace), 697
  - PublishBinaryMessage method, 698
- Proximity namespace (Windows.Networking), 681, 697
- proxy classes, 320, 321
- proxy code, generating, 327
- pubCenter (Microsoft), 819–822
- publication requirements
  - for background audio agents, 394
  - of BTS requests, 360
- publish automatically, 447
- PublishBinaryMessage method (ProximityDevice class), 698
- Publisher Agreement (Microsoft pubCenter), 822
- publishing apps, 439–466
  - beta testing, 458
  - Dev Center, reports available from, 452–454
  - preparations for, 439–443
  - process of, 443–452
  - updates, sending, 454–458
  - using WCF data services, 325
  - versioning, 459
- PushMoreClient app, 579
- PushMoreClient solution, 576
- PushMoreServer solution, 574, 575
- push notification client
  - ErrorOccurred event, 576–578
  - implementing, 565–568
  - PushMoreClient app, 579
  - PushSimpleClient\_Registration sample, 572
  - PushSimpleClient solution, 565
  - Push ViewModel, implementing, 579–584
  - user opt-in/out, 578–579
- Push notifications
  - client, 565–568
  - handled by Windows 8 vs. Phone, 908
  - HttpNotificationReceived events, 567
  - ID\_CAP\_PUSH\_NOTIFICATION capability, 566
  - implementation of, 558
  - message elements of, 561
  - Microsoft Push Notification Service (MPNS), 558
  - process of, 559
  - PushMoreServer solution, 575
  - PushViewModelClient solution, 580
  - registration web services, 569–573
  - response codes, 562
  - security, 584–585
  - server, 561–565
  - server resets and, 568
  - ShellToastNotificationReceived events, 567
  - size limit of, 560
  - X-NotificationClass, 574

## push notification server

- push notification server
  - batching intervals, 574
  - features, 558–586
  - PushMoreServer solution, 574
  - PushSimpleServer solution, 561
  - registration web service and, 569–573
  - required functionality of, 561
  - response codes, 576
  - response information available to, 576
  - XML payloads and, 575
- pushpins (Bing Maps), 714–716
  - Map.ViewportPointToLocation method, 715
  - Phone 8 and, 732
- PushSimpleClient\_Registration sample, 572
- PushSimpleClient solution, 565
- PushSimpleServer\_Registration sample code, 571
- PushSimpleServer solution, 561–565
- PushViewModel class, 582
- PushViewModelClient solution, 580
- PUT operations, 323
  - native code and, XXVI

## Q

- QotdService solution, 315
  - QotdServiceClient, 321
  - QotdService WCF, 341
  - Quotation object, 324
- queries
  - COUNT, 299
  - LINQ, 330
- QueryCompleted event, 736
- QueryString property, 67–68
- Quick Response (QR) code, 789
- quirks mode, 520–521

## R

- RadioDisabledException, 524
- raw push notifications
  - description/constraints on, 560
- reactivated events, 41
- Reactive Extensions for .NET. *See* Rx (Reactive Extensions for .NET)
- ReadAsync and disconnects, 690
- ReadingChanged event, 247
- ReadoutEnabled property (SpeechRecognizerUI class), 769

- real estate available to apps, 79
- ReceiveAsync method and .NET sockets, 676
- RecognizeWithUIAsync method, 769
- redirection, 70–72
- redraw regions, 490–491
- refactoring tests, 472
- reference counting (COM), 946
- ref keyword (C++/CLI), 881
- reflection, 558
- RefreshData method, 328, 337
- RefreshInstallState method, 859
- regional availability, updates and, 454
- RegisterAttached method (DependencyObject), 106, 107
- RegisterCompleted event, 572
- RegistrationService web service sample, 569
- registration web services, 569–573
  - PushSimpleServer\_Registration sample code, 571
  - RegistrationService web service sample, 569
- regular expressions, 457
- relative screen layout model, 84
- Release build, 442, 494
- Release-build XAP, 439
- reminders (background), 354–357
  - triggering, 481
- removable devices access, 909
- RemoveBackEntry method (NavigationService class), 60
- rendering pipeline, 934
- Render method (CameraPreview class), 646
- Render method (CubeRenderer.h), 951
- RenderTransform (Motion class), 269
- ReportInterval property (Geolocator objects), 725
- ReportProductFulfillment method (CurrentApp), 841
- reports
  - app analysis, 495
  - App Responsiveness, 497
  - CPU Usage %, 497
  - External Events, 497
  - Frame Rate, 497
  - GC events, 498
  - Image loads, 498
  - Memory usage MB, 497
  - native profiling summary, 502
  - Network data transfer Mbps, 497
  - Storyboards, 497
- Representational State Transfer. *See* REST
- RequestProductPurchaseAsync method (CurrentApp class), 837
- Request Token Secret (Twitter), 307

- Request Token (Twitter), 302, 306, 307
- required artwork (apps), 450
- requirements
  - certification, 442
  - conflicting, 460
  - market, 460
  - MSDN, 442
- Requirements section (app manifest), 464
- ReRouting solution, 70
- Resizable Tiles, 510
- ResolutionScale enumeration, XVI
- ResourceDictionary, 98
- Resource-intensive agents, 12
- Resource-intensive GBAs, 362
- ResourceIntensiveTask class, 362
- resource management
  - Deactivated events and, 40
  - tombstoning and, 40
- Resource Manager platform service, 9
- resources, 94–99
  - consuming externally defined, 99
  - data, 94–95
  - Name vs. Key, 96
  - Style resources, 99–101
  - XAML resources, 95–99
- Resources\AppResources.resx, 26
- Resources property, 97
- response list, push notifications and, 562
- responsiveness issues, diagnosing, 495
- REST, 297
  - APIs, 297
  - Bing Maps implementation, 719–721
  - HttpRequest, 324
  - SOAP vs., 323–325
  - TestGeocodeService\_REST solution, 716
  - Twitter, 300
  - update, 323
  - URL format, 301
  - WebClient objects, 324
  - web services, 324
- RestQotd solution, 324
- resume policy, 44
- RetemplatedPanorama solution, 93
- reusability and native code, 868
- rich media editing, 659–661
- RichTextBox, 230
- roaming data, 921
- roll, 262
- Root Certificate Authority (CA), 584
- RootPictureAlbum property (MediaLibrary), 624

