Windows Phone 8

UNLEASHED

FREE SAMPLE CHAPTER

SHARE WITH OTHERS











Daniel Vaughan

Windows® Phone 8

UNLEASHED

Windows® Phone 8 Unleashed

Copyright © 2013 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33689-8 ISBN-10: 0-672-33689-8

Library of Congress Control Number: 2013936246

Printed in the United States of America

First Printing May 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Windows is a registered trademark of Microsoft Corporation.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales 1-800-382-3419 corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales international@pearsoned.com

Editor-in-Chief Greg Wiegand

Executive Editor
Neil Rowe

Development Editor Mark Renfrow

Managing Editor Sandra Schroeder

Senior Project Editor Tonya Simpson

Copy Editor Barbara Hacha

Indexer Ken Johnson

Proofreader
Debbie Williams

Technical Editor
J. Boyd Nolan

Publishing Coordinator Cindy Teeters

Book Designer Anne Jones

Compositor

Jake McFarland

Contents at a Glance

	Preface	xxii
Part I	Windows Phone App Development Fundamentals	
1	Introduction to Windows Phone App Development	1
2	Fundamental Concepts in Windows Phone Development	13
3	Understanding the Application Execution Model	47
4	Controlling Page Orientation	99
Part II	Essential Elements	
5	Using Content Controls, Items Controls, and Range Controls	. 115
6	Mastering Text Elements and Fonts	. 153
7	Employing Media and Web Elements	. 183
8	Taming the Application Bar	. 231
9	Enriching the User Experience with the Windows Phone	
	Toolkit Controls	. 253
10	Creating Jump Lists with the Long List Selector	. 303
11	Creating Expansive and Engaging Apps with the Pivot and Panorama	. 325
Part III	Windows Phone App Development	
12	Processing Touch Input	.357
13	Portraying Your App with Live Tiles	. 389
14	Leveraging Built-In Apps via Launchers and Choosers	. 405
15	Receiving Messages from the Cloud Using Push Notification	. 485
16	Sensing Motion and Attitude	. 525
17	Building Location Aware Apps	. 559
18	Incorporating Map-Based Positioning	. 585
19	Supporting Multiple Cultures and Languages	. 613
20	Extending the Windows Phone Picture Viewer	. 631
21	Capturing Images and Video with the Camera	. 655
22	Recording Audio with the Microphone	. 691
23	Creating Speech Driven Experiences	. 699
24	Unit Testing Apps	. 737
25	Manage Payment Instruments Using Wallet Extensibility	.773

Part IV	Building Data Driven Apps	
26	Validating User Input	805
27	Communicating with Network Services	849
28	Preserving App State and Settings	881
29	Storing App Data in a Local Database	905
30	Auto-Launching with File and Protocol Associations	967
31	Extending the Search Experience	983
Part V	Multitasking	
32	Conducting Background Activities with Scheduled Actions	997
33	Performing Background File Transfers	1043
34	Coordinating Background Audio Playback	1059
	Bibliography	1081
	Index	1083

Table of Contents

	Preface	xxii
art I	Windows Phone App Development Fundamentals	
1	Introduction to Windows Phone App Development	1
	Installing the Windows Phone SDK	2
	Creating Your First XAML for Windows Phone App	2
	The Application Class	7
	Summary	11
2	Fundamental Concepts in Windows Phone Development	13
	Understanding the Role of XAP Files	14
	The Application Deployment Tool	
	The Windows Phone Capabilities Model	16
	Determining App Capabilities Using the Marketplace Test Kit	17
	The Threading Model for XAML-Based Graphics and Animation	
	in Windows Phone	18
	Performance and Element Visibility	19
	Deciding Between Visibility and Opacity	20
	Understanding the Frame Rate Counter	20
	The Windows Phone Application Analysis Tool	22
	External Events	25
	Frame Rate Graph	25
	CPU Usage Graph	25
	Memory Usage MB Graph	26
	Storyboards	26
	Image Loads	26
	GC Events	26
	Viewing Detailed Profiling Information	26
	Reading Device Information	27
	Calculating Available Memory	28
	DeviceStatus Events	29
	Applying the Model-View-ViewModel Pattern to a Windows	
	Phone App	31
	Implementing the MVVM Pattern	31
	ViewModelBase Class	32
	Property Change Notification	33
	Implementing INotifyPropertyChanged: The Traditional	
	Approach	33

	implementing inotifyPropertyChanged: All Alternative	
	Approach	
	Using Commands	
	Argument Validation	39
	A Platform-Agnostic Dialog Service	41
	Consuming Local Web Applications	44
	Summary	46
3	Understanding the Application Execution Model	47
	Exploring the Execution Model	48
	Application State	49
	Life Cycle Events	49
	Programmatically Exiting an App	53
	Saving Transient State	53
	Restoring Transient State	55
	Saving Persistent State	56
	Implementing Fast App Resume	56
	Enabling Fast App Resume	56
	Optimizing the Resume Experience	57
	Running Under the Lock Screen	63
	Lock Screen Management	64
	Page Navigation	
	Navigation Using Unmapped URIs	69
	Passing Page Arguments Using Query Strings	71
	URI Mapping	73
	Navigation Using the NavigationService	75
	Handling Page Navigation	76
	Canceling Navigation	77
	Cross-Page Communication	77
	Page Redirection	78
	Hardware Back Button	79
	Creating an Application Splash Screen	80
	Walking Through the Bookshop Sample Application	83
	Displaying the Product List	86
	Design-Time Data	92
	Image Caching	93
	Overview of the Sample Bookshop WCF Service	95
	Summary	97
4	Controlling Page Orientation	99
	Orientation and the PhoneApplicationPage Class	99
	OrientationChanged Event	100

	PhoneApplicationPage Orientation Property	102
	Setting Page Orientation at Runtime	105
	Animating Page Elements When the Page Orientation Changes	106
	Animating the Entire Page When Orientation Changes	108
	Windows Phone Toolkit Animated Page Transitions	110
	Using Windows Phone Toolkit Transitions	111
	Reusing the Transition Attached Properties	113
	Summary	114
Part II	Essential Elements	
5	Using Content Controls, Items Controls, and Range Controls	115
	Control Type Taxonomy	116
	Identifying Controls Not Supported or Absent in Windows	
	Phone	118
	Content Controls	119
	Defining the Default Content Property	120
	Receiving Input with Buttons	121
	Tap and Click Events	123
	Button Click Mode	
	Hyperlink Button	
	Repeat and Toggle Buttons	
	Radio Button	128
	Check Box	
	ToolTip	
	Items Controls	138
	ListBox	
	ComboBox	
	Range Controls	
	ProgressBar	
	Progress Indicator	
	Slider	
	ScrollBar	
	Summary	152
6	Mastering Text Elements and Fonts	153
	Understanding Text Element Types	
	Displaying Text Using the TextBlock.	
	Using the Run and LineBreak Objects	
	Multiline TextBlock Text	
	Controlling Font Properties	
	Understanding the Built-In Fonts	160

	Using Third-Party Fonts via Font Embedding	162
	Leveraging FontSource to Assign a Font Using a Stream	164
	Receiving Input Using a TextBox	165
	Software Input Panel	165
	Input Scope	168
	Input Scope Sample Overview	169
	Receiving Password Text Using a PasswordBox	175
	Displaying Rich Content Using a RichTextBox	176
	Formatting Text at Runtime	178
	Further Reading	179
	Placing Text on the Clipboard	179
	Summary	180
7	Employing Media and Web Elements	183
	Displaying Images with the Image Element	184
	Image Sizing	185
	Providing a Drawing Surface with the InkPresenter Element	186
	Capturing User Input	188
	InkPresenter Sample Code	190
	Playing Audio and Video with the MediaElement	193
	MediaElement Sample Code	194
	XNA SoundEffect Class	201
	Harnessing the XNA Game Loop Via the GameTimer Class	202
	XNA Environment Initialization	203
	Saving Music to the Media Library	205
	Viewing High-Resolution Images with the MultiScaleImage Element	207
	Image Tiling	208
	Creating a Deep Zoom Image	208
	Exploring the MultiScaleImage API	210
	Deep Zoom Viewer Sample Code	212
	Displaying Web Content with the WebBrowser Element	217
	Monitoring and Canceling Navigation	217
	Communicating with a Web Page	218
	Injecting Web Page Behavior	222
	Storing and Browsing Content in Isolated Storage	225
	Summary	228
8	Taming the Application Bar	231
	Exploring the Built-In Application Bar	231
	Using the ApplicationBar	232
	Modifying the Application Bar's Appearance	234
	Icon Button and Menu Item	234

	Icon Button Image	235
	Retrieving a Button or Menu Item at Runtime	236
	Introducing the Custom AppBar	237
	AppBar Buttons and Menu Items	237
	Button and Menu Item Visibility	238
	Sample Code	238
	Switching to Full-Screen	244
	Customizing the Appearance of the System Tray	247
	Inside the AppBar Control	248
	Summary	252
9	Enriching the User Experience with the Windows Phone Toolkit Controls	253
	Getting Started with the Toolkit	254
	In-Place Selection Using the ListPicker	255
	Display Modes	255
	Suggesting Text with the AutoCompleteBox.	259
	Using AutoCompleteBox with MVVM	260
	Defining a Custom Filter Using the ItemFilter Property	264
	Dynamic Population of the Suggestion List	266
	Styling the Suggestion List	268
	Data Binding and the AutoCompleteBox.Text Property	270
	Displaying a ContextMenu	271
	ContextMenu and MVVM	273
	Hosting a ContextMenu in a ListBoxItem	275
	Selecting Date and Time Values with the DatePicker and TimePicker.	277
	Using the DatePicker and TimePicker	279
	Adding a Control Header	281
	Customizing the Value Format	281
	Customizing the Full-Screen Picker Page	282
	Providing Custom Looping Lists with the LoopingSelector	287
	Providing Touch Feedback with the TiltEffect	291
	Supporting Other Controls	292
	Providing a Boolean Option with the ToggleSwitch	293
	Using the ToggleSwitch	294
	Localizing the ToggleSwitch	295
	Changing the Switch Color	297
	Using Flexible Layouts with the WrapPanel	297
	Child Element Spacing	298
	Sample Overview	
	Using a WrapPanel in Conjunction with a ListBox	300
	Summary	302

10	Creating Jump Lists with the Long List Selector	303
	Visual Structure of the Long List Selector	304
	Presenting Flat Lists	305
	Presenting Grouped Lists	307
	Marketplace App List Sample	307
	Displaying a List of Contacts Using the LongListSelector	316
	Summary	323
11	Creating Expansive and Engaging Apps with the Pivot and Panorama	325
	Pivot and Panorama Differences and Similarities	326
	Style Assumptions	326
	Introduction to the Pivot Control	327
	Introduction to the Panorama Control	327
	Pivot and Panorama Placement in the FCL	328
	Performance Improvements in Windows Phone 8	329
	Using the Pivot Control	329
	Placing a Pivot on a Page	329
	Creating a Pivot Application with the Visual Studio	
	New Project Dialog	331
	Pivot Anatomy	331
	Setting the Active PivotItem	
	Pivot Load Events	
	Hosting Multiple Application Bars with the Pivot	
	Populating a Pivot Via a Data Bound Collection	
	Using the Panorama Control	
	Panorama Anatomy	
	PanoramaItem Control	
	Sample Panorama Application	
	Things to Avoid When Using the Panorama and Pivot	
	Windows Phone Toolkit Lockable Pivot	
	Summary	355
Part III	Windows Phone App Development	
12	Processing Touch Input	357
	Handling Touch with Mouse Events	358
	Touch and TouchPoint Classes	360
	Mouse Event Promotion	361
	Handling the Touch.FrameReported Event	362
	Manipulation Events	363
	Handling Manipulation Events	364
	Manipulation Events Example	366

	UIElement Touch Gesture Events	368
	Tap Gesture	369
	Double Tap Gesture	370
	Hold Gesture	371
	Windows Phone Toolkit Gestures	372
	Getting Started with Toolkit Gestures	373
	GestureListener Events in Detail	373
	GestureBegin and GestureCompleted Events	380
	Gesture Sample Code	381
	Designing Touch-Friendly User Interfaces	386
	Three Components of Touch	
	Sizing and Spacing Constraints	387
	General Guidelines	387
	Summary	388
13	Portraying Your App with Live Tiles	389
	Getting Started with Live Tiles	
	Tile Sizes and Resolutions	391
	Introducing the Cycle Tile Template	391
	Introducing the Flip Tile Template	394
	Introducing the Iconic Tile Template	395
	Programmatically Updating Shell Tiles	396
	Controlling the Lock Screen's Background and Notification Area	397
	Setting the Lock Screen's Background Image	
	Setting the Lock Screen's Notification Text	401
	Creating a Lock Screen Icon	
	Summary	404
14	Leveraging Built-In Apps via Launchers and Choosers	405
	API Overview	405
	Choosers and the Application Execution Model	407
	Internal Workings of the ChooserBase.Completed Event	407
	Launchers and Choosers in Detail	408
	Retrieve Driving Directions Using the BingMapsDirectionsTask	409
	Displaying a Location on a Map Using the BingMapsTask	413
	Navigating to a Connection Setting Page Using the	
	ConnectionSettingsTask	413
	Selecting an Email Address with the EmailAddressChooserTask	414
	Preparing an Email with the EmailComposeTask	417
	Saving a Contact's Email Using the SaveEmailAddressTask	418
	Navigating to an App on the Marketplace with the	
	MarketplaceDetailTask	422
	Launching the Marketplace App with the MarketplaceHubTask	425

	Allowing the User to Review Your App Using the	
	MarketplaceReviewTask	426
	Searching the Marketplace with the MarketplaceSearchTask	427
	Playing a Media File Using the MediaPlayerLauncher	428
	Placing a Call with the PhoneCallTask	435
	Selecting a Phone Number with the PhoneNumberChooserTask	437
	Saving a Contact's Phone Number with the	
	SavePhoneNumberTask	439
	Searching the Web with the SearchTask	442
	Sending Contacts a Link Using the ShareLinkTask	444
	Posting a Status Update to Social Networks Using the	
	ShareStatusTask	
	Preparing an SMS With the SmsComposeTask	
	Navigating to a Web Page Using the WebBrowserTask	446
	Selecting a Contact's Address Using the AddressChooserTask	447
	Saving a Contact to the Phone's Contact List Using the	
	SaveContactTask	
	Taking a Photo with the CameraCaptureTask	
	Inviting Game Players with the GameInviteTask	453
	Selecting a Photo from the Photo Hub Using the	
	PhotoChooserTask	454
	Sharing Images with Other Devices Using NFC or with	
	Other Registered Phone Services	460
	Creating a Ringtone with an Audio File Using the	4.61
	SaveRingtoneTask	
	Contacts and Appointments	
	Retrieving Contacts	
	Storing Contacts in the Custom Contact Store	
	Detecting Changes in Stored Contacts	
	Converting a Stored Contact to and from a vCard	
	Retrieving Appointments. Saving an Appointment Using the SaveAppointmentTask	
	Summary	
	Summary	404
15	Receiving Messages from the Cloud Using Push Notification	485
	Push Notifications Types	485
	Benefits of Push Notification	486
	Understanding Push Notification	487
	Getting Started with Push Notification	489
	Subscribing to Push Notification	489
	Binding to the Shell	490
	HttpNotificationChannel Events	491
	Handling Channel Errors	492

	Power Management and Push Notification	493
	Sending Push Notifications	495
	Toast Notifications	495
	Receiving a Toast Notification from Within an Application	496
	Sending a Toast Notification	497
	Tile Notifications	500
	Sending a Tile Notification	501
	Updating an Application Tile Using a Shell Tile Schedule	504
	Raw Notifications	505
	Sending a Raw Notification	505
	Receiving a Raw Notification	508
	Identifying Notifications in an HttpWebResponse	509
	Notification Classes	509
	Cloud Service Authentication	511
	Authenticating a Cloud Service	511
	Creating a Notification Channel for an Authenticated	
	Cloud Service	511
	Building a Stock Ticker Application	511
	Sample App Notifications	520
	Summary	524
16	Sensing Motion and Attitude	525
	Sensors Overview	525
	Measuring Force with the Accelerometer	527
	Using the Accelerometer Class	528
	Simulating Acceleration with the Emulator	529
	Smoothing Accelerometer Readings	530
	Calibrating the Accelerometer	533
	Shake Detection	537
	Measuring Direction with the Compass	539
	Using the Compass Sensor	539
	Compass Orientation	545
	Calibrating the Compass	546
	Sensing Rotation with the Gyroscope	548
	Using the Gyroscope Sensor	549
	Improving Sensor Accuracy with the Motion Sensor	552
	Using the Motion Sensor	553
	Summary	557
17	Building Location Aware Apps	559
	Location Sensing Technologies	559
	A-GPS	560
	Cell Tower Triangulation Wi-Fi Triangulation	