- RootVisual property (Application base class), 81
- RoseColoredCamera solution, 639, 644
- RoutedEventArgs (RoutedEventArgs), 123
- routed events, 121–124
- RoutedEvents class, 121–124
- RouteQuery API (Map controls), 734–739
- RowDefinitions (Grid control), 87
- RTSP:T protocol support, 205
- RuleOfThirdsCamera solution, 634
- RunAsync method, 505
- RunningInBackground event (App.xaml), 742, 744
- Run Slow Method button, 495
- runtime performance, 487
- Rx (Reactive Extensions for .NET), 242–245
  - FilteredAccelerometer app, 242
  - location data and, 749

## S

- SaveAppointmentTask, 188, 618
- SaveAsync method (Deal), 791
- SaveContactTask, 189, 602
  - prepopulating/encapsulating, 603
- SaveEmailAddressTask, 189, 603
- SaveJpeg method (WriteableBitmap class), 630
- SavePhoneNumberTask, 189, 603
- SavePictureToCameraRoll API, 658
- SavePictureToCameraRoll method (MediaLibrary), 648
- SaveRingtoneTask, 189
- say-as element (SSML), 779
- ScaleFactor property, 105
- SCCM. *See* System Center Configuration Manager
- ScheduledActionService objects, 351–352
- scheduling policy, 910
- scopes, 288
  - extended, 288
  - photos, 292
  - wl.basic, 288
  - wl.offline\_access, 288
  - wl.photos, 292
  - wl.signin, 288
  - wl.skydrive, 292
- screen layout, 84–88
  - SimpleLayout solution, 84–138
- screen orientation and native code, XIII–XXVI
- screen resolution
  - FramePageSizes app sample, 79–81
  - image quality and performance, 544

## screen resolution levels

- integrating with native code, XV–XXVI
- opting out of, 81
- support, 78
- screen resolution levels, 544
- ScreenResolutions section, 465
- scripting in WebBrowser control, 286
- ScriptNotify event, 286
  - handler, 286
- SearchAsync method (Contacts object), 595
  - filters supported by, 596
  - LINQ vs. (performance), 599
  - query return format, 597
- search extensions, 193
- SearchTask, 188
- secondary tiles, 519
- SecondaryTiles app example, 546
- Second Level Address Translation (SLAT), 19
- Secured network access, 846
- Secure Sockets Layer (SSL), 339, 675, 846
  - push notifications and, 584
- security
  - Bing Maps and, 713
  - CredentialsProvider, 713
  - domain network, 317
  - push notifications, 584–585
  - security chambers, 13
- security chambers, 13
- security models, 13–16
  - capabilities, 13–16
  - least privilege, 13
- security sandbox, 460
- Segoe WP, 5
- SelectionChanged event handler, 685
- SelectionChanged handler, 356
- sensor APIs
  - measurement units, Compass vs. Gyroscope classes, 265
  - SensorBase<T> base class and, 237
- SensorBase<T> base class
  - IsValid property, 259
  - sensor APIs and, 237
- SensorReadingEventArgs<AccelerometerReading> events, 238
- SensorReading (Motion class), 270
- sensors, 233–272
  - accelerometer, 238–256
    - APIs for, 237
    - calibration of, 237
    - error susceptibility of, 266
    - gyroscope, 261–265
      - handled by Windows 8 vs. Phone, 907
      - magnetic, 256–261
      - motion APIs, 266–271
      - orientation, 233–236
        - Rx (Reactive Extensions for .NET), 242
  - Separation of Concerns (SoC), 171–185, 472
    - DataBoundApp solution, 175
    - MVVM pattern, 174–175
  - Serialize method (DataServiceState), 334
  - Serializer, 375
  - ServiceContract, 315
  - service endpoint, 318
  - service method, 322
  - service plans and Data Sense API, 312–314
  - ServiceReferences.ClientConfig, 318, 320, 322
  - SessionChanged event, 289
  - SetBinding method (BindingOperations), 144
  - SetEntitySetPageSize method (DataServiceConfiguration), 332
  - SetProperty extension method, 516
  - SetProperty method (PhotoCaptureDevice class), 641
  - SetText method (Clipboard object), 230
  - setting breakpoints, 467
  - SetUserAttentionRequiredNotification method (WalletItem), 808
  - SetValue method (DependencyObject), 101
  - SetView method (Map controls), 731
  - 720p (screen resolution), 78, 467, 544
    - Direct3D games and, XV
    - page sizes for, 80
  - shaders, 934
  - shake, 252–256, 252–272
    - TestShakeHelper solution, 255
  - ShakeGestureEventArgs, 254
  - Shake Gesture Library, 252
    - modifications required when using updated Accelerometer Helper library, 254
    - usage, 253–255
  - ShakeGesturesHelper class, 252
  - ShakeMagnitudeWithoutGravitationThreshold property, 255
  - Share charm, II
  - Share contracts
    - handled by Windows 8 vs. Phone, 905
    - native code and, I–XXVI
  - SharedCodeClass, 536
  - SharedMethod, 536
  - shared pointers, 870
  - SharedStorageAccessManager class, 75