	Geographic Location Architecture	562
	Getting Started with Location	563
	Geolocator Class	563
	Background Location Tracking	569
	Testing Apps That Use the Geolocator	570
	Code-Driven Location Simulation	571
	A Walkthrough of the Location Viewer Sample	575
	GeoLocationViewModel Class.	575
	Displaying Location Using the GeoLocationView Page	577
	Civic Address Resolution	580
	Sampling the PositionChanged Event with Rx	580
	Getting Started with Rx for Windows Phone	580
	Summary	584
18	Incorporating Map-Based Positioning	585
	Getting Started with the Map Control	586
	Obtaining an ApplicationId and AuthenticationToken	587
	Overview of the Sample Map View	587
	Adjusting for Low Light Conditions	588
	Changing Cartographic Modes	589
	Panning and Zooming the Map Control	590
	Centering the Map to the Phone's Current Location	590
	Setting the Viewable Area of the Map	592
	Adjusting Pitch and Heading	593
	Location Tracking	594
	Overlaying Custom Map Content	596
	Calculating and Displaying a Route	598
	Calculating the Shortest Route Between Two Points	598
	Using the Route Calculator	601
	Searching for a Route Using the View	602
	Modifying Page Elements Using Visual States	604
	Displaying the Route and Itinerary	609
	Showing a Location with the Built-In Maps App	
	Showing Directions with the Built-In Maps App	611
	Downloading Maps for Offline Use	612
	Summary	612
19	Supporting Multiple Cultures and Languages	613
	Terminology	613
	Localizability Using Resx Files	614
	Getting Started with Resx Files	615
	Working with Resx Files	615

	Dynamic Localizability—Updating the UI When the Culture Changes.	
	Localizing Images Using Resx Files	
	The Resx Localizability Sample	
	Controlling the UI Culture from the LocalizabilityViewModel	. 622
	Displaying Localized Text and Images Within the	
	LocalizabilityView Page	
	RTL Support	
	Summary	. 630
20	Extending the Windows Phone Picture Viewer	631
	Creating a Photos Extras Application	. 632
	Adding Your App to the Extras Menu	. 634
	An Edge-Tracing Extras Application	. 634
	Saving the Image	. 643
	Share Menu Extensibility	. 645
	Adding Your App to the Share Menu	. 646
	A Simple Photo Upload Share Application	. 647
	Summary	. 654
21	Capturing Images and Video with the Camera	655
	PhotoCamera	. 656
	Building a PhotoCamera App	. 658
	Using the Silverlight Webcam API	. 678
	CaptureSourceViewModel	. 680
	Displaying Video in the CaptureSourceView	. 684
	Extending the Camera Experience with a Lens	. 686
	Registering as a Lens	. 687
	Creating Lens Icons	. 689
	Summary	. 689
22	Recording Audio with the Microphone	691
	Recording Audio with the Microphone	. 691
	Creating a Helium Voice App	. 692
	Summary	. 698
23	Creating Speech Driven Experiences	699
	User Input with Speech Recognition	. 700
	Getting Started with Speech Recognition	. 700
	Overview of Recognition Grammars	
	Recognizing Speech Using the Dictation and Web	
	Search Grammars	. 702
	Recognizing Speech with Phrase List Grammars	. 703

	Recognizing Speech with SRGS Grammars	710
	Speech Recognizer Settings	717
	Providing a Custom Speech Recognition UI	718
	Recognizing Speech in Other Languages	721
	Launching Your App via Voice Commands	723
	Understanding the Structure of the VCD File	724
	Installing a VCD	726
	Updating a Phrase List Programmatically	
	Making the Phone Speak with Text-to-Speech	
	Getting Started with TTS	
	Creating a Speech Synthesis Markup Language File	
	Summary	735
24	Unit Testing Apps	737
	Automated Testing	738
	Unit Testing	738
	Integration Testing	
	Coded UI Testing	
	Introduction to the Windows Phone Unit Test Framework	
	Creating a Test Project	
	Creating a Test Class	
	Tag Expressions	
	Setting the Tag Expression Programmatically	
	Metadata and Assertions	
	TestClass Attribute	
	TestMethod Attribute	
	Metadata for Test Initialization and Cleanup	
	Miscellaneous Metadata	
	WorkItemTest: The Base TestClass Type	
	Verifying Conditions with Assertions	
	Verifying Collection Conditions with CollectionAssert	
	Verifying String Conditions with StringAssert	
	Hiding the Expressions Editor	
	Testing Multiple Assemblies	
	Testing Nonpublic Members	
	A Testable Chat Client	
	Building the View	
	Code Driven UI Testing	
	Using Automation Peers to Manipulate UI Elements at Runtime	
	Inversion of Control (IoC)	
	A Custom IoC Container and DI Framework	
	Testing Trial Conditions	
	Abstracting the LicenseInformation Class	767

	Testing with Launchers and Choosers	769
	Summary	771
25	Manage Payment Instruments Using Wallet Extensibility	773
	Introducing the Wallet Hub	774
	Required Capabilities for Wallet Integration	775
	Storing Payment Instruments in the Wallet	775
	Overview of the Payment Instruments Sample App	777
	Keeping Payment Instruments Up-To-Date with a	
	Background Agent	789
	Storing Membership Information in the Wallet	792
	Storing Deals in the Wallet	796
	Keeping Deals and Membership Information Up-to-Date	
	with a Background Agent	802
	Summary	804
Part IV	Building Data Driven Apps	
26	Validating User Input	805
	Defining Input Validation	805
	Syntactic Validation	806
	Semantic Validation	806
	Input Validation Using Property Setters	806
	Validation Class	809
	Critical Exceptions	809
	Binding Errors	809
	Defining Validation Visual States in Windows Phone	810
	Validating a TextBox as the User Types	815
	Performing Group Validation	818
	Displaying Error Details	820
	Property Setter Validation Limitations	823
	Asynchronous and Composite Validation	824
	A Reusable Implementation of the NotifyDataErrorInfo	
	Interface	825
	Provisioning for Asynchronous or Synchronous Validation	827
	Asynchronous Validation of All Properties	833
	An Example of Asynchronous Input Validation	836
	Detecting a Change of Data Context	840
	Adding INotifyDataErrorInfo Support to the	
	ValidationSummary Control	842
	Incorporating Group Validation	844
	Summary	847

27	Communicating with Network Services	849
	Network Service Technologies	850
	Monitoring Network Connectivity	850
	Connection Priorities	850
	Introduction to OData	855
	Consuming OData	855
	OData URI Structure	856
	Generating an OData Proxy	858
	OData Query Options	858
	Using an OData Proxy	860
	Building an eBay OData Consumer Application	862
	Creating an OData Wrapper	862
	EbaySearchViewModel Class	864
	EbaySearchView Page	
	Fetching Data When the User Scrolls to the End of a List	871
	Extending OData Entity Classes	
	Simulating Real-World Conditions with the Simulation Dashboard	876
	Moderating Data Usage with Data Sense	877
	Using the Data Sense API	
	Summary	879
28	Preserving App State and Settings	881
	Understanding Managed Storage	882
	Working with Files and Directories	883
	Introducing the .NET Isolated Storage API	884
	Introducing the WinPRT Windows Storage API	887
	Measuring the Amount of Available Free Space	888
	Application Settings	888
	Abstracting IsolatedStorageSettings	
	Building an Automatic State Preservation System	
	Customizing ViewModel State Preservation	
	Automatic State Preservation Inner Workings	
	Unwinding a Property Lambda Expression	901
	Creating Property Accessor Delegates	
	Summary	903
29	Storing App Data in a Local Database	905
	SQL Server Compact	906
	Deployment of Local Databases	906
	Deployment of Local Databases.	
	LINQ to SQL on the Phone	
	± ,	908

	Getting Started with Local Databases	909
	Code-First Data Model Creation	909
	Sample Twitter Timeline Viewer	910
	Using the Column Attribute	912
	Data Context Change Tracking	915
	TwitterUser Class	915
	Multiplicity and the Association Attribute	917
	Twitter DataContext	918
	Database Utilities	919
	Connection Strings	920
	Leveraging a Custom Twitter Service	924
	Gathering the User's Credentials with the Sign-In View	929
	Viewing Tweets with the Timeline View	932
	Viewing a Local Database Schema	937
	File Explorers for Isolated Storage	938
	Viewing and Modifying an SQL CE Database File	940
	Database-First Using SqlMetal	942
	Deploying a Database to Isolated Storage	
	Abstracting the Navigation Service	946
	Observing LINQ to SQL Queries with a Custom Log	948
	Updating a Database Schema	
	AddColumn	952
	AddTable	952
	AddIndex	954
	AddAssociation	
	Schema Versioning	
	Mapping an Inheritance Hierarchy	
	Concurrency	
	ColumnAttribute.UpdateCheck	
	Summary	
	,	
30	Auto-Launching with File and Protocol Associations	967
	Understanding File and Protocol Associations	968
	A Note About the Sample	
	Auto-Launching with a File Extension Association	
	Registering a File Association	
	Receiving a File Launch Request	
	Launching a File	
	How File and Protocol Associations Affect the User Experience	
	Auto-Launching with a Protocol Association	
	Registering for a Protocol Association	
	Receiving a Protocol Launch Request	

	Launching a Protocol URI	979
	Reserved Protocol Names	981
	Launching Built-In Apps	981
	Summary	982
31	Extending the Search Experience	983
	Understanding Quick Cards	983
	Configuring Your App for Search Integration	985
	Configuring the Application Manifest	985
	Customizing Captions via the Extras.xml File	987
	Defining a Quick Card Target Page	989
	Creating a Landing Page to Display Quick Card Information	994
	App Instant Answer	996
	Summary	996
Part V	Multitasking	
32	Conducting Background Activities with Scheduled Actions	997
	Background Tasks	998
	Scheduled Notifications	998
	Alarm Registration	1001
	Alarm Sample	1002
	Reminder Registration	1005
	Reminder Sample	1006
	Scheduled Tasks	1010
	Background Agent Types	1011
	Using Scheduled Tasks	1015
	To-Do List Scheduled Task Sample	1019
	Updating Tiles Using a Scheduled Task Agent	1035
	Scheduled Task API Limitations	1036
	Using a Mutex to Access Common Resources Safely	1038
	Summary	1041
33	Performing Background File Transfers	1043
	Background Transfer Requests	1043
	Handling App Termination and Resubscription to	
	Transfer Events	
	Background File Transfer Sample Code	1048
	Using URL Rerouting with a WCF Service	1049
	Retrieving the User's Windows Live Anonymous ID	1050
	TodoListViewModel	1052

	Backing Up the Local Database	. 1052
	Restoring the Local Database	. 1055
	Summary	. 1058
34	Coordinating Background Audio Playback	1059
	Background Agent Recap	. 1059
	Background Audio Overview	. 1060
	Background Audio Player	. 1060
	Representing Audio Files with the AudioTrack Class	. 1062
	Creating a Custom Audio Player Agent	. 1063
	AudioPlayerAgent Sample	. 1064
	AudioPlayerAgent Virtual Methods	
	Controlling Background Audio from Your Foreground App	. 1070
	Audio Streaming Agents	. 1076
	Using a MediaStreamSource to Play Back an Assembly Resource	. 1078
	Summary	. 1079
	Bibliography	1081
	Index	1083

Preface

Welcome to the second in the series of *Windows Phone Unleashed* titles. This book extends the previous book, *Windows Phone 7.5 Unleashed*, and provides new content covering the terrific new features of the Windows Phone 8 SDK. The Windows Phone 8 SDK builds upon the strong foundation of the previous SDK for creating XAML-based apps. If you read *Windows Phone 7.5 Unleashed*, you will find much of the content familiar. The content has, however, been updated across the board to cover changes in the 8.0 SDK, of which there are many. You also will find seven new chapters covering the new features of the 8.0 SDK.

Although the scope of this book remains squarely on building XAML-based apps, there have been some major changes to the other UI technologies supported by Windows Phone 8. The new SDK still enables you to build XNA UI apps for Windows Phone 7.1 devices, but support has been discontinued for Windows Phone 8 apps. And although Windows Phone 7.1 apps can be downloaded from the Windows Phone Marketplace and run on Windows Phone 8 devices, Windows Phone 7.1 apps cannot use the new features found in the 8.0 SDK. For high-performance games, Microsoft now encourages the use of Direct3D and C++ for native code, which was not available in Windows Phone 7.1. Although XNA is no longer directly supported for developing Windows Phone 8 apps, you can use third-party frameworks, such as MonoGame (www.monogame.net), to create XNA-based WP8 apps. However, this is outside the scope of this book.

As Microsoft continues to converge its various platforms, the Windows Phone SDK now contains some APIs that overlap with Windows 8's WinRT. Windows Phone does not inherit all these new APIs, and in fact, has its own new set of partially overlapping APIs known as WinPRT. No prizes for guessing what the 'P' stands for.

The environmental requirements for developing Windows Phone 8 apps are now restricted to Windows 8 x64 and Visual Studio 2012. And Windows Phone 8 apps do not run on older devices running Windows Phone 7.1. You can, however, develop Windows Phone 7.1 apps using the Windows Phone 8.0 SDK.

Scope of This Book

This book targets Windows Phone 8 (Apollo). Although you see some examples incorporating XNA for audio and media, this book's focus is squarely on XAML for Windows Phone. The book covers all main areas of the topic in a deep, yet easily comprehensible way, using practical examples with a real-world context. The goal is to provide you with concepts and techniques that will help you to design and develop well-engineered and robust Windows Phone apps.

Throughout this book you see a small number of techniques and custom code applied to make developing phone apps easier. It is not the intention to make what you will learn in the book harder to reach; on the contrary, the techniques are tried and tested approaches that, when they become familiar, will help you build more testable and maintainable apps

that can be potentially ported to other platforms. The competition between apps on the Windows Phone platform has intensified as the number of apps in the Windows Phone Marketplace has increased dramatically. This competition not only brings with it a "long tail," where independent developers find evermore niche categories to create apps for, but also requires apps competing in the more popular categories to increase their feature sets. As apps become more complex, maintainability comes to the fore, and greater attention to managing complexity is required.

This book is not a book for those without at least some knowledge of XAML. Although there is considerable reference material for some essential XAML infrastructure, included within these chapters are advanced topics, such as the Model-View-ViewModel design pattern (MVVM). In fact, most sample apps follow the MVVM pattern. The concepts and techniques used throughout the book are described in Chapter 2, "Fundamental Concepts in Windows Phone Development." Do not worry if some of these approaches seem foreign to you; by the end of the book they will be second nature.

Wherever possible, you are provided with tips and techniques that go beyond the topic, and you will frequently find content not easily found elsewhere. A substantial amount of custom code is provided that extends the Windows Phone SDK to support real app scenarios.

Assumptions About the Reader

If you are an experienced developer who has basic experience in Silverlight, WPF, or Windows 8 XAML apps looking to transfer your skills to Windows Phone, then this book is for you. It is assumed that you are familiar with C#, XAML, and Visual Studio.

Book Structure

The book is divided into five parts:

- ▶ Part I, "Windows Phone App Development Fundamentals"
- ▶ Part II, "Essential Elements"
- ▶ Part III, "Windows Phone App Development"
- ▶ Part IV, "Building Data Driven Apps"
- ▶ Part V, "Multitasking"

Most chapters have sample apps. Chapter 2 is required reading to understand the techniques used throughout the book and the samples.

Some chapters, in particular Chapter 9, "Enriching the User Experience with the Windows Phone Toolkit Controls," are not intended to be read from beginning-to-end, but rather are intended as a reference that you may refer back to when you need to learn about a particular topic within the chapter.