- ShareLinkTask, 188, 487, I
- ShareMediaTask, 188
- share picker, 661–662
- ShareStatusTask, 188, I
- sharing
  - photos, 661–664
  - share picker and photos, 661–662
- ShellTile, 515, 516
- ShellTile.Create method, 547
- ShellTileData, 518
- ShellTileData-derived classes, 549
- ShellTile.Update method
  - cycle tiles and, 541
  - iconic tiles, 543
- ShellToastNotificationReceived events, 567
- Short Message Service (SMS) message, 442, 578
- short-range wireless protocols, 696
- SHOUTcast protocol support, 205
- ShowCamera property (PhotoChooserTask), 623
- ShowConfirmation property (SpeechRecognizerUI class), 769
- ShowGridLines property (Grid control), 84
- ShutterKeyHalfPressed event (CameraButtons class), 637
- ShutterKeyPressed event (CameraButtons class), 637
  - CapturePhoto method and, 648
- Signature method, 307
  - HMAC-SHA1, 307
- Silverlight, 506
- Silverlight4 (assembly version), 485
- Silverlight apps and XNA types, 217
- Silverlight Toolkit, 117
- SimpleAccelerometer solution, 239
- SimpleAppConnect solution, 199
- SimpleAppInstantAnswer solution, 201
- SimpleBingMaps solution, 713
- SimpleCbe\_MultiPage solution, 746
- SimpleCbe solution, 742
- SimpleCompass solution, 257
- SimpleDataBinding\_Format solution, 145
- SimpleDataBinding solution, 140
- SimpleEvents solution, 121
- SimpleGeoLocator app, 722
- SimpleGeoWatcher solution, 708
- SimpleGyroscope solution, 262
- SimpleLayout solution, 84–138
- SimpleMotion app, 266
- SimpleMotion\_More solution, 270
- Simple Object Access Protocol. *See* SOAP
- simple operator-precedence bug, 478
- SimplePeerFinder app, 681, 686
- SimplePixelShader.hlsl and SimpleVertexShader.hlsl, 942
- SimplestAppConnect solution, 194
- SimplestD3DApp solution, 954
- SimpleWeather app, 373
- SIMULATE\_71 (symbolic constant), 519
- Simulation Dashboard, 481
- sine wave, 225
- single-contact Choosers, 590–594
  - app lifecycle events and, 591
  - PhoneNumberChooserTask, 592
  - PhoneNumberResult object, 593
- SingleContactsChoosers solution, 590, 593
- single-core processors, 460
- SinglePhotoChooser solution, 622
- single-threaded apartment (STA), 946
- SIP. *See* Soft Input Panel
- SipDemo solution, 135
- skeuomorphic elements, 5
- SkyDrive, 273, 312
  - connecting to, with Live SDK, 290–294
  - MediaLibrary and, 625
  - photos, 290, 293
- SkyDrivePhoto
  - ObservableCollection, 292
- SkyDrivePhoto class, 290
- Slider control
  - behavior, 527
  - in Phone 7 vs. Phone 8, 527
  - MediaElement objects, controlling with, 209–211
  - XAML template, 527
- SliderMigration solution, 527
- SlowApp solution, 494, 498
- smart pointers (C++), 869–872
- SMS. *See* Short Message Service
- SmsComposeTask, 188
- snapped view (Windows 8), 931–932
- SOAP
  - Bing Maps, implementing, 717–719
  - Bing Maps services, 716–720
  - REST vs., 323–325
  - TestGeocodeService\_SOAP solution, 716
  - update, 323
  - vs. Open Data, 325
- SoC. *See* Separation of Concerns
- social graph data, 298
- social network integration
  - DataPackage type and, III
  - in emulator, 487

## SocketAsyncEventArgs class

- SocketAsyncEventArgs class, 676
- SocketClient app, 669
- SocketEventCompleted event handler, 676
- sockets, 667–680
  - handled by Windows 8 vs. Phone, 907
  - .NET sockets, 675–680
  - Windows Runtime sockets API, 668–675
  - WinSock API, 680
- SocketServer app, 669
- Sockets namespace (System.Net), 675
- Soft Input Panel
  - tuning, 137
- Soft Input Panel (SIP)
  - clipboard and, 230
  - orientation and, 233
- Software Loopback connection, 313
- solutions
  - QotdService, 315
  - RestQotdClient, 324
  - RestQotdService, 324
  - SliderMigration, 527
  - SlowApp, 494, 498
  - TestFrameworkApp, 472
  - TileLightUp, 511
- SortDescription property (CollectionView Source), 158, 160
- SortedDictionary, 309
- SoundEffect classes, 211–214
  - vs. MediaElement, 211
  - TestSoundEffect solution, 211
  - XNA SoundEffect classes, 211–214
- SoundEffectInstance class, 211–214
  - looping/3D audio effects, support for, 211
- SoundEffect.MasterVolume, 219
- SoundFx\_Persist solution, 222
- SoundFx solution, 218
- source compatibility, 913
- space character (%20), 761
- SpeakTextAsync method (Synthesis.SpeechSynthesizer class), 777
- speech, 753–784
  - custom grammars, 770–776
  - recognition, 767–776
  - text-to-speech, 776–781
  - ToDoList solution, 759
  - VCD files, 756–765
  - voice commands, 753–767
- speech recognition
  - accuracy of, vs. voice commands, 770
  - apps and, 767–776
  - built-in UX for, 767–769
  - capabilities required for, 768
  - PhraseLists, 762–765
  - reco (URI query element), 761
  - ToDoList\_CustomSpeechRecognizerUI solution, 769
  - ToDoList solution, 759
  - voiceCommandName (URI query element), 761
- Speech Recognition Grammar Specification. *See* SRGS (Speech Recognition Grammar Specification)
- SpeechRecognitionResult object, 769
- SpeechRecognizer class, 776
- SpeechRecognizerUI class
  - SRGS and, 774
  - turning off elements of, 775
- Speech Synthesis Markup Language. *See* SSML (Speech Synthesis Markup Language)
- SpeechSynthesizer class, 777
- splash screens, 27
- Split app template (Visual Studio), 932
- SpriteBatch class (DirectXTK toolkit), 645
- SpriteFont object, 963
- SQL Azure database, 345
- SQLCE, 423
- SQLite, 434–436
  - acquiring, 435
  - code wrappers for, 436
  - managed code and, 436
- SQL Server Compact 3.5 database, 427
- SQL Server Date type, 423
- SRGS (Speech Recognition Grammar Specification), 773–776
  - tag elements, 774
- SSL. *See* Secure Sockets Layer; *See* Secure Socket Layer
  - Root Certificate Authority (CA), 584
  - TLS certificates and, 584
- SSL mutual authentication, 340
- SSML (Speech Synthesis Markup Language), 777–781
  - keeping in synch with screen, 779–781
  - output, crafting, 777–779
- StackPanel control (Panel class), 84, 86, 505
  - in visual tree, 82
- StandardTileData object, 519
- Start button and Deactivated events, 37–38
- Start screen, 366
- startup experience, 484

- state
  - apps, 53–56
  - page, 56–59
  - simulating in emulator, 487
- static validation, 452
- StatusChanged event (GeoCoordinateWatcher), 710
- stenciling, 934
- storage, system, 906
- StoredContact class, 606
- Store Test Kit, 439–443
  - Application Analysis, 440
  - Application Details, 440
  - Automated Tests, 440
  - Manual Tests, 440
  - parts, 440
- store tile, 441
- Storyboards, 497
- Streams namespace (Windows.Storage), 671
- StreamSocketListener object, 671
- Stretch property (MediaElement class), 204
- StringConversions solution, 884
- StringFormat attribute, 145
- strings
  - in Windows Runtime, 884
  - "yes/no", 462
- structured testing, 472
- Style resources, 99–101
- styles
  - controlling, for UserControls, 928
  - implicit, 99–101
- submission tool, 446
- SubmitChanges method (System.Data.DataCon-  
text), 416, 417
- Subscribe method (IObservable<IEvent<>>), 243
- subscriptions, 832
- support
  - ActiveX controls, 846
  - enterprise-focused, 845
  - Exchange ActiveSync, 846
  - Facebook, 294
  - JavaScript, 846
- SupportedOrientations attribute, setting, 233–234
- supporting classes, 321
- surface counter, 489
- swap chain, 934
  - configuring, 945
- Symantec, 851
- synchronous code, moving to background, 889–890
- SyndicationFeed class, 485
- System.Array, 320

- System Center Configuration Manager (SCCM), 847
- System.Data.DataContext database schema, 412
  - appdata scheme, 430
- System.Globalization.RegionInfo.CurrentRegion.  
TwoLetterISORegionName, 815
- System integrity, 845
- System namespace, 904
- System.Observable.dll, 242
- System.ServiceModel.Syndication assembly, 485
- System.ServiceModel.Web.dll, 338
- System.Services.Data.Client.dll, 327
- System.Services.Data.Client.WP80.dll, 327
- System.Threading.ThreadPool, 504
- system tray, 77, 108–113
- SystemTray class, 109
- System.Windows.Media.Imaging.Extensions  
namespace, 630
- System.Windows.Media.Imaging namespace, 547
- System.Windows.Navigation namespace, 71
- System.Windows.Visibility, 113
- System.Windows.xaml, 82

## T

- table/view class, 328
- tag elements (SRGS), 774
- Tap+Send
  - DataPackage type and, III
- tap-to-connect communication, 686–695
  - reconnecting, 689–694
  - in Windows 8 apps, 694–695
- targeting by OS version, 510
- TargetNullValue attribute ({Binding} syntax), 147
- Task class (System.Threading namespace), 896
- TaskCompletionSource object, 306
- TaskFactory, 280
- TaskHost app model. *See* XAML app model
- TaskID property (URI), 73
- Task Parallel Library (TPL), 280–281, 306
  - Task<T> type, 280
- tasks
  - periodic tasks, 12
  - Resource-intensive agents, 12
- Task<T> type, 280
- TCP vs. UDP, 668
- TDD. *See* test-driven development
- team development, 172
- Team Foundation Service (TFS), 912
- Technical Exception, 451

## Technical Report 1 (C++ library)

- Technical Report 1 (C++ library), 869
- Terminate method, 523
- TestAccelerometerHelper\_CurrentValueChanged solution, 249
- test class, 475
- TestClass attribute, 474
- TestClipboard application, 230
- TestCommandBinding solution, 153
- TestDependencyProps solution, 103
- test-driven development (TDD), 472
- TestDynamicSounds\_Controls solution, 225
- TestDynamicSounds solution, 224
- TestFrameworkApp solution, 472
- TestGeocodeService\_REST solution, 716
- TestGeocodeService\_SOAP solution, 716
- TestGeocodeService solution, 716
- testing, 472–487
  - Ad Controls, 814, 822
  - backward compatibility, 521–538
  - camera apps, 634
  - IAP, 832
  - interruptions, 478
  - location sensor simulator (Visual Studio), 747–748
  - real-world conditions, simulating, 481–485
  - refactoring, 472
  - structured, 472
  - tombstoning, 485–486
  - unit testing, 472–480
  - with Windows Phone Emulator, 486–487
  - Windows Phone Emulator and, 29
- TestLambdas function, 873
- TestLifecycle solution, 42
- TestMapsTasks solution, 739
- TestMediaElement solution, 204
- TestMediaHelpers solution, 206
- TestMediaHub solution, 227
- TestMediaPlayer solution, 203
- TestMethod attribute, 474
- TestMobileServices app, 346
- TestNavigating solution, 59
- TestObscured solution, 45
- TestResumePolicy app, 44
- TestShakeHelper solution, 252, 255
- TestSoundEffect solution, 211
- TestUriMapping solution, 71
- TestVideo solution, 208
- TextBlock controls, 326, 461
- TextBox controls, 472
  - {Binding} syntax, 142
  - clipboard and, 230–231
- TextInput event, 526
- TextInput event (Phone 7), 526
- TextInputStart event, 526
- TextInputStart event (Phone 7), 526
- TextInputUpdate event, 526
- TextInputUpdate event (Phone 7), 526
- text-to-speech. *See* TTS (text-to-speech)
- textural data, 444
- Texture memory usage, 489
- texture resource views, 935
- textures, 933
- themes
  - native code and, XI–XXVI
- threads
  - background, 276
  - handled by Windows 8 vs. Phone, 907
- Thumb objects, 527
- Ticks property (.NET DateTime class), 408
- tile functionality, 512
  - enhancing, 513
- TileLightUp solution, 511
- TileLightUpViewModel class, 512
- tile push notifications
  - description/constraints on, 560
  - push notification clients and, 565
- tiles, 539–559
  - AppExtra element, registering, 511–512
  - Back-button behavior and pinned tiles, 554
  - checking for OS version when adding, 517–519
  - cycle tiles, 541
  - file-types for, 546
  - flip tiles, 539–540
  - for camera extension UI, 656
  - iconic tiles, 542
  - lighting up Windows Phone 7.x apps and, 510–520
  - OnNavigatedTo method, 548
  - Phone 7 and, 557–558
  - pinning, 550–557
  - pinning and the DetailsPage, 552
  - PinTiles\_Home solution, 554
  - PinTiles solution, 550
  - recommended sizes for, 546
  - resizing, 515
  - secondary, 546–558
  - SecondaryTiles app example, 546
  - ShellTile.Create method, 547
  - ShellTileData-derived classes, 549
  - ShellTile.Update method, 540
  - sizes/templates, 539–546