Code Samples

To demonstrate each concept, this book contains more than 100 samples. The sample code for this book can be downloaded from www.informit.com/title/9780672336898.

All code is in C#. The project structure is divided into topic areas. To view a particular sample, you can run the main solution and select the sample page from the index (see Figure 1).



FIGURE 1 The sample code index page.

In the downloadable sample code are several solutions. In most cases, the Vaughan.WPUnleashed.sln is used. The code for some topics, however, has been placed into separate solutions because of technical constraints.

Much of the infrastructure code presented in the book has been consolidated into the Calcium open source project. You can find more information about the Calcium SDK at http://calciumsdk.net.

About the Author



Daniel Vaughan is cofounder of Outcoder, a Swiss software and consulting company dedicated to creating best-of-breed connected user experiences and leading-edge back-end solutions, using the Microsoft stack of technologies—in particular WPF, Windows 8, WinRT, Silverlight, and Windows Phone.

Daniel is a Microsoft MVP for Client Application Development, with commercial experience across a wide range of industries, including finance, e-commerce, and multimedia. Daniel is a Silverlight and WPF Insider, a member of the elite WPF Disciples group, a three-time CodeProject MVP, and is a member of the Microsoft Developer Guidance Advisory Council.

He is a technical advisory board member of PebbleAge, a Swiss finance company specializing in business process management.

He also is the creator of a number of open source projects, including Calcium and Clog.

Daniel blogs at http://danielvaughan.org, where he publishes articles and software prototypes. You also can catch Daniel on Twitter at @dbvaughan.

Daniel has a degree in Computer Science from UNE, where he received various awards, including twice the annual School of Mathematics, Statistics and Computer Science Prize, and the Thomas Arnold Burr Memorial Prize in Mathematics.

With his wife, Daniel runs the Windows Phone Experts group on LinkedIn at http://linkd.in/jnFoqE.

Originally from Australia and the UK, Daniel is based in Zurich, Switzerland, where he lives with his wife, Katka, and son Bertie.

Dedication

To my wonderful wife, Kateřina and my son, Bertie.

Acknowledgments

Katka Vaughan, for endless advice, proofreading, and formatting. Your contribution and unending patience made this book possible.

Peter Torr, for invaluable assistance on Microsoft forums.

Microsoft, for answering questions and building great tools.

The terrific team at Sams. Especially Neil Rowe, for guidance throughout the entire process. Technical editor J. Boyd Nolan, for going over my code with a fine-toothed comb.

The great folks at PebbleAge. In particular, Olivier Parchet and Christian Kobel for their encouragement and goodwill.

Inspiration and support (in no particular order):

Cliff Simpkins, John Papa, Davide Zordan, Jaime Rodriguez, Marlon Grech, Glenn Block, Charles Petzold, Josh Smith, Sacha Barber, David Anson, Peter O'Hanlon, Corrado Cavalli, Colin Eberhardt, René Schulte, Laurent Bugnion, Jeff Wilcox, Damian Mehers, Shawn Wildermuth, Sascha Corti, Rob Tiffany, Walt Ritscher, Michael Crump, Matteo Pagani, Atley Hunter, Ginny Caughey, my friends at Wintellect, Jeremy Likness, Steven Porter, Todd Fine, and Jeff Prosise.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

E-mail: feedback@samspublishing.com

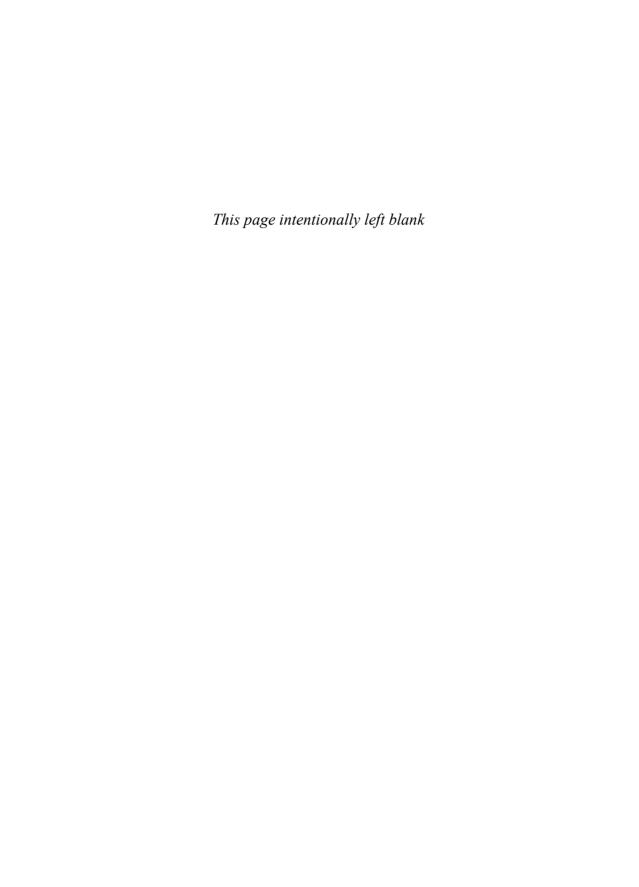
Mail: Sams Publishing

ATTN: Reader Feedback 800 East 96th Street

Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.



CHAPTER 2

Fundamental Concepts in Windows Phone Development

This chapter provides an overview of some common pieces of phone infrastructure and describes various techniques that are useful when building Windows Phone apps.

This chapter begins with a look at the principal output of a Windows Phone Application project: the XAP file. The chapter discusses its composition and shows how to deploy a XAP file to a developer unlocked device.

The chapter then looks at the security capability model of the phone and at how this model is used to notify the user of any potential dangers before downloading an app from the Windows Phone Marketplace. You also look at using the Marketplace Test Kit to determine the capability requirements of your app.

Next, the chapter examines the threading model of XAML for Windows Phone and examines various performance considerations when creating animations or hiding and showing visual elements. You see how the Windows Phone frame rate counter works and learn how to interpret each reading shown on the display.

The chapter then looks at the Windows Phone Application Analysis tool. You see how to profile your app's performance and memory usage, improve the responsiveness of your app, and help ensure that your app passes the Windows Phone Marketplace certification requirements.

The chapter turns to the custom code and commonly used techniques that you see throughout the book and that underpin many of the examples in subsequent chapters.

It is not uncommon to have many pages in a Windows Phone app, and having a solid codebase that contains a

IN THIS CHAPTER

- ► Understanding the role of XAP files
- ► Using the Application Deployment tool
- ► Enabling and identifying app capabilities using the Marketplace Test Kit
- ► The threading model for XAML graphics and animation
- Optimizing animation performance
- ► Enabling and reading the Frame Rate Counter
- Using memory and execution profiling to identify bottlenecks
- ► Retrieving device information using the DeviceStatus class
- ► Applying MVVM to a Windows Phone app
- Simplifying property change notification
- ► Using commanding with MVVM
- ► Validating method arguments
- ▶ Building a platform agnostic dialog service
- ► Enabling communication with local WCF services

common infrastructure and frequently used services can save a lot of time. In fact, with more than 100 example pages included in the downloadable sample code, creating the code for this book would have taken considerably longer without it.

The techniques demonstrated are tried and tested approaches that help you build more maintainable apps and, by the end of the book, will have become exceedingly familiar to you if they are not so already.

The overview of the custom infrastructure begins with an exposé of the Model-View-ViewModel pattern, and you see how it is applied in the downloadable sample code. You then examine how property change notification is implemented and see techniques for improving the traditional implementation of INotifyPropertyChanged so that it works effortlessly with multithreaded apps.

Next, the chapter looks at the commanding infrastructure used throughout the book and examines a custom ICommand that allows you to specify handlers for command evaluation and execution. There is also a brief overview of the argument validation system commonly used in the sample code to validate method arguments.

The chapter then explores a custom dialog service that enables you to ask the user a question from your viewmodel, while remaining compatible with unit testing.

Finally, the chapter shows how to consume Windows Communication Foundation (WCF) services that reside on the same machine as the emulator, and outlines important steps to enable several of the apps in the downloadable sample code.

Understanding the Role of XAP Files

The output of a Visual Studio project normally consists of a multitude of files, which may include assemblies, images, config files, manifest files, and so forth. XAP (pronounced *zap*) files contain project output that is bundled up, ready for deployment.

XAP files have been around since the early days of Silverlight 2 (beta 1) and allow developers to easily deploy an entire Silverlight application to a remote server. On the Windows Phone platform, they are used to deploy an app to the Windows Phone Marketplace or to a developer unlocked device.

A XAP file is a compressed zip file that contains your project assemblies and resources, along with two application manifest files: AppManifest.xml and WMAppManifest.xml, both of which are located in the Properties directory of the project.

NOTE

It is a certification requirement that the XAP file contains both an AppManifest.xml file and a WMAppManifest.xml file. Both of these files are automatically generated when creating a new Windows Phone application from within Visual Studio. In nearly all cases, AppManifest.xml does not require changes by you. WMAppManifest.xml, however, may require editing depending on the features supported by your app.

When publishing to the Windows Phone Marketplace, your app's XAP file is submitted as part of the publishing process.

To obtain the XAP file for your app, perform a build using a Release build configuration. You can then find the XAP file located in your app's Bin/Release directory.

NOTE

When submitting your application to the Windows Phone Marketplace, the XAP file that you submit must be built using a release configuration without debug information or it may fail the certification process.

NOTE

The maximum allowed size of the XAP package file for Windows Phone Marketplace certification is 225MB.

The Application Deployment Tool

XAP files allow you to circulate your app to developers that have a developer unlocked device. This is done using the Application Deployment tool that is installed along with the Windows Phone SDK. The tool allows you to navigate to select a XAP file and deploy it to a connected phone device (see Figure 2.1).

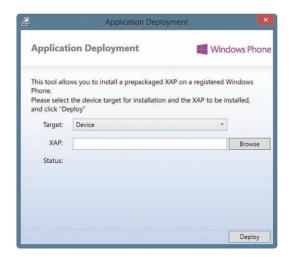


FIGURE 2.1 The Application Deployment tool.

If you have not encountered XAP files before, they will certainly become relevant when you want to publish your first app to the Windows Phone Marketplace.

The Windows Phone Capabilities Model

Microsoft recognizes that making the user experience on the phone the best it can be helps to secure greater adoption of the platform. To this end, users should never regret installing an app, and one way to ensure this is by using a security model that requires users to opt-in to certain functionality within the app, called capabilities.

A capability is a phone resource that when used by your app may bring with it privacy or security concerns, or it may incur a cost that the user should be made aware of. Examples of capabilities include the camera, geographic location services, microphone, and SMS.

Capabilities are a way of disclosing to the user what an app is potentially able to do. Your app's capabilities are displayed to potential users, those considering downloading the app from the Windows Phone Marketplace. It is at the user's discretion whether to download your app; if a user does not want an app to have access to, for example, the phone's camera, the user may decide not to download that app.

You define your app's capabilities in its WMAppManifest.xml file. When a new Windows Phone application is created, a subset of the available capabilities is included by default in the manifest file. The Windows Phone operating system grants security permissions to the application according to the capabilities listed in the manifest file. See http://bit.ly/Pj2YgE for a list of these capabilities.

When an app is submitted to the Windows Phone Marketplace, the XAP file is decompressed, validated, and repackaged. During this process the security capabilities of the app are discovered and written back to the WMAppManifest.xml file. As a result, if the manifest does not contain capabilities that are used by your app, these capabilities are inserted as part of the submission process.

NOTE

The capabilities specified in the WMAppManifest.xml file before submission are relevant only while debugging your app. By removing unnecessary capabilities from the manifest you ensure that no unintended capabilities have crept in during development.

Two capabilities, however, are exceptions to this process: ID_CAP_NETWORKING and ID_HW_FFCCAMERA.

If the ID_CAP_NETWORKING (networking) capability is removed from your app's manifest, it will not be reinserted during the submission process; this enables you to prevent all network activity from your app if you want.

If the ID_HW_FFCCAMERA (front facing camera) capability is specified in your manifest file, it is not automatically removed during the submission process.

NOTE

After submission to the Windows Phone Marketplace, and during the capability discovery process, the Microsoft Intermediate Language (MSIL) of the assemblies located in your XAP file are analyzed. If a phone API that requires a particular capability is detected, the capability is added to the WMAppManifest.xml file. This occurs even if your app never

calls the code at runtime. It is therefore important to be mindful that referencing another assembly can inadvertently add security capabilities to your app if the other assembly uses an API that requires capabilities. The security capability detection mechanism is not clever enough to walk your MSIL to discover whether it is actually used; it merely identifies the presence of the API.

NOTE

To pass Windows Phone Marketplace certification, apps are not allowed to use P/Invoke or COM Interop.

Determining App Capabilities Using the Marketplace Test Kit

The Windows Phone SDK includes a tool for analyzing the capabilities required by your app. To analyze your app's capability requirements, perform the following steps:

- **1.** Build the app using a Release configuration.
- **2.** Right-click the Windows Phone app project's node in the Visual Studio Solution Explorer and select Open Store Test Kit (see Figure 2.2).
- **3.** Click the Run Tests button to begin the analysis process.

NOTE

The Capabilities Validation test that is present for Windows Phone 7.1 apps is not available for apps that target Windows Phone 8.

NOTE

The Iconography and Screenshots automated tests fail if you have not specified any icons or screenshots for your app.

The Marketplace Test Kit offers numerous tests that can assist you in ensuring that your app is Marketplace ready. It can also save you time and the frustration caused by failing the Marketplace submission requirements.

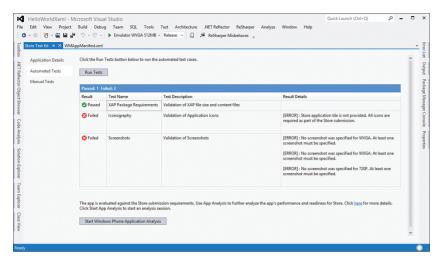


FIGURE 2.2 Store Test Kit.

The Threading Model for XAML-Based Graphics and Animation in Windows Phone

XAML apps use two threads for graphics and animation: a UI thread and a composition thread. The composition thread was introduced with the second release (7.5) of the Windows Phone OS. The first release of the OS had issues with performance around user input. A single UI thread had been largely acceptable for Silverlight for the desktop and browser because both generally rely on the mouse for input. The phone, however, relies on touch, which, as it turned out, needs to be substantially more reactive to user input. When using a mouse, a slight delay does not unduly affect the user's perception of your app, but when using touch, a slight delay can make the user feel like the device is broken. Thus, the composition thread was introduced in Windows Phone 7.5¹ to assist in rendering visuals by offloading some of the work traditionally done by the UI thread.

The UI thread is the main thread in Windows Phone XAML apps and handles user input, events, parsing, and creation of objects from XAML, and the initial drawing of all visuals.

The composition thread aides the UI thread in handling graphics and animation, freeing up the UI thread and making it more responsive to user input. Storyboard-driven animations that run on the composition thread are cached and handled by the device GPU in a process called autocaching.

^{1.} The composition thread was also introduced to Silverlight for the browser with Silverlight 5 but still lacks the autocaching capabilities present in Windows Phone. Autocaching is described in a moment.

NOTE

Although the composition thread frees the UI thread in some situations, the key to writing responsive apps is still making sure that the UI thread is not overloaded or blocked by user code—in event handlers, for example. If you anticipate that a particular section of code will tie up the UI thread for a considerable amount of time, for more than, say, 50 milliseconds, use a background thread to perform the activity. The web service APIs, for example, are all designed to be used asynchronously so that they do not block the UI thread.

The Windows Phone 8 SDK sees the inclusion of the new .NET async keyword that makes consuming asynchronous APIs far easier than it used to be.

If you are not familiar with the various mechanisms for spawning background threads, do not be concerned; you see many examples throughout the book.

The composition thread is used for animations involving the <code>UIElement</code>'s <code>RenderTransform</code> and <code>Projection</code> properties. Typically these animations include the following from the <code>System.Windows.Media</code> namespace:

- ▶ PlaneProjection
- ▶ RotateTransform
- ▶ ScaleTransform
- ▶ TranslateTransform

NOTE

The composition thread is used only for scale transforms that are less than 50% of the original size. If the scale transform exceeds this amount, the UI thread performs the animation. In addition, the <code>UIElement.Opacity</code> and <code>UIElement.Clip</code> properties are handled by the composition thread. If an opacity mask or nonrectangular clip is used, however, the UI thread takes over.

Animations and Threads

The composition thread is ideal for handling storyboard animations because it is able to pass them to the device GPU for processing, even while the UI thread is busy. Code-driven animations, however, do not benefit from the composition thread because these kinds of animations are handled exclusively by the UI thread, frame by frame using a callback. They are, therefore, subject to slowdown depending on what else occupies the UI thread, and the animation will update only as fast as the frame rate of the UI thread.

Performance and Element Visibility

XAML-based apps provide two properties that allow you to hide or reveal UI elements: UIElement.Visibility and UIElement.Opacity—each of which has performance implications depending on how it is used.

The UIElement.Visibility property is handled by the UI thread. When an element's Visibility property is set to Collapsed, the visual tree must be redrawn. The upside is that when collapsed, the UIElement is not retained in visual memory, and thus decreases the amount of memory used by your app.

Conversely, controlling the visibility of an element using the <code>UIElement.Opacity</code> property allows the element to be bitmap cached; the element is stored as a simple bitmap image after the first render pass. Bitmap caching allows the rendering system to bypass the render phase for the cached element and to use the composition thread to display the bitmap instead, which can free up the UI thread considerably. By setting the opacity of a cached element to zero, you hide the element without requiring it to be redrawn later. This, however, does mean that unlike the <code>Visibility</code> property, the element is still retained in visual memory.

NOTE

Avoid manipulating the $\tt UIElement.Opacity$ property without enabling bitmap caching. Set the $\tt UIElement.CacheMode$ property to BitmapCache, as shown in the following example:

```
<Path CacheMode="BitmapCache" ... />
```

Deciding Between Visibility and Opacity

Element opacity in conjunction with bitmap caching usually produces the best performance when hiding and revealing elements. There may be times, however, when the <code>UIElement.Visibility</code> property is better, and this is influenced by the number and complexity of the visual elements being rendered. In such cases it may require experimentation to determine the best approach.

Understanding the Frame Rate Counter

Developing for a mobile device requires particular attention to performance. Mobile devices have less computing power than desktop systems and are more susceptible to performance bottlenecks.

The Windows Phone SDK comes with a built-in control that allows you to monitor the performance of your app, including frames per second and memory usage.

By default, the frame rate counter is enabled in your app's App.xaml.cs file if a debugger is attached, as shown in the following excerpt:

```
if (System.Diagnostics.Debugger.IsAttached)
{
    // Display the current frame rate counters.
    Application.Current.Host.Settings.EnableFrameRateCounter = true;
...
}
```

NOTE

It is possible to enable or disable the frame rate counter programmatically at any time from your app.

The EnableFrameRateCounter property is somewhat of a misnomer because the control also reports a number of other UI metrics, such as texture memory usage, as shown in Figure 2.3.

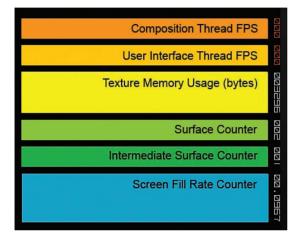


FIGURE 2.3 The Frame Rate Counter.

Each field is updated periodically while the app is running. Table 2.1 describes each counter field.

TABLE 2.1 Frame Rate Counter Field Descriptions

Field	Description
Composition (Render) Thread Frame Rate (FPS)	The rate at which the screen is updated. It also represents how often supported animations driven by a storyboard are updated. This value should be as close to 60 as possible. Application performance begins to degrade when this value is below 30. The text in this counter is red when displaying a value below 30.
User Interface Thread Frame Rate (FPS)	The rate at which the UI thread is running. The UI thread drives input, per-frame callbacks, and any other drawing not handled by the composition thread. The larger this value, the more responsive your application should be. Typically this value should be above 20 to provide an acceptable response time to user input. The text in this counter is red when displaying a value below 15.
Texture Memory Usage	The video memory and system memory copies of textures being used in the application. This is not a general memory counter for the application but represents only memory that surfaces use.

Field	Description
Surface Counter	The number of explicit surfaces being passed to the GPU for processing. The biggest contributor to this number is automatic or developer-cached elements.
Intermediate Surface Counter	The number of implicit surfaces generated as a result of cached surfaces. These surfaces are created in between UI elements so that the application can accurately maintain the Z-order of elements in the UI.
Screen Fill Rate Counter	The number of pixels being painted per frame in terms of screens. A value of 1 represents 480×800 pixels. The recommended value is about 2.5. The text in this counter turns red when displaying a value higher than 3.

Source: MSDN: http://bit.ly/l8i020

The frame rate counter is a valuable tool for identifying performance bottlenecks in your app. For more detailed performance metrics turn to the Application Analysis tool, discussed next.

The Windows Phone Application Analysis Tool

Not only is performance important in ensuring that your app provides an enjoyable experience for your users, but it is also important in a stricter sense: for meeting the certification requirements of the Windows Phone Marketplace. Marketplace certification includes a number of performance related criteria that your app must adhere to. The requirements are as follows:

- ▶ If an application performs an operation that causes the device to appear to be unresponsive for more than 3 seconds, such as downloading data over a network connection, the app must display a visual progress or busy indicator.
- ▶ An app must display the first screen within 5 seconds after launch. You see how to work around this requirement for slow loading apps by creating a splash screen, in Chapter 3, "Understanding the Application Execution Model."
- ▶ An app must be responsive to user input within 20 seconds after launch.

The Windows Phone Application Analysis tool comes with the Windows Phone SDK and is integrated into Visual Studio, allowing you to analyze and improve the performance of your apps. The tool profiles your app during runtime to gather either execution metrics or memory usage information.

Execution profiling may include method call counts and visual profiling, allowing you to view the frame rate of your app over time, while memory profiling allows you to analyze your app's memory usage.

To launch the tool select Start Windows Phone Application Analysis from the Debug menu in Visual Studio. You can select the profiling type, along with other advanced metrics, by expanding the Advanced Settings node, as shown in Figure 2.4.

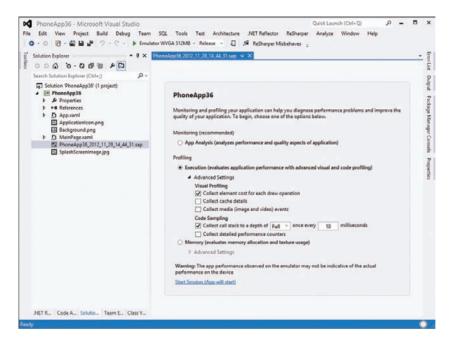


FIGURE 2.4 Configuring the Application Analysis settings.

To begin the profiling session, click the Start Session link.

Whenever the Application Analysis tool runs, it creates a .sap file in the root directory of your project. A .sap file is an XML file that contains the profiling information gathered during a profiling session and can later be opened by the profiling analysis tools built in to Visual Studio.

When done putting your app through its paces, click the End Session link, shown in Figure 2.5. You can, alternatively, use the device's hardware Back button to end the profiling session.

NOTE

Avoid disconnecting the phone device to end a profiling session, because this can lead to sampling errors. Instead, always use the End Session link or the hardware Back button.

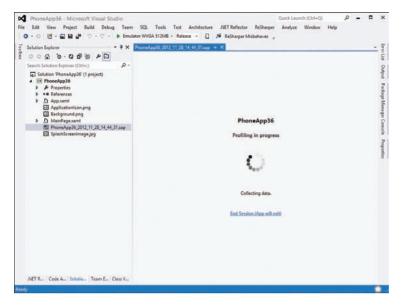


FIGURE 2.5 Profiling in progress with the Application Analysis tool.

Once stopped, the analysis tool automatically parses the .sap file and presents a summary of the analyzed data. Clicking the Alerts link presents a graph view (see Figure 2.6).

The .sap file can be reloaded into the analysis tools by double-clicking the .sap file in the Visual Studio Solution Explorer.

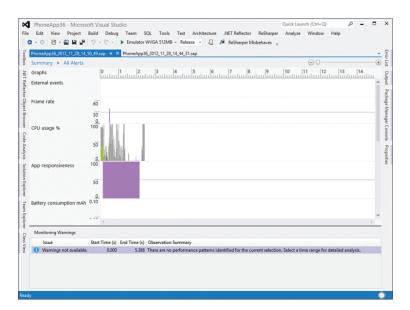


FIGURE 2.6 Viewing performance metrics in the Application Analysis tool.

Each section of the analysis tools view is discussed in the following sections.

External Events

External events indicate user events, such as UI input, or simulated network changes.

Frame Rate Graph

The Frame Rate graph displays the number of screen redraws (in frames per second) that the app completed at the particular point in the timeline.

NOTE

The Frame Rate graph shows nonzero values for periods in the timeline where the application was updating the display in some way. Therefore, areas that appear to have a zero frame rate indicate that no updating was taking place and not necessarily that your app was not able to render any frames.

NOTE

You should aim to have the frame rate value averaging between 30 and 60fps.

CPU Usage Graph

The CPU Usage graph displays the activity of various threads using different colors, as described in Table 2.2.

TABLE 2.2 CPU Graph Colors

Color	Thread	Notes
Green	UI Thread	Green shading indicates screen updates and touch input. You should aim to keep the UI thread to less than 50% of CPU usage.
Purple	App Threads	Purple indicates application activity that is not on the UI thread. Activity can be from the composition thread or from your apps background threads, such as those used from the AppPool.
Gray	System Threads	Gray indicates activity that is independent of your app, such as background agent activity.
White	Idle Threads	White indicates the available CPU percentage. The higher the idle thread percentage, the more responsive the app should be.

Memory Usage MB Graph

Memory Usage MB shows the amount of RAM being consumed by your app in megabytes, at any point along the timeline. This graph allows you to identify excessive memory usage.

Storyboards

Storyboards are displayed as an S flag on the timeline to indicate the occurrence of a storyboard event, and typically indicate the start of an animation. There are two kinds of flags: A red flag indicates a storyboard that is CPU bound; a purple flag indicates a storyboard that is not CPU bound.

Image Loads

When an image is loaded into memory, an I flag is displayed on the graph. Although JPG and PNG files might have a small size when stored in isolated storage, when displayed using an Image control for example, images are expanded into bitmaps and consume a lot more memory. Use the image load flag to identify places in your app where excessive memory consumption is taking place.

GC Events

When the CLR performs garbage collection, a G flag is displayed on the graph. Garbage collection reclaims memory and ordinarily decreases the value shown in the Memory Usage MB graph.

Viewing Detailed Profiling Information

Within the analysis tool, a region can be selected within the graph to view detailed performance warnings for that period. Much like Visual Studio's Error List view, the Performance Warnings view identifies three types of items: Information, Warning, and Error items (see Figure 2.7).

The Observation Summary provides advice on how to rectify each particular warning item.

The CPU Usage breadcrumb can also be expanded and allows you to view various other CPU-related metrics, such as a function call tree.

The Application Analysis tool provides detailed runtime performance metrics and allows you to identify the source of performance bottlenecks, enabling you to improve the responsiveness of your app and in turn the user experience for your app.

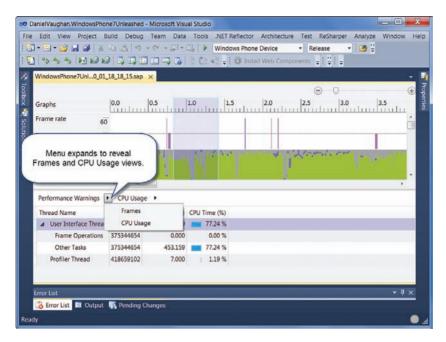


FIGURE 2.7 The Performance Warnings view.

Reading Device Information

The Microsoft.Phone.Info.DeviceStatus class is a static class used to retrieve information about the phone device, such as the device manufacturer, firmware version, and total memory available to your app.

Table 2.3 describes each property of the DeviceStatus class.

TABLE 2.3 DeviceStatus Properties

Name	Description
ApplicationCurrentMemoryUsage	The memory usage of the current application in bytes.
ApplicationMemoryUsageLimit	The maximum additional amount of memory, in bytes, that your application process can allocate.
ApplicationPeakMemoryUsage	The peak memory usage of the current application in bytes.
DeviceFirmwareVersion	The firmware version running on the device.
DeviceHardwareVersion	The hardware version running on the device.
DeviceManufacturer	The device manufacturer name.
DeviceName	The device name.
DeviceTotalMemory	The physical RAM size of the device in bytes.
IsKeyboardDeployed	If true the user has deployed the physical hardware keyboard of the device.

Name	Description
IsKeyboardPresent	If true the device contains a physical hardware keyboard.
PowerSource	Indicates if the device is currently running on battery power or is plugged in to an external power supply.

In the first release of the Windows Phone OS, the DeviceExtendedProperties class was used to retrieve many of the DeviceStatus property values. DeviceExtendedProperties has since been deprecated, and DeviceStatus takes its place for retrieving most device information.

The downloadable sample code contains a DeviceStatusView.xaml page, which displays each of the DeviceStatus properties. The memory related values have been converted from byte values to megabytes to make them more easily comprehendible (see Figure 2.8).



FIGURE 2.8 DeviceStatusView page.

Calculating Available Memory

Windows Phone 8 device manufacturers are obligated to produce phones that have at least 512MB of RAM.

NOTE

Although many phones have more than 512MB of RAM, be mindful of the minimum specification and aim to support the lowest common denominator. Do not assume your app will

be running on a device with more than 512MB of RAM. Regardless of how much memory the device has, your app's memory is capped at either 150MB for lower memory phones or 300MB for higher memory phones. The ID_FUNCCAP_EXTEND_MEM capability can be added to your WMAppManifest.xml file to grant more memory to your app. See http://bit.ly/13o7Pmu for details.

To determine how much memory your app has to work with, use the DeviceStatus.ApplicationMemoryUsageLimit.

For example, if a particular task is estimated at costing an additional 10MB of memory, determining whether the task will exceed the memory usage limit can be calculated as follows:

NOTE

If your app attempts to allocate more memory than is available on the device, that is, it exceeds the value of <code>DeviceStatus.ApplicationMemoryUsageLimit</code>, the application terminates with an <code>OutOfMemoryException</code>.

In addition to foreground app memory constraints, background tasks are limited to 6MB of memory. Background tasks and their memory usage requirements are discussed in Chapter 32, "Conducting Background Activities with Scheduled Actions."

DeviceStatus Events

While DeviceStatus allows you to retrieve device information, it also includes the following two events:

- ► KeyboardDeployedChanged
- ▶ PowerSourceChanged

If the phone device has a hardware keyboard, such as a sliding keyboard, the KeyboardDeployedChanged event allows you to detect when the keyboard is extended.

You can subscribe to the KeyboardDeployedChanged event as shown:

DeviceStatus.KeyboardDeployedChanged += HandleKeyboardDeployedChanged;

The event handler can be used to determine whether the keyboard is deployed using the DeviceStatus class, as shown:

```
void HandleKeyboardDeployedChanged(object sender, EventArgs e)
{
   bool keyboardDeployed = DeviceStatus.IsKeyboardDeployed;
...
}
```

PowerSourceChanged Event

When the phone device is connected to a user's computer, it may be a good time to perform some processor-intensive task that could potentially consume a lot of power, which would otherwise flatten the user's battery. The <code>PowerSourceChangedEvent</code> allows you to detect when the user attaches or detaches an external power supply.

The PowerSourceChanged event can be subscribed to as shown:

```
DeviceStatus.PowerSourceChanged += HandlePowerSourceChanged;
```

The event handler can be used to retrieve the new PowerSource value from the DeviceStatus class, as shown:

```
void HandlePowerSourceChanged(object sender, EventArgs e)
{
    PowerSource powerSource = DeviceStatus.PowerSource;
...
}
```

NOTE

The DeviceState.PowerSourceChanged event is not raised on the app's UI thread. All updates to visual elements must, therefore, be invoked on the UI thread, either directly by using the app's global Dispatcher or indirectly via a custom property change notification system, discussed later in this chapter.