- TileSizes app sample, 545
  - updating with GBAs, 373–377
  - upgrading from Phone 7.x, 513–517
- TileSizes app sample, 545
- Tiles subfolder, 26
- TileViewModel, 513, 514
- TimeBetweenUpdates property (Accelerometer class), 241
- Timestamp, 307
- Title attribute (WMAppManifest.xml), 201
- toast notifications
  - CBEs and, 742
  - Obscured events and, 45
- toast push notifications
  - description/constraints on, 560
  - push notification clients and, 565
- Todoltem objects, 347
- ToDoList\_CustomSpeechRecognizerUI solution, 769
- ToDoList solution, 759
- ToDoList\_TTS solution, 777
- Token, 307
- tombstoning, 35, 38–43, 328
  - IsApplicationInstancePreserved property (Application.Activated), 40
  - NFC and, 692
  - page creation order, 49
  - testing, 485–486
  - testing on emulator, 591
- tools
  - AETGenerator, 852
  - Capabilities Detection, 282
  - DataSvcUtil, 326, 337
  - Isolated Storage Explorer, 455
  - MDILXAPCompile, 853
  - MemoryDiagnosticsHelper, 492
  - Mobile Device Management (MDM), 847
  - OData client, 326
  - PerformanceProgressBar, 504
  - submission, 446
  - Visual Studio Add Service Reference, 318, 319
  - Windows Phone Application Analysis, 487
- “touch” display type, 295
- TouchDevice property (TouchPoint), 133
- touch events, 124–126
  - handled by Windows 8 vs. Phone, 905
  - handling, 125–126
  - Mouse events as, 132–133
- TouchPointCollection, 134
- TouchPoints, 134
- TPL. *See* Task Parallel Library (TPL)
- TrailReminders app, 354
- TransactionHistoryNavigationUri property, 802
- Transact-SQL commands, phone-based LINQ-to-SQL and
  - , 423
- transient application state, 52
- transient page state, 52
- transient panels, 113–121
- TransientPanels solution, 114
- transient visuals, 114
- transit passes, managing with Wallet Hub, 796
- Transmission Control Protocol (TCP), 667
- Transport Layer Security (TLS), 339
  - Common Name (CN), 585
- transaction items (Wallet), 796–806
- trial mode (apps), 823–829
  - upgrading to full version of, 829
- TrialModeApp solution, 823
- Trident rendering engine, 17
- TriggeredConnectedStateChanged event, 683
- TriggeredConnectionStateChanged event (Peer-Finder service), 688
- triggering reminders, 481
- trusted platform API, 823
- TryGetValue method (IsolatedStorageSettings), 398
  - QueryString and, 68
- TryStart method (GeoCoordinateWatcher), 710
- TTS (text-to-speech), 776–781
  - keeping in synch with screen, 779–781
  - SpeakTextAsync method, 777
  - SpeechSynthesizer class, 777
  - ToDoList\_TTS solution, 777
- Tweet class, 300
- Twitter, 273
  - Access Token, 302
  - Consumer Key, 302
  - Consumer Secret, 302
  - create app, 302
  - custom scheme, 303
  - new tweet, 312
  - OAuth 1.0a protocol, 300
  - operations, 312
  - request string, 309
  - Request Token, 302
  - REST service, 300
  - TwitterWriter app, 304
- Twitter APIs
  - alternatives, 302
  - connect, 302

## TwitterHelper class

- header keys, 307
- parameters, 307
- secured, 305
- TwitterHelper class, 305
- TwitterHelper.GetRequestTokenAndLoginUrl method, 306
- TwitterReader app, 300
- Twitter SDK, 300–312
- TwitterWriter app, 304
- Type classes, 517
- type inference, 871–872
- Type objects, 517
  - reusing, 518
- Types view, 501
- type/value converters, 161–163
- typography, 5

## U

- UbuDu, 804
- UDP vs. TCP, 668
- UIElementColor method (UISettings class), XII
- UI elements, 77–88
  - screen layout, 84–88
  - standard, 77–81
  - visual tree, 81–84
- UISettings class, XII
- UI thread frame rate, 489
- UnhandledException handlers, 42
- unhandled exceptions, 453, 478
  - Application.Terminate event, 42
- uninstall/reinstall, 455
- Unique pointers, 870
- United States Geological Survey, 427
- unit testing, 472–480, 912
  - framework, 472
- UnitTestSystem class, 474
- Universal Product Code (UPC), 789
- Universal Volume Control (UVC), 12, 227, 385
- unlocking (emulator), 481
- unmanaged phones, 849–854
- Unobscured events, 45–47, 481
  - ApplicationIdleDetectionMode, 47
  - TestObscured solution, 45
- Update method (Direct3D), 950–954
- Update method (XNA FrameworkDispatcher), 212
- UpdateSubresource method (ID3D11DeviceContext), 951
- UpdateVideosForUri method, 483

- updating
  - artwork, 454
  - clean-and-rebuild, 455
  - descriptions, 454
  - keywords, 454
  - pricing, 454
  - regional availability, 454
  - REST, 323
  - SOAP, 323
  - XAP, 454
- UpgradeTilesToFlipTemplate method, 515, 516
- upgrading
  - limits, 515
  - primary tiles, 513
  - secondary tiles, 519
  - tiles, vs. adding, 517
- Upload page, 451
- URI classes and XML serialization, 375
- URI mappers, custom, 196
- UriMapper (System.Windows.Navigation namespace), 71
- UriMapping (System.Windows.Navigation namespace), 71
- Uri property (WebBrowserTask), 191
- URIs
  - app activation, handling, 74–75
  - association restrictions on, 74
  - associations, 72–75
  - formatting for pages in separate assemblies, 64
  - handlers for, 73–76
  - MappedUri property, 71
  - mapping, 71
  - reco parameter, 761
  - sending across devices, 702–705
  - starting apps based on, 72–73
  - TestUriMapping solution, 71
  - voiceCommandName parameter, 761
  - Windows Runtime and, 887
  - writing to NFC tags, 699–700
- URI schemes, 468
  - ms-appx, 760
- URLs, 296
- UserControls
  - built-in styles and, 928–930
  - cross-platform development and, 930–931
  - custom vs., 89–92
  - sharing code with, 927–932
- UserData APIs, 594
  - required references for, 595
- User Datagram Protocol (UDP), 667

- user experience (UX), 515
- UserExtendedProperties class, 524
- UserIdleDetectionMode and Obscured events, 46
- user input
  - events, 497
  - incorporating in Direct3D apps, 959–961
  - native code and, XVII–XXVI
- user interface (UI), 3–7, 77–138, 276. *See* user interface
  - app bar, 108–113
  - apps, 6–7
  - controls, UserControl vs. custom, 89
  - custom speech recognition and, 769
  - design principles for, 4–6
  - FrameReported events, 133–134
  - keyboard inputs, 135–137
  - ListBox, 276
  - manipulation events, 126–132
  - Mouse events, 132–133
  - persisting data, 58
  - personalization of, 5
  - resources for, 94–99
  - re-templating controls for, 92–94
  - routed events, 121–124
  - system tray, 108–113
  - tough gestures, 124–126
  - transient panels, 113–121
  - View (MVVM pattern), 174
- %USERPROFILE% environment variable, 318
- user standards, 467
- UserToken property (SocketAsyncEventArgs class), 677