NOTE

Avoid using DeviceState.PowerSource for determining whether to use the phone's network connection to transfer a substantial amount of data. See Chapter 27, "Communicating with Network Services," to learn how to monitor network connectivity and how to determine the type of network connection being used.

Applying the Model-View-ViewModel Pattern to a Windows Phone App

A dominant pattern that has emerged in XAML UI based technologies, in particular WPF and Silverlight, is the Model-View-ViewModel (MVVM) pattern. MVVM is an architectural pattern largely based on the Model-View-Controller (MVC) pattern, which, like the MVC pattern, serves to isolate the domain logic from the user interface logic. In addition, MVVM leverages the strong data binding capabilities of XAML based technologies, which allows loose coupling between the view and the viewmodel so that the viewmodel does not need to directly manipulate the view. This eliminates the need for almost all code-beside, which has a number of benefits, including freeing interactive designers from writing view specific code.

The following are the principal elements of the MVVM pattern:

- ▶ Model—The model is responsible for managing and delivering data.
- ► View—The view is responsible for displaying data. The view is ordinarily a UI element, and, in the case of XAML-based Windows Phone apps, it is a UserControl such as a PhoneApplicationPage.
- ▶ ViewModel—A bridge or intermediary between the model and the view, which commonly retrieves model objects and exposes them to the view. Often the viewmodel is designed to respond to commands that are bound to UI elements in the view. The viewmodel can be thought of as the model of the view.

With the release of the Windows Phone 7.1 SDK came Silverlight 4 and support for ICommands. The use of commands is discussed in the section "Using Commands" later in the chapter.

There are numerous benefits to using MVVM in your apps. MVVM can improve an app's testability because it is easier to test code from a unit test that does not rely on surfacing UI objects. Testing apps is discussed further in Chapter 24, "Unit Testing Apps."

Placing application interaction logic in a viewmodel also makes it easier to redesign your app while reducing the need to refactor interaction logic. Occasionally you may like to reuse some of your UI logic in different apps or, to a lesser extent, you might want to target different UI technologies, such as WPF, Silverlight, or Windows Store XAML applications. Decoupling interaction logic from any particular UI technology makes it easier to target multiple platforms.

Implementing the MVVM Pattern

There are two general approaches to MVVM viewmodel and view creation: view-first and viewmodel-first. The first approach sees the creation of the view before the viewmodel. Conversely, in the viewmodel-first approach, it is the viewmodel that creates the view. Both approaches have their pros and cons. Viewmodel-first potentially offers complete independence from the UI, allowing an app to be executed entirely without a UI; yet it

suffers from various implementation challenges. View-first is far simpler to implement when page navigation is used, as is the case in a Silverlight for Windows Phone app.

This book uses the view-first approach exclusively.

MVVM in a XAML app relies on the assignment of a viewmodel to the view's <code>DataContext</code> property. There are a number of commonly used techniques for marrying a viewmodel to its view. Some offer a high degree of flexibility at the cost of greater complexity and decreased visibility. The technique employed throughout this book, and the one I find to be adequate in most cases, has the viewmodel instantiated in the view's constructor. In the following example a viewmodel is assigned to the view's <code>DataContext</code> property:

```
public partial class FooView : PhoneApplicationPage
{
    public FooView()
    {
        InitializeComponent();

        DataContext = new FooViewModel();
    }
...
}
```

With the DataContext set to the viewmodel, properties of the viewmodel can be used in data binding expression in the view's XAML.

ViewModelBase Class

Windows Phone apps often consist of many pages and, in turn, many viewmodels. It is useful to employ a viewmodel base class to share common infrastructure across all viewmodels in your app.

In the samples throughout this book, most viewmodels subclass a custom ViewModelBase class that provides, among other things, navigation support, error validation, state preservation, and property change notification (see Figure 2.9). Each of these capabilities is discussed alongside related topics in subsequent chapters.

ViewModelBase inherits from a custom NotifyPropertyChangeBase class, which provides for property change notification, discussed in the next section.

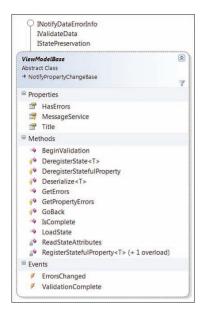


FIGURE 2.9 ViewModelBase class diagram.

Property Change Notification

A key aspect of MVVM related to data binding is property change notification. Property change notification allows a source object (for example, a viewmodel) to signal to a target FrameworkElement that a value needs updating in the UI.

There are two ways to implement change notification in a source class: either using dependency properties or by implementing the INOTIFYPROPERTYCHANGED interface, which is often referred to as just INPC.

NOTE

The use of dependency properties is not recommended for viewmodels because it requires that the viewmodel class inherit from <code>DependencyObject</code> and that all property updates occur on the UI thread. This can lead to a lot of thread-related plumbing code in the viewmodel and makes your code less portable because of dependence on the dependency property system.

Implementing INotifyPropertyChanged: The Traditional Approach

The INotifyPropertyChanged interface has a single event called PropertyChanged. The implementation of INotifyPropertyChanged ordinarily includes the following construct:

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(

```
[CallerMemberName] string propertyName = null)
{
    PropertyChangedEventHandler tempEvent = PropertyChanged;
    if (tempEvent != null)
    {
        tempEvent(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

The CallerMemberName attribute is used by a new compiler feature that automatically passes the name of the calling member to the target method. No longer is it necessary in most cases to pass the name of a property as a loosely typed string.

NOTE

To determine whether the PropertyChanged event field has any subscribers, it is copied to a temporary local variable, which allows you to then test whether it is null in a thread-safe manner. Without first obtaining a copy, another thread could potentially unsubscribe from the event after the null check but before the event is raised, which would inadvertently lead to a NullReferenceException being thrown.

An alternative that avoids the null check is to assign the event to an empty handler, as shown:

```
public event PropertyChangingEventHandler PropertyChanging = delegate {};
```

A property is then able to signal to a subscriber of the event that a property value needs updating, like so:

```
string foo;

public string Foo
{
    get
    {
        return foo;
    }
    set
    {
        if (foo != value)
        {
            foo = value;
               OnPropertyChanged();
        }
    }
}
```

When setting a property that is the source property of a data binding, the update must occur on the UI thread or an <code>UnauthorizedAccessException</code> will ensue. Source properties can be set from non-UI threads using the application's <code>Dispatcher</code> as shown in the following excerpt:

There are a number of reasons why peppering your code with <code>BeginInvoke</code> calls is not a good idea. First, it imposes an unnecessary threading model on your viewmodel code. Second, it can lead to code that need not be executed on the UI thread, creeping in to the delegate. And, third, it is pretty ugly and decreases the readability of your code.

The next section looks at extracting INPC into a reusable and UI thread friendly class.

Implementing INotifyPropertyChanged: An Alternative Approach

Although there is nothing manifestly wrong with adding the <code>OnPropertyChanged</code> method to every class that implements <code>INotifyPropertyChanged</code> (apart from violating the DRY principle), it makes sense to extract the change notification code into a reusable class, because this allows you to not only reduce boilerplate code but also to add other features to the event-raising code, such as improving support for multithreaded apps and implementing <code>INotifyPropertyChanging</code> (as well as <code>INotifyPropertyChanged</code>).

The WPUnleashed project in the downloadable sample code includes such a class, named PropertyChangeNotifier. The ViewModelBase class delegates change notification to a PropertyChangeNotifier instance.

Throughout this book you frequently see viewmodel properties (with backing fields) resembling the following:

```
string foo;

public string Foo
{
    get
    {
        return foo;
    }
    set
    {
        Assign(ref foo, value);
    }
}
```

Here, the name of the property, the current value, and the new value are passed to the base class's Assign method. The following excerpt shows the signature of the Assign method:

```
public AssignmentResult Assign<TField>(
          ref TField field,
          TField newValue,
          [CallerMemberName] string propertyName = "")
{
....
}
```

The Assign method updates the field value, while also offering the following advantages:

- ► The application's Dispatcher automatically raises the PropertyChanged event on the UI thread if called from a non-UI thread. This eliminates the need to add Dispatcher.BeginInvoke calls to a viewmodel to avoid cross-thread errors.
- ► The Assign method also raises a PropertyChanging event. PropertyChangeNotifier implements INotifyPropertyChanging interface as well as INotifyPropertyChanged and allows a subscriber to cancel an update if desired.
- ▶ PropertyChangeNotifier assists the viewmodel in remaining UI technology agnostic. That is, retrieving an application's Dispatcher in a Windows Phone app is done differently in a WPF application.
- ▶ PropertyChangeNotifier uses a weak reference to its owner, thereby preventing memory leaks from occurring when targets fail to unsubscribe from events.
- ▶ The single line Assign method reduces the amount of boilerplate code in properties.

The return value of the Assign method is an AssignmentResult enum value, whose values are described in the following list:

- ▶ **Success**—The assignment occurred and the field value now equals the new value.
- ► Cancelled—A subscriber to the PropertyChanging event cancelled the assignment. This relies on a custom extension to the INotifyPropertyChanging event.
- ▶ AlreadyAssigned—No assignment was made because the existing field value was already equal to the new value.
- ▶ OwnerDisposed—The PropertyChangeNotifier uses a weak reference to the object for which it is providing property changing monitoring. This value indicates that no assignment was performed because the owner object has been disposed.

Because property change notification is such a common requirement of model and viewmodel classes, for the sake of convenience a NotifyPropertyChangeBase class is also provided in the downloadable sample code. It leverages an instance of the PropertyChangeNotifier, and can be used as a base class for any class that needs INotifyPropertyChanged to be implemented.

In particular, the ViewModelBase class inherits from this class (see Figure 2.10).

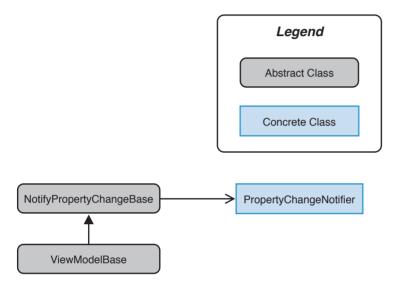


FIGURE 2.10 ViewModelBase inherits from NotifyPropertyChangeBase, which has a PropertyChangeNotifier.

The implementation details of the PropertyChangeNotifier are lengthy and are not included in this section. However, you can find an article already covering the topic at http://danielvaughan.org/post/Property-Change-Notification-using-a-Weak-Referencing-Strategy.aspx.

Before moving on to commanding, be assured that you do not need to use the property notification system presented here in your own projects. If you are happy using the traditional approach to INPC, that is perfectly fine. Be mindful, however, that a lot of the phone SDK APIs have events that do not always return on the UI thread, and you may need to rely more heavily on the <code>Dispatcher</code> to prevent cross-thread errors.

Using Commands

Windows Phone XAML apps support the ICommand interface for buttons and various other controls. Commands are useful because when exposed from a viewmodel they allow your view to bind to them just like other properties; when the user interacts with the visual element, the command is executed. This enables you to move your UI logic from event handlers to higher level classes.

The ICommand interface defines the following three members:

- ► CanExecute (object) —A method called by the commanding infrastructure, which automatically sets the enabled state of the target control
- ▶ Execute (object) —A method that performs the logic of the command
- ► CanExecuteChanged—An event that signals that the commanding infrastructure should reevaluate the executable state of the command by calling its CanExecute method

Within the downloadable sample code there is a default implementation of the ICommand interface called DelegateCommand<T>. This class has features such as object parameter type coercion, which, for example, enables you to use strings to represent enum values in binding expressions, which are automatically converted to the appropriate enum type.

In this book you commonly see commands defined as read-only fields exposed using a property get accessor, as this excerpt from the MediaViewModel in Chapter 7, "Employing Media and Web Elements," shows:

```
readonly DelegateCommand playCommand;
public ICommand PlayCommand
{
    get
    {
        return playCommand;
    }
}
```

Most often, you see commands instantiated in the viewmodels constructor.

The DelegateCommand constructor accepts an Action argument, which is invoked when the command is executed. In the following excerpt you see the instantiation of a command called playCommand that when executed sets a number of viewmodel properties:

```
public MediaViewModel()
{
    playCommand = new DelegateCommand(
```

```
obj =>
{
      PlayerState = PlayerState.Playing;
      CanPlay = false;
      CanPause = true;
});
...
}
```

DelegateCommand along with its generic counterpart DelegateCommand<T> also allow you to specify an Action that is used to evaluate whether the command is able to be executed.

Ordinarily the built-in commanding infrastructure is supported only on buttons (ButtonBase) and a couple of specialized controls. Some extra capabilities are provided in the ICommand implementation that allow you to wire the command to any FrameworkElement, such as in the following example, which shows an Image element that when tapped causes an ICommand to be executed:

NOTE

The event used to trigger command execution can be specified by using the Commanding.Event attached property. In subsequent chapters you see several examples of using these custom commanding attached properties.

Argument Validation

The book sample code commonly uses a custom ArgumentValidator class to ensure that method arguments are not null or fall within a valid range. This allows a method to fail fast, rather than continuing and raising a more difficult to diagnose error.

You frequently see statements like the following at the beginning of a method:

```
string PerformSomeAction(string value)
{
    stringField = ArgumentValidator.AssertNotNull(value, "value");
...
}
```

Here, if value is null, then an ArgumentNullException is thrown. If not null, then the stringField field is set to the value in a fluent manner.

NOTE

Microsoft has a far more feature rich argument validation tool called Code Contracts, which integrates into Visual Studio and can provide static checking as well as runtime checking, along with documentation generation. See http://bit.ly/10zWtK.

All the ArgumentValidator methods are fluent; they return the value passed to them so that they can be assigned to local variables or fields in a single statement.

The Argument Validator. Assert Not Null method is as follows:

```
public static T AssertNotNull<T>(T value, string parameterName) where T : class
{
    if (value == null)
    {
        throw new ArgumentNullException(parameterName);
    }
    return value;
}
```

ArgumentValidator contains various other assertion methods for strings and numeric values. Some are briefly discussed.

ArgumentValidator allows you to assert that an argument falls within a particular range. The following AssertLessThan method ensures that the value is less than a certain value:

This then allows you to validate that a numeric value is less than, for example, 1:

```
ArgumentValidator.AssertLessThan(1, value, "value");
```

Other methods, such as AssertNotNullAndOfType, allow you to raise an exception if an argument is null or not of the expected type, and AssertNotNullOrWhiteSpace accepts a string and raises an ArgumentException if string.IsNullOrWhiteSpace(value) returns true.

A Platform-Agnostic Dialog Service

Over the past few years, I have found myself doing a lot of cross-platform development, in particular Silverlight for the browser, WPF, and now Windows Phone development. Being able to abstract common tasks away from technology specific types, such as displaying simple dialogs, has made reusing code far easier. In addition, mocking things, which would otherwise cause a unit test to fail on a build server, such as displaying a message box, has proven invaluable.

In several places throughout the book you see the use of an IMessageService, which is used to display message dialogs to the user. The ViewModelBase class exposes the IMessageService as a MessageService property, and you see calls like the following:

```
MessageService.ShowMessage("Hi from Windows Phone!");
```

If you are itching to sink your teeth into more phone-specific content, feel free to skip this section and return to it later.

The IMessageService interface describes a class that is able to display messages to the user, and to ask the user questions (see Figure 2.11).

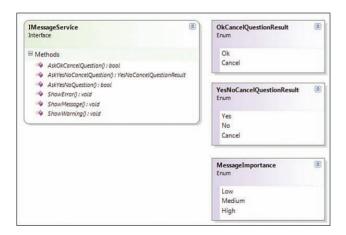


FIGURE 2.11 IMessageService class diagram.

There are various parameters for specifying captions and so forth, along with the capability to provide a message importance threshold value, so that the user can nominate to have messages filtered based on importance.

Differences exist between the built-in dialog related enums in the MessageBox APIs of Windows Phone and WPF. Hence, these types have been replaced with the technology-agnostic enum types shown in Figure 2.11.

The Windows Phone implementation of the IMessageService is done by extending a single class, the MessageServiceBase class, and by overriding two abstract methods: one called ShowCustomDialog, the other AskQuestion (see Figure 2.12).

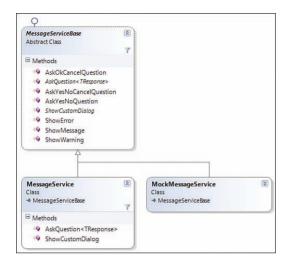


FIGURE 2.12 MessageService extends MessageServiceBase.

The ShowCustomDialog method uses the Dispatcher to display the dialog on the UI thread (see Listing 2.1). Extension methods are used to convert the native Silverlight MessageBoxButton enum values and MessageBoxResult enum values to the technologyagnostic enum values.

LISTING 2.1 MessageService Class (excerpt)