## V

- ValidatesOnExceptions property ({Binding} syntax), 166
- validating behavior, 482
- VCD (Voice Command Definition) file, 756–765
  - initializing, 759–760
  - PhraseLists, 766–767
  - schema for, 758–759
- Vector3 type (AccelerometerReading), 240
- Verifier Pin, 302
- Version class, 307, 513
- version column and LINQ-to-SQL updates, 433
- versions, multiple, 459
- VertexShaderOutput, 953
- VerticalAlignment (StackPanel), 116
- VerticalCenterElement, 527
- VerticalThumb, 527
- vertices, 933
- VibrateController, 690
- VideoBrush element, 634
- videos
  - capturing, 649
  - controls, 283
  - embed, 282
  - H.264, 282
  - in WebBrowser, 283
  - video button, 283
- ViewModel (MVVM pattern), 174
  - persisting objects between pages with, 51
  - push, 579–584
- View (MVVM pattern), 174
- viewport, 935
- views
  - Caller/Callee, 503
  - Detailed Analysis, 498
  - GC Roots, 501
  - native profiler report, 503
  - Types, 501
- Virtual IP Address (VIP), 344
- Virtual-PC, 317
- Visibility property (transient windows), 115
- Visual Basic
  - compiler, 516
  - and Windows 8 vs. Phone, 911
- Visual C++ 11, 867
- VisualStateManager, 932
- Visual Studio 2012. *See* Microsoft Visual Studio 2012
- Visual Studio Add Service Reference tool, 318, 319
- Visual Studio Solution Explorer
  - data resources and, 94
- Visual Studio template, 79
- visual tree, 81–84
  - app bar and, 110
  - best practices for, 505
  - GlobalElementChange solution, 83
  - XAML resources and, 96
- VisualTreeHelper class, 82
- Voice Command Definition files. *See* VCD (Voice Command Definition) file
- voice commands, 753–767
  - accuracy of, vs. speech recognition, 770
  - building apps for, 756–765
  - initializing VCD files, 759–760
  - invoking, 761–765
  - labels, 762–765

## VoiceCommands element (VCD schema)

- PhraseLists, 766–767
- registering, 756–759
- VCD files, 756–765
- VoiceCommands element (VCD schema), 758
- VoiceCommandService class, 759
- voice element (SSML), 779
- Voice over IP (VoIP), 910
- Volume property (MediaElement class), 204

## W

- W3C standards
  - SRGS (speech recognition), 773
  - SSML, 778
- WaitForMessage method (ConnectionReceived event), 672
- Wallet, 785–810
  - adding/removing accounts for, 802–806
  - adding/removing accounts within apps, 797–802
  - AddWalletItemTask, 799
  - agents, 806–809
  - checking state of Deals in, 795–796
  - deals, managing, 787–796
  - Hub, 786
  - ID\_CAP\_WALLET capability, 787
  - NFC and, 785–786
  - object model of, 787–806
  - OnlinePaymentInstrument class, 806
  - payment instruments in, 806
  - testing, 797
  - transaction items in, 796–806
  - unlinking cards, 805
- WalletAgent background agent, 806–809
- Wallet Hub, 786
  - membership/loyalty cards and, 796
- WalletItem class, 787
  - CustomProperties dictionary, 790
  - SetUserAttentionRequiredNotification method, 808
- WalletTransactionItemBase class, 787
- WalletTransactionItem class, 796–806
- WalletTransactionItem objects, 801
- warnings
  - performance, 500
- WCF. *See* Windows Communication Foundation (WCF)
- WCF data services, 325–338
  - caching data with, 333–336
  - dynamic query results with, 331–332
  - filtering queries in, 329–331
  - JSON-formatted data and, 337–338
  - paging data in, 332
  - to publish data, 325
  - Windows Store apps vs., 569
- WCF service solution, 317
- WCF Service Web Role, 341
- WCF web service, 317
- WebBrowser control, 281–286
  - create apps using, 281
  - JavaScript, integrating with, 284–286
  - local webpages and, 282–284
  - in Phone 7.x vs. Phone 8, 526
  - scripting, 286
  - update, 282
  - videos in, 283
  - vs. desktop version, 281
  - Windows Runtime API and, 911
- WebBrowser.InvokeScript method, 284
- WebBrowser.IsScriptEnabled property, 284
- WebBrowser.Navigated event, 296
- WebBrowser.ScriptNotify event, 284
- WebBrowserTask, 188
- WebClient class, 273–281, 312
  - asynchronous call pattern, 278
  - Async Pack, 278–280
  - BitmapImage, 277
  - BTS vs., 358
  - create instance, 274
  - operations, 276
    - DownloadStringAsync, 276
  - Rest, 324
  - Task Parallel Library (TPL), 280–281
  - usage, 274–278
  - use with REST, 324
  - vs. HttpRequest, 273
  - Windows Phone version, 274
  - WriteableBitmap, 277
- WebClient.DownloadStringAsync, 300
- web connectivity, 273–314
  - Data Sense API, 312–314
  - Direct3D apps and, XXIV–XXVI
  - Facebook C# SDK, 294–300
  - HttpRequest class, 273–281
  - Microsoft Silverlight WebBrowser, 273
  - native code and, XXIV–XXVI
  - Twitter SDK and, 300–312
  - WebBrowser control, 281–286
  - WebClient, 273
  - WebClient class, 273–281

- Windows Live SDK and, 287–294
- WebExceptions, HTTP failures, 469
- web registration service
  - PushSimpleClient\_Registration sample, 572
  - RegisterCompleted event, 572
- web roles, 340
- web search grammar, 771–772
- web services, 315–348
  - authentication, 339
  - connecting to, 321–323
  - localhost and, 317–321
  - security for, 339–340
  - SOAP vs. REST, 323–325
  - testing in emulator, 317–321
  - WCF data services, 325–338
  - Windows Azure, 341–345
- WideBackBackgroundImage property, 515
- WideBackContent (flip tiles), 515, 540
- WideBackgroundImage property, 515
- WideCharToMultiByte function, 884, 885
- Wi-Fi Direct protocol, 695
- Win32 APIs, 407–411
  - calling from managed apps, 8
  - IsolatedStorageFile vs., 399
  - legacy code and, 407
  - Windows Runtime vs., 898–900
- Win32CreateFile method, 409
- Win32OpenFile method, 410
- Win32ReaderWriter solution, 408
- WINAPI\_FAMILY\_APP symbol, 921
- Windows 8, 482
  - app types supported by, 902
  - code sharing with Phone 8, 911–932
  - FileOpenPicker class and, X
  - framework differences with Phone 8, 908
  - HTTP activation, 320
  - Phone UI as snapped view, 931–932
  - tap-to-connect and, 694–695
  - Windows Phone 8 and, 901–932
  - Windows Runtime API in, 905–908
  - Windows Runtime sockets in, 668
- Windows Advanced Rasterization Platform (WARP), 949
- Windows authentication (NTLM), 339
- Windows Azure, 340–348
  - API, 340
  - mobile services for Windows Phone, 345–348
  - registration web service and, 569
- WindowsAzure.MobileServices.Managed.dll, 346
- Windows Azure SDK, 340
- Windows Communication Foundation (WCF), 315
  - administrator access, requirement for, 571
- Windows Communications Foundation (WCF) Data Services, 569
  - DataServiceCollection<T> class, 161
  - registration web services, 569
- Windows.Devices.Sensors namespace, 238
- windows.foundation.h, 889
- Windows Imaging Component (WIC), 936
- Windows Intune, 847
- WindowSizeDependentResources method, 945
- Windows Live, 287
  - connecting to, with Live SDK, 287–294
  - REST API, 289
- Windows namespace, 904
- Windows Phone 7, 317, 708
  - Bing Maps service, 712–721
  - cross-targeting for, 557
  - GeoCoordinateWatcher class, 708–712
  - IPv4 support in, 670
  - projects, vs. Windows Phone 8, 326
  - trial versions and, 826
  - vs. Windows Phone 8, 317
- Windows Phone 7.1
  - IAP in, 830
  - photo apps picker, 651–653
  - rear camera requirements and, 636
- Windows Phone 7.8 SDK, 536
- Windows Phone 7.x apps, 278, 509–538
  - Anonymous ID (ANID) (UserExtendedProperties class), 524
  - API overlap with Windows Phone 8, 908
  - DataContext constructor and, 528
  - FM radio hardware support, 524
  - functionality no longer supported in Phone 8, 521–528
  - LayoutUpdated event, 526
  - lighting up, 509–519
  - LongListSelector control and, 528
  - quirks mode, 520–521
  - simulating app update from, 519
  - simulating OS upgrade for, 519
  - Slider control in, 527
  - testing coverage for, 537
  - text input events and, 526
  - WebBrowser control in, 526
  - XML in, 528
- Windows Phone 8, 317
  - API surface, 904–909
  - apps, delivery methods for, 20–21

- architecture, 8–18
- backstack management, 61
- backup URI handlers in, 73
- capability detection, 16
- code sharing with Windows 8, 911–932
- Company Apps menu (Settings), 847–849
- Continuous Background Execution (CBE) app, 741–747
- Core System and, 16–17
- enterprise-focused support in, 845–846
- extensibility points, 7
- framework differences with Windows 8, 908
- GSE help icon and, 755
- lighting up Phone 7 apps, 558
- Maps API, 727–740
- namespaces, 904–905
- Phone 7.1 libraries, sharing, 530–531
- projects, 21–31
- projects, vs. Windows Phone 7, 326
- screen resolutions supported by, 544
- supported CLSIDs in, 899
- UI as snapped view in Windows 8, 931–932
- UX guidelines, 34
- vs. Windows Phone 7, 317
- Windows 8 and, 16–18, 901–932
- Windows Phone 7.x functionality no longer supported in, 521–528
- Windows Phone Runtime, 18
- Windows Runtime API in, 905–908
- Windows Runtime programming mode/API set, 17–18
- XNA app support in, 10
- Windows Phone 8 SDK, 18
  - CBEs, permitted APIs for, 742
- Windows Phone app guidelines
  - home buttons, 554
  - unpinning tiles programatically, 551
- Windows Phone Application Analysis tool, 487
- Windows Phone Application Certification Requirements, 452
- Windows Phone Application Marketplace, 20
- Windows Phone Application Store, 20
- Windows Phone Audio Playback Agent project, 384
- Windows Phone chassis specification, 125
- Windows Phone Class Library project, 531
- Windows Phone Dev Center, 444
  - IAP and, 832–835
  - pubCenter and, 820
  - submitting apps as beta to, 841
- Windows Phone Developer Tools, 108
- Windows Phone emulator, 18, 486–487
  - keyboard shortcuts for, 486
  - speech recognition and, 756
  - system requirements for, 19–20
  - testing on device vs., 487
- Windows.Phone namespace, 904
- Windows Phone Performance Analysis Tool, 493–502
  - app analysis report, 495–499
  - memory analysis report, 500–502
- Windows Phone Runtime, 18
- Windows Phone Runtime component (Visual Studio), 925
- Windows Phone SDK, 318, 319
  - debugging mixed-mode projects with, 888
  - Isolated Storage Explorer (ISE), 401
  - play icon, location of, 777
  - Reactive Extensions and, 242
  - Simulation Dashboard, 481
  - standard XAML resources in, 96
- Windows.Phone.Speech.Synthesis.SpeechSynthesizer class, 777
- Windows.Phone.Speech.VoiceCommands.VoiceCommandService class, 759
- Windows Phone Store, 439
  - apps
    - icon, 450
  - certification, 36
  - Company Apps feature vs., 855
  - excluding devices with, 81
  - graphical data, 444
  - IAP and, 830–831
  - icon
    - apps, 450
  - image requirements, 441
    - application screenshot 720p, 441
    - application screenshot WVGA, 441
    - application screenshot WXGA, 441
    - store tile, 441
  - keywords, 449
  - Launchers, 455
  - NFC tags and, 702
  - publication process, 440
  - rules, 440
  - test cases, 440
  - textural data, 444
  - trial mode, 823–829
  - Wallet app, registering as, 804