```
public partial class MessageService : MessageServiceBase
    public override MessageResult ShowCustomDialog(
        string message,
        string caption,
       MessageButton messageButton,
       MessageImage messageImage,
       MessageImportance? importanceThreshold,
       string details)
    {
        /* If the importance threshold has been specified
        * and it's less than the minimum level required (the filter level)
         * then we don't show the message. */
        if (importanceThreshold.HasValue
                 && importanceThreshold.Value < MinumumImportance)
            return MessageResult.Ok;
        if (Deployment.Current.Dispatcher.CheckAccess())
             /* We are on the UI thread,
```

```
and hence no need to invoke the call.*/
        var messageBoxResult = MessageBox.Show(message, caption,
            messageButton.TranslateToMessageBoxButton());
        return messageBoxResult.TranslateToMessageBoxResult();
   MessageResult result = MessageResult.Ok;
    var context = new DispatcherSynchronizationContext(
        Deployment.Current.Dispatcher);
    context.Send(
        delegate
          var messageBoxResult = MessageBox.Show(
                                   message, caption,
                             messageButton.TranslateToMessageBoxButton());
         result = messageBoxResult.TranslateToMessageBoxResult();
        }, null);
   return result;
}
/* Content omitted. */
```

The downloadable sample code also contains a MockMessageService class that inherits from MessageService and is designed to be used for unit testing purposes. It allows you to verify that code correctly displayed a message or asked a question. The absence of a mocking framework for Windows Phone makes it especially useful.

The MessageService can be used to display a message, or ask the user a question, from any viewmodel (see Figure 2.13).

You see how to use the <a>IMessageService in greater detail in the next chapter.



FIGURE 2.13 Using the MessageService to display a message.

By using an interface based approach, it affords the opportunity to substitute the IMessageService implementation for a mock implementation, or to even change the behavior of the IMessageService entirely.

These classes are, of course, included in the downloadable sample code. Yet, they also reside in the CalciumSdk repository at http://calciumsdk.com, where you can always find the most up-to-date code, freely available for use in your projects.

Consuming Local Web Applications

Changes to the Windows Phone Emulator have made developing apps that communicate with web applications on the local machine more difficult to set up. Prior to Windows Phone 8, apps could rely on the emulator resolving localhost to the underlying host operating system IP address. This allowed you to rapidly set up a WCF project that could be readily consumed by your Windows Phone app.

In Windows Phone 8, the emulator is unable to resolve WCF services on the host machine without adding a URL reservation and adding a port exception to the Windows firewall.

Perform the following steps to allow a Windows Phone app to consume a service on the host machine:

- **1.** Assuming that you are working with IIS Express for local development, use the Windows File Explorer to navigate to the IIS Express config files in C:\Users\[Username]\My Documents\IISExpress\config
- **2.** Open the applicationhost.config file. Update the physicalPath attribute to the path of the project on your computer. Add an http binding for your machine's name, as shown:

- **3.** To locate your machine name, from a command prompt type "HOSTNAME". Although not necessary, you can update your hosts file and use a different name if you want, which will allow your app to be debugged on other machines more easily.
- **4.** Make a URL reservation by typing the following from an administrative command prompt:

netsh http add urlacl url=http://YourMachineName:27571/ user=everyone

This informs HTTP.SYS that it is okay to allow access to the URL.

To remove the reservation at a later stage, use the following:

netsh http delete urlacl url=http://YourMachineName:27571/

5. Allow IIS Express through the Windows Firewall. You can do this from the command line by typing:

netsh firewall add portopening TCP 27571 IISExpressWeb enable ALL

These steps must be completed for the main WPUnleashed. Web project in the downloadable sample code, and also for the Background Agents and Wallet samples seen later in the book.

NOTE

If you do not perform the steps to enable communication with WCF service on the local machine, several of the sample apps presented in this book that make use of local WCF services will not work.

To save time, Table 2.4 lists the path and port information for the projects in the downloadable sample code.

TABLE 2.4 Path and Ports for Sample Projects

Path	Port
C:\[Path to Samples]\WP8Unleashed\Source\MainExamples\Web\	27571
<pre>C:\[Path to Samples]\WP8Unleashed\Source\Wallet\WalletWcfServices\</pre>	14122
C:\[Path to Samples]\WP8Unleashed\Source\BackgroundAgents\Web\	60182

TIP

To locate the port that a Web project is using, open its properties page and select the Web tab.

Summary

This chapter provided an overview of some common pieces of phone infrastructure and described various techniques that are useful when building Windows Phone apps.

The chapter began with a discussion of the deployment and composition of XAP files. The security capability model of the phone was then discussed, and you learned how to use the Marketplace Test Kit to determine the capability requirements of your app.

The chapter examined the threading model of Windows Phone XAML apps, and you saw how the Windows Phone frame rate counter works.

You then learned about the Windows Phone Application Analysis tool and saw how to profile your app's performance and memory usage.

The chapter then turned to the custom code and commonly used techniques that you see used in subsequent chapters.

The overview of the custom infrastructure began with an exposé of the Model-View-ViewModel pattern, and you saw how it is applied in the downloadable sample code. How property change notification is implemented was discussed, and you saw techniques for improving the traditional implementation of INotifyPropertyChanged so that it works effortlessly with multithreaded apps.

The chapter then looked at the commanding infrastructure used throughout the book and gave a brief overview of the argument validation system commonly used in the sample code to validate method arguments.

The chapter explored a custom dialog service that enables you to ask the user a question from your viewmodel, while remaining compatible with unit testing.

Finally, the chapter showed how to consume WCF services that reside on the same machine as the emulator, and outlined important steps to enable several of the apps in the downloadable sample code.

Index

A

```
accelerometers, 525
   Accelerometer class, 528-529
   calibrating, 533-536
   shake detection, 537-539
   simulating acceleration via emulator,
    529-530
   smoothing readings, 530-533
accuracy, location aware apps, 560
   DesiredAccuracy property, 565
   DesiredAccuracyInMeters property, 565
   monitoring position changes, 566
   MovementThreshold property, 565-566
   PositionChanged events, 566-584
   StatusChanged events, 567-568
ad hoc testing, 737
Add New Project dialog (Visual Studio Express
 2012), 2
Additional Tools window (Windows Phone
 Emulator), 4
AddressChooserTask, choosers/launchers and,
 447-449
A-GPS (Assisted Global Positioning System),
 location aware apps and, 560-561
alarms (scheduled notifications)
   registering, 1001
   sample code, 1002-1004
animation/graphics
   live tiles, 391
   page orientation
      animating entire pages when orientation
       changes, 108-110
      animating page elements when orienta-
       tion changes, 106-107
   page transitions, 110-114
```

XAML	item visibility, 250
composition thread, 18-20	menu items, 237-238
UI thread, 18-20	minimizing Icon Button Tray, 243
App Instant Answer, 996	opacity, 244
App List (Windows Phone Emulator), 6	sample code, 238-242
App.RootFrame_Navigated method (Fast App Resume), 60-61	switching to full-screen, 244-246
App.RootFrame_Navigating method (Fast App Resume), 59-61	multiple applications bars, hosting via Pivot control, 333-343
Application Analysis tool, 21, 26	Application class, 7-8
CPU Usage Graph, 25	Application Deployment tool, 15
execution profiling, 21-23	application state, 49
. 5	persistent state, 56, 97
External Events view, 25	transient state, 97
Frame Rate Graph, 25	requirements, 54-55
GC Events, 26	restoring, 55
Image Loads, 26	saving, 53-54
launching, 21-23	ApplicationCurrentMemoryUsage property
Memory Usage MB Graph, 26	(Device Status class), 28
performance metrics, viewing, 24	ApplicationMemoryUsageLimit property
Performance Warnings view, 26-27	(DeviceStatus class), 27-29
Storyboards, 26	ApplicationPeakMemoryUsage property (DeviceStatus class), 28
application bar, 231, 252	appointments, 464
built-in application bar, 231-233	retrieving, 478-483
customizing colors, 234	saving, 483-484
expanding, 233	apps
hiding, 233	animated page transitions, 110-114
icon buttons, 234-236	Application class, 7-8
menu items, 234-237	auto-launching via
minimizing Icon Button Tray, 234	file associations, 968-975
system tray dimensions, 234	
custom application bar, 237	protocol associations, 977-981
AppBar control, 248-252	Windows.System.Launcher class, 981
customizing colors, 246-247	background file transfers, 1043
customizing system tray, 247-248	app termination/resubscription to transfer events, 1048
disabling application bar menu, 242-243	sample code, 1048-1049
extending icon base classes, 250-252	transfer requests, 1043-1047
extending menu item base classes, 250-252	background processing, 997
icon buttons, 237-238	background tasks, 998

battery life and, 997 multitasking and, 997 multitasking and, 997 Mutex and shared resources, 1038-1041 Mutex and shared resources, 1038-1041 helium voice app, creating, 692-698 scheduled notifications, 998-1010 integration testing, 739 scheduled tasks, 1010-1038 launching via voice commands, 723-732 ScheduledTaskAgent, 1011-1012 lens apps (cameras), 686-687 app registration, 687-689 Bookshop sample app, 83 design-time data, 92-93 icon creation, 689 displaying product lists, 86-92 life cycle of, 49 image caching, 93-95 deactivating events, 52-53 ProductsView class, 85-86 launching events, 50-51 ProductsViewModel class, 84-85 subscribing to events, 51 WCF services, 95-97 tombstoning, 52-53 capabilities location aware apps. 559 defining, 16 A-GPS and, 560-561 determining via Marketplace Test Kit, 17 background location tracking, 569-570 ID_CAP_NETWORK capability, 16 cell tower triangulation and, 561-562 ID HW FFCCAMERA capability, 16 DesiredAccuracy property, 565 customizing DesiredAccuracyInMeters property, 565 determining location, 559-562 MainPage, 6, 9-11 XAML and Windows Phone Emulator, 5-7 geographic location API, 563 deploying via Application Deployment tool, Geolocator class, 563, 570-575 15 hardware laver, 563 exiting apps programmatically, 53 location simulator, 570-575 FAS. 56 location viewer sample code, 575-580 Fast App Resume, 56-57, 63 managed layer, 563 App.RootFrame_Navigated method, 60-61 monitoring position changes, 566 App.RootFrame_Navigating method, MovementThreshold property, 565-566 native code layer, 563 FastAppResumeViewModel class, 61-62 PositionChanged events, 566-584 launching, 57-59 retrieving current location, 563-565 optimizing, 57 Rx and, 580-584 recommended resume behaviors, 58 StatusChanged events, 567-568 file associations, 967-968 Wi-Fi triangulation and, 562 auto-launching apps via file associations, architecture of, 561-562 968-975 Maps app effects on user experience, 976 downloading maps, 612 foreground apps showing directions via, 611 background file transfers, 1043

showing locations via, 610-611	search integration, 983
Marketplace	App Instant Answer, 996
allowing app user reviews, 426	application manifests, 985-987
launching apps from, 425-426	configuring, 985-995
navigating to apps in, 422-425	creating landing pages to display quick card information, 994-995
memory usage, determining, 27-29	customizing captions, 987-989
page navigation, 67-69	defining quick card target page, 989-994
canceling, 77	speech-driven apps, 699
cross-page communication, 77-78	•
handling, 76	launching apps via voice commands, 723-732
hardware Back button, 79-80	speech recognition, 700-723
navigation history stacks, 72-73	TTS, 732-735
NavigationService property, 75-76	Voice Paint app creation, 703-721
page redirection, 78-79	splash screens, 80-83
passing page arguments via query	state preservation, 47-48
strings, 71-73	automatic state preservation system
unmapped URI, 69-71	build, 890-903
URI mapping, 73-74	IsolatedStorageSettings class, 889-890
page orientation	local folder, 881-882
animating entire pages when orientation changes, 108-110	managed storage, 882
animating page elements when orienta-	.NET Isolated Storage API, 883-886
tion changes, 106-107	persistent state, 49, 56, 97
PhoneApplicationPage class, 99-104	transient state, 49, 53-55, 97
setting at runtime, 105-106	WinPRT Windows Storage API, 883, 887-889
performance, Windows Phone Marketplace requirements, 21	terminating apps, 53
PhotoCamera app, building, 658-668	testing via UTF, 739-740
thumbnails, viewing, 672-678	AssemblyCleanup attribute, 746
video effects, 668-671	AssemblyInitialize attribute, 745-746
protocol associations, 967-968	assertions, 751-752
auto-launching apps via protocol associa-	Asynchronous attribute, 749
tions, 977-981	asynchronous testing, 762-763
effects on user experience, 976	Bug attribute, 749-750
retitling, 6	ClassCleanup attribute, 746
running under Lock screen, 63-64	ClassInitialize attribute, 746
LockScreenService class, 64-66	creating a test project, 740-741, 747
LockScreenViewModel class, 66-67	creating test classes, 741-743
Rx and, 580-584	Description attribute, 748

ExpectedException attribute, 749 argument validation, 39-40 hiding Expressions Editor, 752-753 assertions (UTF), 751 Ignore attribute, 747-748 collection conditions, verifying, 752 metadata, 745-751 CollectionAssert, 752 Owner attribute, 748-749 string conditions, verifying, 752 Priority attribute, 750 StringAssert, 752 Tag attribute, 749 associations tag expressions, 743-744 file associations, 967-968 testable chat client app example, auto-launching apps via file associations, 754-764 968-975 TestClass attribute, 744 effects on user experience, 976 testing multiple assemblies, 753 protocol associations, 967-968 testing nonpublic members, 753 auto-launching apps via protocol associations. 977-981 TestInitialize attribute, 747 effects on user experience, 976 TestMethod attribute, 745 asynchronous testing (UTF), 762-763 TestProperty attribute, 747 asynchronous validation, 824 Timeout attribute, 748 decoupling validation, 830 WorkItemTest class, 751 example of, 836-840 unit testing, 737-738 group validation, 844-847 WCF services, consuming, 44-45 INotifyDataErrorInfo interface, 824, 842-844 web content, hosting within apps, 70-71 provisioning for, 827-830 Windows Phone Application Analysis tool, reusable NotifyDataErrorInfo interface, 21, 26 825-827 CPU Usage Graph, 25 validating all properties, 833-836 execution profiling, 21-23 validating changing properties, 830-833 External Events view, 25 ValidationSummary control, adding Frame Rate Graph, 25 INotifiyDataErrorInfo support for, 842-844 GC Events, 26 audio Image Loads, 26 background audio playback, 1059 launching, 21-23 audio streaming agents, 1076-1079 Memory Usage MB Graph, 26 AudioPlayerAgent class, 1060, Performance Warnings view, 26-27 1064-1075 Storyboards, 26 AudioStreamingAgent class, 1060 viewing performance metrics, 24 AudioTrack class, 1061 Windows Phone Marketplace BackgroundAudioPlayer class, 1060-1061 capability discovery phase, 16-17 controlling playback from foreground app, submitting apps to, 15-16 1070-1075 WMAppManifest.xml files, defining app capablitites, 16

creating custom audio player agent,	receiving file launch requests, 970-973
1061-1064	registering file associations, 968-970
overview of, 1060	protocol associations, 977
playing assembly resources via MediaStreamSource, 1078-1079	launching protocol URI, 979-980
media files, playing, 428-435	receiving protocol launch requests, 977-979
microphone, 691	registering protocol associations, 977
helium voice app, creating, 692-698	reserved protocol names, 981
recording audio, 691-692	Windows.System.Launcher class, 981
music	automated testing, 738-739
deleting from media library, 205-207	choosers, testing via, 769-771
saving to media library, 205	integration testing, 739
playing via MediaElement, 193-194	loC, 765-767
controlling audio output, 194	launchers, testing via, 769-771
sample code, 194-200	trial conditions, testing, 767-769
streaming content, 194	unit testing, 737-739
XNA SoundEffect class, 201-202	UTF, 739-740
ringtones, creating, 461-463	AssemblyCleanup attribute, 746
speech-driven apps, 699	AssemblyInitialize attribute, 745-746
launching apps via voice commands,	assertions, 751-752
723-732	Asynchronous attribute, 749
speech recognition, 700-723	asynchronous testing, 762-763
TTS, 732-735	Bug attribute, 749-750
Voice Paint app creation, 703-721	ClassCleanup attribute, 746
AudioPlayerAgent class, 1060, 1064-1075	ClassInitialize attribute, 746 ClassInitialize attribute, 746 creating a test project, 740-741, 747 creating test classes, 741-743 Description attribute, 748 ExpectedException attribute, 749
AudioStreamingAgent class, 1060	
AudioTrack class, 1061	
authentication, cloud services and, 511	
AutoCompleteBox component (Toolkit), 259-260	
data binding, 270-271	hiding Expressions Editor, 752-753
defining custom filters, 264-266	Ignore attribute, 747-748
ItemFilter property, 264-266	metadata, 745-751
MVVM and, 260-264	Owner attribute, 748-749 Priority attribute, 750
suggested lists	
dynamically populating, 266-267	Tag attribute, 749
styling, 268-270	tag expressions, 743-744
auto-launching apps via	testable chat client app example, 754-764
file associations, 968	
launching files, 973-975	

TestClass attribute, 744 background processing, 997 testing multiple assemblies, 753 background tasks, 998 testing nonpublic members, 753 battery life and, 997 TestInitialize attribute, 747 multitasking and, 997 TestMethod attribute, 745 Mutex, shared resources and, 1038-1041 TestProperty attribute, 747 scheduled notifications, 998-1001 Timeout attribute, 748 alarm registration, 1001 WorkItemTest class, 751 alarm sample, 1002-1004 reminder registration, 1005-1006 reminder sample, 1006-1010 scheduled tasks, 1010-1011, 1014-1017 API limitations, 1036-1038 background agents, 1011-1014 Back button, page navigation via, 79-80 debugging, 1025 background agents, 1011-1012, 1059-1060 periodic tasks, 1013-1014 background audio playback, 1059 registering, 1018-1019 assembly resources, playing via MediaStreamSource, 1078-1079 resource intensive tasks, 1014 audio streaming agents, 1076-1079 scheduled task agents, 1011-1012, 1017-1018, 1035-1036 AudioPlayerAgent class, 1060, 1064-1075 to-do list example, 1019-1035 AudioStreamingAgent class, 1060 banking, Wallet hub and, 773 AudioTrack class, 1061 introduction to, 774-775 BackgroundAudioPlayer class, 1060-1061 payment instruments, storing in, 775-777 custom audio player agent, creating, 1061-1064 Payment Instruments sample app, 777-789 foreground app, controlling playback from, 1070-1075 updating payment instruments, 789-792 overview of, 1060 required capabilities for wallet integration, background file transfers, 590 battery life (power management) sample code, 1048-1049 background processing and, 997 backing up local databases, 1052-1055 push notifications and, 493-494 restoring local databases, 1055-1058 BingMapsDirectionTask, choosers/launchers retrieving user Windows Live Anonymous and, 409-412 IDs, 1050-1052 BingMapsTask, choosers/launchers and, 413 URL rerouting with WCF services, 1049-1050 bold text, 159-160 transfer requests, 1043-1047 Bookshop sample app, 83 background images (lock screen), setting, design-time data, 92-93 399-401

B

Background layer (Panorama control), 348-349

image caching, 93-95

product lists, displaying, 86-92

ProductsView class, 85-86	cameras, 655
ProductsViewModel class, 84-85	lenses, 686-687
WCF services, 95-97	app registration, 687-689
Boolean option, providing via ToggleSwitch	icon creation, 689
component (Toolkit), 293-295	PhotoCamera class, 655-656
localizing ToggleSwitch component, 295-297	building PhotoCamera app, 658-678
switch color, changing, 297	selection criteria, 656
built-in fonts (text), 160-162	thumbnails, viewing in PhotoCamera app,
buttons	672-678
Button Click mode, 124	video effects, adding to PhotoCamera
check boxes, 131-138	app, 668-671
Hyperlink button, 124	Silverlight webcam API, 655-656, 678-685
Icon Button Tray (application bar), minimiz-	taking photos, 450-452
ing, 234	canceling page navigation, 77
icon buttons (application bar)	CanExecute (object) method (iCommand interface), 38
custom application bar, 237-238	CanExecuteChanged method (iCommand inter-
customizing, 234-235	face), 38
disabling, 235	capabilities (apps)
retrieving at runtime, 236-237	app submissions to Windows Phone
Radio button, 128-131 Repeat button, 124-126	Marketplace, capability discovery phase, 16-17
sizing, 121-123	defining, 16
tap and click events, 123	ID_CAP_NETWORK capability, 16
Toggle button, 124, 126-128	ID_HW_FFCCAMERA capability, 16
buying items, Wallet hub and, 773	Marketplace Test Kit, determining via, 17
introduction to, 774-775	cartographic modes (Map control), 589-590
payment instruments, storing in, 775-777	cell tower triangulation and location aware apps, 561-562
Payment Instruments sample app, 777-789	certification requirements, launching events, 50
updating payment instruments, 789-792	chat client app UTF testing example, 754-758
required capabilities for wallet integration,	automation peers, manipulating UI elements at runtime, 763-764
	code driven UI testing, 760-763
	views, building, 758-760
0	check boxes, 131-138
C	choosers, 405, 408-409, 484
calle (phone) placing 425.426	AddressChooserTask, 447-449
calls (phone), placing, 435-436 CameraCaptureTask, choosers/launchers and,	API overview, 405-406
450-452	application execution model, 406-408

appointments, 464 MarketplaceHubTask, 425-426 retrieving, 478-483 MarketplaceReviewTask, 426 saving, 483-484 MarketplaceSearchTask, 427-428 automated testing, 769-771 media files, playing, 428-435 BingMapsDirectionTask, 409-412 MediaPlayerLauncher, 428-435 BingMapsTask, 413 phone calls, placing, 435-436 CameraCaptureTask, 450-452 phone numbers ChooserBase.Completed events, 406-408 saving, 439-442 connection setting pages, navigating to, selecting, 437-438 413-414 PhoneCallTask, 435-436 ConnectionSettingsTask, 413-414 PhoneNumberChooserTask, 437-438 contacts, 464 Photo Hub, selecting photos from, 454-459 converting stored contacts to/from PhotoChooserTask, 454-459 vCards, 478 photos, taking, 450-452 deleting changes to stored contacts, ringtones, creating, 461-463 477-478 SaveContactTask, 449-450 retrieving, 464-469 SaveEmailAddressTask, 418-421 saving to contact lists, 449-450 SavePhoneNumberTask, 439-442 selecting addresses, 447-449 SaveRingtoneTask, 461-463 storing in Custom Contacts Store, 469-477 SearchTask, 442-443 driving directions, retrieving, 409-412 ShareLinkTask, 444 email ShareMediaTask, 460-461 composing, 417-418 ShareStatusTask, 445 saving contact addresses, 418-421 SMS, composing, 445 selecting email addresses, 414-417 SmsComposeTask, 445 EmailAddressChooserTask, 414-417 status updates, posting to social networks, 445 EmailComposeTask, 417-418 tombstoning and, 406-408 GamelnviteTask, 453 web pages, navigating, 446-447 games, inviting players to, 453 web searches, 442-443 images, sharing, 460-461 WebBrowserTask, 446-447 links, sending to contacts, 444 click events (buttons), 123 map locations, displaying, 413 clipboard, placing text on, 179-180 Marketplace Close button (Windows Phone Emulator), 5 allowing app user reviews, 426 cloud services, authenticating, 511 launching apps from, 425-426 club memberships, Wallet hub and, 773 navigating to apps in, 422-425 introduction to, 774-775 searching, 427-428

MarketplaceDetailTask, 422-425

membership information	changing properties, 830-833
storing, 792-796	ValidationSummary control, adding
updating, 802-804	INotifiyDataErrorInfo support for, 842-844
required capabilities for wallet integration, 775	Composition (Render) Thread Frame Rate (FPS) field (Frame Rate Counter), 22
coded UI testing, 739	composition thread (XAML animation/graphics) 18-20
CodePlex.com, Toolkit source code, 254	concurrency, LINQ to SQL and, 962-965
color	connection setting pages, navigating to,
application bar color, customizing	413-414
built-in application bar, 234	ConnectionSettingsTask, choosers/launchers
custom application bar, 246-247	and, 413-414
colored fonts (text), 160	connectivity (network), monitoring, 850-855
switch color, changing via ToggleSwitch com-	contacts, 464
ponent (Toolkit), 297	contact lists, saving contacts to, 449-450
ComboBox control, 142	Custom Contacts Store, storing in, 469-477
Commanding.Event property (iCommand interface), 39	retrieving, 464-469
commands, iCommand interface	stored contacts
CanExecute (object) method, 38	converting to/from vCards, 478
CanExecuteChanged method, 38	deleting changes to, 477-478
Commanding.Event property, 39	content controls, Windows Phone FCL, 119-121
DelegateCommand constructor, 38-39	ContextMenu component (Toolkit)
Execute (object) method, 38	displaying, 271-273
playCommand constructor, 38-39	hosting in a ListBoxItem, 275-277
compass sensors, 525, 539-545	MVVM and, 273-275
calibrating, 546-548	control headers, adding to TimePicker component (Toolkit), 281
compass orientation, 545-546	controls
compatibility, Windows Mobile 6.5 apps and	buttons
Windows Phone 8, 1	Button Click mode, 124
composite validation	check boxes, 131-138
decoupling validation, 830	Hyperlink button, 124
example of, 836-840	Radio button, 128-131
group validation, 844-847	Repeat button, 124-126
INotifyDataErrorInfo interface, 824, 842-844	sizing, 121-123
provisioning for, 827-830	tap and click events, 123
reusable NotifyDataErrorInfo interface, 825-827	Toggle button, 124, 126-128
validating	content controls, 119-121
all properties, 833-836	FrameworkElement class, 116

items controls required capabilities for wallet integration, ComboBox control, 142 storing in Wallet hub, 775-789 ListBox control, 140-142 cross-page communication and page navigation, Selector class members, 139 77-78 range controls cross-platform development, IMessageService progress indicators, 145-151 interface, 41-44 ProgressBar control, 144 Custom Contacts Store, storing contacts in, 469-477 RangeBase class properties, 143-144 scrollbars, 152 customizing application bar, 234, 246-247 sliders, 151-152 DatePicker/TimePicker components sensors full-screen picker pages, 282-286 accelerometers, 525, 527-539 value formats in. 281-282 compass sensors, 525, 539-548 maps, 604-610 gyroscope sensors, 525, 548-552 motion sensors, 525, 552-557 picker pages, 282-286 switch color, changing via ToggleSwitch comoverview of, 525-527 ponent (Toolkit), 297 ToolTip control, 138 system tray, custom application bar, touch input 247-248 designing for, 386-388 cycle tiles, 390-394 double tap gestures, 370 hold gestures, 371-372 manipulation events, 357, 363-367 ח mouse events, 357-359, 361-362 tap gestures, 369-370 data binding Toolkit gestures, 291-293, 372-386 AutoCompleteBox component (Toolkit), TouchPoint class, 357, 360-363 270-271 UIElement gesture events, 357-358, Pivot control, populating via data bound col-368-372 lections, 344-347 Windows Phone Emulator, 358 Data Sense, monitoring data usage via, Windows Phone FCL 877-879 control types within, 117 databases (local), data storage and, 905 Silverlight controls not in FCL, 118-119 database schemas unsupported Silverlight controls, 119 updating, 951-957 CPU Usage Graph (Windows Phone Application versioning, 957-958 Analysis tool), 25 viewing, 937-942 credit/debit cards, Wallet hub and, 773 deploying databases, 906-907 introduction to, 774-775 isolated storage, deploying databases to, payment instruments, storing in, 789-792 943-946