- WCF services vs., 569
  - Windows Phone Dev Center, 444
  - Windows Push Notification System (WNS), 558
  - Windows Phone style resource guidelines, 84
  - WINDOWS\_PHONE symbol, 921
  - Windows Phone Toolkit
    - visual tree controls/tools in, 83
  - Windows.Phone.UserInformation namespace, 605
  - “Windows Phone XAML and Direct3D app” template (Visual Studio), 644
  - Windows Push Notification System (WNS), 558
  - Windows Runtime API, 522
    - authoring components on Phone 8, 911
    - C++ code and, 878–888
    - Component Object Model (COM), 18
    - consuming components of, from C#, 885–887
    - creating components from, 880–885
    - immutable interfaces, 905
    - mapping to .NET types, 887
    - native code and, 878–888
    - .NET sockets vs., 675
    - portability/performance of, 883
    - Win32 vs., 898–899
    - Windows 8 vs. Phone 8, 905–908
  - Windows Runtime sockets API, 667, 668–675
  - Windows Runtime storage API, 402–407
    - app lifecycle and, 405
    - await keyword, 403, 406
    - IStorageFile interface, 402
    - IStorageFolder interface, 402
    - large data sets, persisting, 407
    - ms-appx scheme, 403
    - ProductID property, 404
  - WindowsRuntimeStorageReaderWriter solution, 402
  - WindowsRuntimeStringManipulator, 884
  - Windows Runtime String type, 410
  - Windows.Storage namespace, 402
  - Windows.System.Threading.ThreadPool, 505
  - Winmd (Windows 8), 909
  - WinSock API, 667, 680
  - wireless networking
    - Bluetooth, finding peers with, 683–686
    - endianness and, 679
    - NFC and, 696–705
    - peer-to-peer socket communication, 680–695
    - sockets, 680–695
  - wl.basic scope (LiveSDK), 288
  - wl.offline\_access scope (LiveSDK), 288
  - wl.photos scope (LiveSDK), 292
  - wl.signin scope (LiveSDK), 288
  - wl.skydrive scope (LiveSDK), 292
  - WMAAppManifest graphical editor, 543
  - WMAAppManifest.xml, 25
    - CBE requirements for, 741
    - Extensions section, 195
    - graphical manifest editor (Visual Studio), 711
    - GUI vs. raw XML editor, 26
    - ID\_CAP\_LOCATION capability, 711
    - ID\_CAP\_MAP capability, 727
    - ID\_CAP\_MICROPHONE capability, 768
    - ID\_CAP\_SPEECH\_RECOGNITION capability, 768
  - worker roles, 340
  - WP8 conditional compilation symbol, 533
  - WP71Method, 536
  - WPToolkitTestFx, 473
  - WrapOrientation solution, 234
  - WrapPanel control (Windows Phone Toolkit), 234–236
  - wrapper functions and asynchronous calls, 738
  - WriteableBitmap class (System.Windows.Media.Imaging.Extensions namespace), 547, 630
    - in HttpWebRequest, 277
    - in WebClient, 277
  - WriteableBitmapEx library, 548
  - WriteEndElement method (XmlWriter), 575
  - WriteStartElement method (XmlWriter), 575
  - wstring (string type), 410
  - WVGA (screen resolution), 78
    - Direct3D games and, XV
    - page sizes for, 80
    - performance and, 544
    - targeting, 467
  - WXGA (screen resolution), 78, 544
    - devices, 464
    - Direct3D games and, XV
    - page sizes for, 80
    - targeting, 467
- ## X
- XAML
    - element binding and, 164
    - Facebook, 294
    - type/value converters and, 163
  - XAML apps, 10
    - code sharing and, 927–928
    - Direct3D in, 937–939
    - model, 8
    - profiling, 502
  - XAML hierarchy, 81–82

- XAML layout
  - Grid, 505
  - StackPanel, 505
- XAML parse error, 98
- XAML resources, 95–99
  - resolving references to, 97
- XAML template, 527
- XAP
  - resubmitting, 444
  - updating, 454
  - validation, 440
- Xbox, DataPackage type and, III
- Xbox Live, 276
- XML
  - in Phone 8 vs. Phone 7, 528
  - type/value converters and, 163
  - Uri classes, serialization of, 375
  - vs. JSON, 337–338
  - XDocument, 575
- XmlSerializer, 375
  - IsolatedStorageFile and, 400
- XNA, 466
- XNA app type, 10
- XNA FrameworkDispatcher, 212
- XNA Game class, 212
- XNA game loop, 212
- XNA Microphone class, 214–223
- XNA SoundEffect classes. *See* SoundEffect classes
- X-NotificationClass, 574
- XP. *See* extreme programming

## Y

- yaw, 262
- Yelp, 787
- “yes/no” strings, 462

## Z

- ZoomLevel property (GeoCoordinate), 714
- ZoomLevel property (Map control), 728
- Zune media queue (ZMQ), 384
- ZXing.Net, 789

# About the Authors



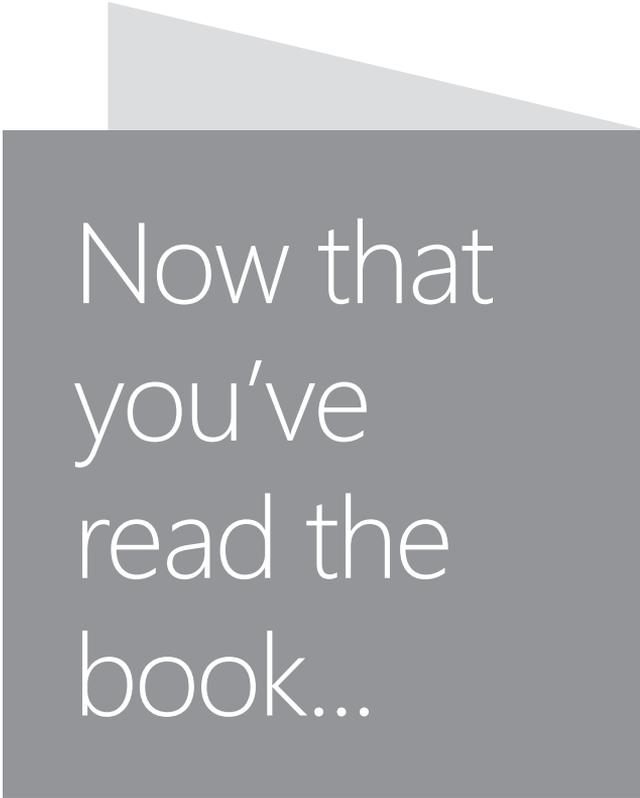
**ANDREW WHITECHAPEL** is a senior program manager for the Windows Phone Developer Platform team, responsible for internal components within the platform. He is the author of several books, including *Windows Phone 7 Development Internals* (Microsoft Press, 2012).



**SEAN MCKENNA** is a senior program manager on the Windows Phone Developer Platform team. He has been responsible for several key features, including local database support and app-to-app communication.







Now that  
you've  
read the  
book...

Tell us what you think!

Was it useful?

Did it teach you what you wanted to learn?

Was there room for improvement?

**Let us know at <http://aka.ms/tellpress>**

Your feedback goes directly to the staff at Microsoft Press,  
and we read every one of your responses. Thanks in advance!