LINQ to SQL, 905, 908 required capabilities for wallet integration, code-first data model creation, 909-910 storing in Wallet hub, 775-789 concurrency, 962-965 debugging mapping inheritance hierarchies, 958-962 Lock screen, engaging while debugging via Simulation Dashboard, 63 platform differences, 909 scheduled tasks, 1025 viewing queries via custom log, 948-950 Deep Zoom technology, viewing high-resolution navigation services, abstracting, 946-948 images via MultiScaleImage element, SQL Server Compact, 906 207-217 SqlMetal and database-first data model cre-DelegateCommand constructor (iCommand ation. 942-943 interface), 38-39 Twitter timeline viewer example, 910-912 deleting Association attribute, 917-918 changes to stored contacts, 477-478 Column attribute, 912-914 music to media library, 205-207 connection strings, 920-924 deploying apps via Application Deployment tool, database utilities, 919-920 DeviceExtendedProperties class, See leveraging custom Twitter services, DeviceStatus class 924-929 DeviceFirmwareVersion property (DeviceStatus multiplicity, 917-918 class), 28 one-to-many relationships, 917-918 DeviceHardwareVersion property (DeviceStatus tracking data context changes, 915 class), 28 TwitterDataContext class, 918-919 DeviceManufacturer property (DeviceStatus TwitterUser class, 915-917 class), 28 user credentials, gathering in Sign In DeviceName property (DeviceStatus class), 28 view, 929-932 DeviceStatus class, 27 viewing tweets in Timeline view, 932-937 ApplicationCurrentMemoryUsage property, XAP files, 906-907 DatePicker/TimePicker components (Toolkit), ApplicationMemoryUsageLimit property, 277-281 27-29 control headers, adding to TimePicker com-ApplicationPeakMemoryUsage property, 28 ponent (Toolkit), 281 DeviceFirmwareVersion property, 28 customizing DeviceHardwareVersion property, 28 full-screen picker pages, 282-286 DeviceManufacturer property, 28 value formats, 281-282 DeviceName property, 28 deactivating events (life cycle of apps), 52-53 DeviceTotalMemory property, 28 deals/special offers, Wallet hub and, 796-802 IsKeyboardDeployed property, 28 debit/credit cards, Wallet hub and, 773 IsKeyboardPresent property, 28 introduction to, 774-775 KeyboardDeployedChanged event, 29-30 payment instruments, storing in, 789-792 PowerSource property, 28, 30

PowerSourceChanged event, 29-30	embedding fonts (text), 162-164
properties, viewing, 27	emulators. See Windows Phone Emulator
DeviceTotalMemory property (DeviceStatus class), 28	events (life cycle of apps) deactivating events, 52-53
dictation grammars (speech recognition), 701-703	launching, 50-51
directions	subscribing to, 51
driving directions, retrieving, 409-412	tombstoning, 52-53
Maps app, showing in, 611	Execute (object) method (iCommand interface), 38
disabling	execution model, 48
application bar menu, 242-243	application life cycle, 49
icon buttons (application bar), 235	deactivating events, 52-53
double tap gestures (touch input), 370, 374, 382	launching events, 50-51
drag gestures (touch input), 375-377, 383	subscribing to events, 51
drawing surface, presenting via InkPresenter	tombstoning, 52-53
element, 186-193	application state, 49
driving directions, retrieving, 409-412	restoring transient state, 55
	saving persistent state, 56
	saving transient state, 53-54
E	transient state requirements, 54-55
	apps
eBay OData consumer application, building, 862	exiting programmatically, 53 terminating, 53
EbaySearchView page, 867-871	execution profiling (Windows Phone Application
EbaySearchViewModel class, 864-867	Analysis tool), 21-23
extending OData entity classes, 874	exiting apps programmatically, 53
fetching data when user scrolls to the end	expanding application bar, 233
of a list, 871-873	Expression Blend for Windows Phone, 2
OData wrapper creation, 862-864	Expressions Editor (UTF), hiding, 752-753
ScrollViewerMonitor class, 871-873	External Events view (Windows Phone Application Analysis tool), 25
edge tracing photo extras application, creating, 634-645	
email	
composing, 417-418	F
contact addresses, saving, 418-421	•
email addresses, selecting, 414-417	FAS (Fast App Switching), 56
EmailAddressChooserTask, choosers/launchers and, 414-417	Fast App Resume, 56, 63 App.RootFrame_Navigated method, 60-61
EmailComposeTask, choosers/launchers and, 417-418	App.RootFrame_Navigating method, 59-61

enabling, 56-57	flick gestures (touch input), 377-378, 384-386
FastAppResumeViewModel class, 61-62	flip tiles, 390, 394-395
launching, 57-59	floating menu (Windows Phone Emulator), 4
optimizing, 57	fonts (text)
recommended resume behaviors, 58	assigning, 164
file associations, 967-968	bold text, 159-160
auto-launching apps via file associations, 968	built-in fonts, 160-162
launching files, 973-975	colored fonts, 160
receiving file launch requests, 970-973	embedding fonts, 162-164
registering file associations, 968-970	language support, 160-161
user experience, effects on, 976	OpenType fonts, 159
file transfers (background), 1043	properties of, 158-160
app termination/resubscription to transfer	Segoe font, 160
events, 1048	selecting, 159
sample code, 1048-1049	sizing, 159-160
backing up local databases, 1052-1055	stretching, 159
restoring local databases, 1055-1058	styling, 159
retrieving user Windows Live Anonymous	third-party fonts, 162-164
IDs, 1050-1052	weight adjustments, 159-160
URL rerouting with WCF services, 1049-1050	Windows Font Preview tool, 163
	foreground apps
transfer requests, 1043-1047	background audio playback, controlling, 1070-1075
financial information, storing in Wallet hub, 773 introduction to, 774-775	background file transfers, 1043
	multitasking and, 997
payment instruments, storing in, 775-777	Mutex and shared resources, 1038-1041
Payment Instruments sample app, 777-789	foreign languages/cultures. See international-
updating payment instruments, 789-792	ization
required capabilities for wallet integration,	Frame Rate Counter, 20-21
775	Frame Rate Graph (Windows Phone Application
Fit to Screen button (Windows Phone Emulator),	Analysis tool), 25
5	FrameworkElement class, 116
flat lists, 305-306	full-screen, switching application bar to, 244-246
flexible layouts via WrapPanel component (Toolkit), 297	full-screen picker pages, customizing, 282-286
child element spacing, 298	
ListBox, using with, 300-302	
sample code, 298-299	

UI thread, 18-20

	group validation, 818-820, 844-847
game loops, controlling via GameTimer class,	grouped lists, 307
202-203	Marketplace app list sample, 307-314
update intervals, controlling, 203	user contacts list sample, 316-323
XNA environment initialization, 203-204	gyroscope sensors, 525, 548-552
GameInviteTask, choosers/launchers and, 453	
games, inviting players to, 453	
GC Events (Windows Phone Application Analysis tool), 26	Н
geographic location API, 563	
DesiredAccuracy property, 565	headers
DesiredAccuracyInMeters property, 565	control headers, adding to TimePicker com- ponent (Toolkit), 281
Geolocator class, 563	Pivot headers, 331-332
location simulator sample code, 571-575	heading, adjusting in Map control, 593
testing, 570-571	helium voice app, creating, 692-698
monitoring position changes, 566	hiding
MovementThreshold property, 565-566	application bar, 233
PositionChanged events, 566-584	Expressions Editor (UTF), 752-753
retrieving current location, 563-565	high-resolution images, viewing via
StatusChanged events, 567-568	MultiScaleImage element, 207-208
geographically aware apps. See location aware apps	Deep Zoom technology, 207-217
gesture events (touch input), 357-358,	Deep Zoom Viewer sample code,
368-369	212-217
double tap gestures, 370	image creation, 208-209
GestureBegin events, 380-381	image tiling, 208
GestureCompleted events, 380-381	MultiScaleImage API, 210-212
hold gestures, 371-372 tap gestures, 369-370	hold gestures (touch input), 371-372, 375, 382
	HTTP services, 850
globalization, 613-614	Hyperlink button, 124
graphics/animation	
page orientation	
animating entire pages when orientation changes, 108-110	I
animating page elements when orientation changes, 106-107	iCommand interface, 38
page transitions, 110-114	CanExecute (object) method, 38
XAML	CanExecuteChanged method, 38
composition thread. 18-20	Commanding.Event property, 39

G

DelegateCommand constructor, 38-39 localizability (internationalization), 620-621, 625-629 Execute (object) method, 38 Photo Hub, selecting photos from, 454-459 playCommand constructor, 38-39 picture viewer, 631 Icon Button Tray (application bar), minimizing, 234 edge tracing extras application, 634-645 icon buttons (application bar) photo extras applications, 631-645 built-in application bar photo share applications, 631, 645-654 customizing, 234-235 photo upload share application, 647-654 disabling, 235 pixel density/picture quality, 632 custom application bar sharing, 460-461 icon buttons, 237-238 sizing, 185-186 minimizing Icon Button Tray, 243 stock ticker application example, 520-521 iconic tiles, 390, 395-396 taking photos, 450-452 ID_CAP_NETWORK capability, 16 thumbnail images, viewing in PhotoCamera app, 672-678 ID HW FFCCAMERA capability, 16 IMessageService interface, 41-44 Image Loads (Windows Phone Application Analysis tool), 26 InkPresenter element, drawing surfaces, 186-193 images INPC (INotifyPropertyChanged) class, MVVM background images (lock screen), setting, property change notification 399-401 alternative implementation approach, 35-37 caching, Bookshop sample app, 93-95 traditional implementation approach, 33-35 cameras, 655 in-place selection in Toolkit, 255-258 lenses, 686-689 input validation, 805 PhotoCamera class, 655-678 asynchronous validation, 824 Silverlight webcam API, 655-656. 678-685 decoupling validation, 830 displaying, 184-185 example of, 836-840 drawing surface, presenting via InkPresenter group validation, 844-847 element, 186-193 INotifyDataErrorInfo interface, 824, high-resolution images, viewing via 842-844 MultiScaleImage element, 207-208 provisioning for, 827-830 Deep Zoom image creation, 208-209 reusable NotifyDataErrorInfo interface, Deep Zoom technology, 207-217 825-827 Deep Zoom Viewer sample code, validating all properties, 833-836 212-217 validating changing properties, 830-833 image tiling, 208 ValidationSummary control, adding MultiScaleImage API, 210-212 INotifyDataErrorInfo support for, 842-844 icon buttons (application bar), 235-236 composite validation live tiles, 390 decoupling validation, 830

example of, 836-840 integration testing, 739 group validation, 844-847 IntelliSense text support, 168 INotifyDataErrorInfo interface, 824. Intermediate Surface Counter field (Frame Rate 842-844 Counter), 22 internationalization, 613 provisioning for, 827-830 reusable NotifvDataErrorInfo interface. defining, 613 825-827 globalization, 613-614 validating all properties, 833-836 localizability validating changing properties, 830-833 defining, 614 ValidationSummary control, adding dynamic localizability, 618-620 INotifyDataErrorInfo support for, images, 620-621, 625-629 842-844 resx files, 614-618, 620-630 group validation, 818-820, 844-847 RTL support, 629-630 property setters, 806-809 sample code, 621-630 binding errors, 809-810 text, 625-629 critical exceptions, 809 UI updates when cultures change, validation class, 809 618-620 validation limitations, 823-824 speech-driven apps, 721-723 semantic validation, 806 text support, 160-161 synchronous validation Internet decoupling validation, 830 links, sending to contacts, 444 provisioning for, 827-828 web pages, navigating, 446-447 validating changing properties, 830-833 web searches, 442-443 syntactic validation, 806 invitations (games), 453 validation visual states, defining, 810-815 IoC (Inversion of Control), 765-767 displaying error details, 820-823 IsKeyboardDeployed property (DeviceStatus group validation, 818-820 class), 28 property setter validation limitations, IsKeyboardPresent property (DeviceStatus 823-824 class), 28 validating textboxes as user types, isolated storage 815-818 IsolatedStorageSettings class, 889-890 inputting text via TextBox control, 165 local databases, deploying to, 943-946 input scope, 168-174 SQL CE database files, 940-942 IntelliSense support, 168 storing/browsing web content in, 225-228 SIP, 165-168 Windows Phone SDK Isolated Storage word prediction, 168 Explorer, 938-939 installing WP7 Isolated Storage Explorer, 939-940 Toolkit, 254 item visibility, custom application bar, 250 Windows Phone SDK 8.0, 2

BingMapsDirectionTask, 409-412

items controls	BingMapsTask, 413
ComboBox control, 142	CameraCaptureTask, 450-452
ListBox control, 140-142	connection setting pages, navigating to
Selector class members, 139	413-414
Items layer (Panorama control), 349	ConnectionSettingsTask, 413-414
itineraries, displaying via visual states in Map	contacts, 464
control, 604-610	converting stored contacts to/from vCards, 478
J - K - L	deleting changes to stored contacts 477-478
) - K - L	retrieving, 464-469
JavaScript, web page behaviors, 222-223	saving to contact lists, 449-450
· · · · · ·	selecting addresses, 447-449
KeyboardDeployedChanged event (DeviceStatus class), 29-30	storing in Custom Contacts Store, 469-477
keyboards, SIP layouts, 165-166	driving directions, retrieving, 409-412
	email
languages (internationalization), 613	composing, 417-418
globalization, 613-614	saving contact addresses, 418-421
internationalization, 613	selecting email addresses, 414-417
localizability	EmailAddressChooserTask, 414-417
defining, 614	EmailComposeTask, 417-418
dynamic localizability, 618-620	games, inviting players to, 453
images, 620-621, 625-629	images, sharing, 460-461
resx files, 614-618, 620-630	links, sending to contacts, 444
RTL support, 629-630	map locations, displaying, 413
sample code, 621-630	Marketplace
text, 625-629	allowing app user reviews, 426
UI updates when cultures change, 618-620	launching apps from, 425-426
speech-driven apps, 721-723	navigating to apps in, 422-425
text support, 160-161	searching, 427-428
launchers, 405, 408-409, 484	Marketplace Unit Tank, 425-425
AddressChooserTask, 447-449	MarketplaceHubTask, 425-426
API overview, 405-406	MarketplaceReviewTask, 426
appointments, 464	MarketplaceSearchTask, 427-428
retrieving, 478-483	media files, playing, 428-435
saving, 483-484	MediaPlayerLauncher, 428-435
automated testing, 769-771	phone calls, placing, 435-436

phone numbers subscribing to, 51 saving, 439-442 tombstoning, 52-53 selecting, 437-438 light (low), Map control adjustments, 588-589 PhoneCallTask, 435-436 LineBreak objects, TextBlock element, 155-158 PhoneNumberChooserTask, 437-438 links, sending to contacts, 444 Photo Hub, selecting photos from, 454-459 LINO PhotoChooserTask, 454-459 maps and, 597 photos, taking, 450-452 OData and, 855-856 ringtones, creating, 461-463 LINQ to SQL SaveContactTask, 449-450 concurrency, 962-965 SaveEmailAddressTask, 418-421 inheritance hierarchies, mapping, 958-962 SavePhoneNumberTask, 439-442 local databases, data storage and, 905, 908 SaveRingtoneTask, 461-463 code-first data model creation, 909-910 SearchTask, 442-443 platform differences, 909 ShareLinkTask, 444 queries, viewing via custom log, 948-950 ShareMediaTask, 460-461 list grammars (speech recognition), 701-710, ShareStatusTask, 445 731-732 SMS, composing, 445 ListBox control, 140-142 SmsComposeTask, 445 ListPicker component (Toolkit), 255-258 status updates, posting to social networks, lists 445 flat lists, 305-306 web pages, navigating, 446-447 grouped lists, 307 web searches, 442-443 Marketplace app list sample, 307-314 WebBrowserTask, 446-447 user contacts list sample, 316-323 launching events (life cycle of apps), 50-51 LongListSelector, 303-304 layouts (flexible) via WrapPanel component live tiles, 389 (Toolkit) child element spacing, 298 animation, 391 flexible layouts via WrapPanel component changing templates, 391 (Toolkit), 297 configuring, 390 ListBox, using with, 300-302 cycle tiles, 390-394 sample code, 298-299 flip tiles, 390, 394-395 lenses (cameras), 686-687 iconic tiles, 390, 395-396 app registration, 687-689 images, 390 icons, creating, 689 large tiles, 390 life cycle of apps, 49 lock screen customization, 397-399 events background images, 399-401 deactivating, 52-53 creating lock screen icons, 403-404 launching, 50-51 notification text, 401-403

resolution, 391	user credentials, gathering in Sign In
schedules, updating tile notifications,	view, 929-932
504-505	viewing tweets in Timeline view, 932-937
sizing, 390-391	XAP files, 906-907
sound, 391	local folder, state preservation and, 881-882
Start Experience (Windows Phone Emulator), 389-390	.NET Isolated Storage API, 883-886 WinPRT Windows Storage API, 883, 887-889
updating, 391, 396-397	
local databases, data storage and, 905	localizability
database schemas	defining, 614
updating, 951-957	dynamic localizability, 618-620
versioning, 957-958	images, 620-621, 625-629
viewing, 937-942	resx files, 614-618
deploying databases, 906-907	image localization, 620-621
isolated storage, deploying databases to,	localizability sample code, 621-630
943-946	RTL support, 629-630
LINQ to SQL, 905, 908	sample code, 621-630
code-first data model creation, 909-910	text, 625-629
concurrency, 962-965	UI updates when cultures change, 618-620
mapping inheritance hierarchies, 958-962	location aware apps. See also maps, 559 A-GPS and, 560-561
platform differences, 909	architecture of, 562-563
viewing queries via custom log, 948-950	background location tracking, 569-570
navigation services, abstracting, 946-948	cell tower triangulation and, 561-562
SQL Server Compact, 906	determining location, 559-562
SqlMetal and database-first data model cre-	geographic location API, 563
ation, 942-943	DesiredAccuracy property, 565
Twitter timeline viewer example, 910-912	DesiredAccuracyInMeters property, 565
Association attribute, 917-918	Geolocator class, 563, 570-575
Column attribute, 912-914	monitoring position changes, 566
connection strings, 920-924	MovementThreshold property, 565-566
database utilities, 919-920	PositionChanged events, 566-584
leveraging custom Twitter services, 924-929	retrieving current location, 563-565
multiplicity, 917-918	StatusChanged events, 567-568
one-to-many relationships, 917-918	hardware layer, 563
tracking data context changes, 915	location simulator, 570-575
TwitterDataContext class, 918-919	location viewer sample code, 575
TwitterUser class, 915-917	civic address resolution, 580

displaying location via GeoLocationView М page, 577-579 GeoLocationViewModel class. 575-577 MainPage (XAML apps), 6, 9-11 managed layer, 563 managed storage, state preservation and, 882 native code laver, 563 manipulation events (touch input), 357, Rx and, 580-584 363-367 Wi-Fi triangulation and, 562 maps. See also location aware apps, 559 location tracking via Map control, 594-596 LINQ and, 597 lock screen Map control, 585-587 customizing via live tiles, 397-399 Application ID, 587 AuthenticationToken, 587 background images, 399-401 creating lock screen icons, 403-404 calculating shortest routes between two points, 598-601 notification text, 401-403 cartographic modes, 589-590 running apps under, 63-64 centering to current location, 590-591 LockScreenService class, 64-66 customizing page elements via visual LockScreenViewModel class, 66-67 states, 604-610 Windows Phone Emulator, engaging Lock heading adjustments, 593 screen within, 63 location tracking, 594-596 LockablePivot control (Toolkit), 355 low light conditions, 588-589 LongListSelector, 303-304 overlaying content, 596-597 flat lists, 305-306 panning/zooming, 590 grouped lists, 307 pitch adjustments, 593 Marketplace app list sample, 307-314 route calculator, 601-602 user contacts list sample, 316-323 sample Map View, 587-588 visual structure of, 304 searching for routes via looping lists, 287-291 RouteSearchView, 602-604 LoopingSelector component (Toolkit), 287-291 setting viewable area of map, 592-593 low light conditions, adjusting Map control for, showing directions via Maps app, 611 588-589 showing locations via Maps app, loyalty cards, Wallet hub and, 773 610-611 introduction to, 774-775 Maps app membership information downloading maps, 612 storing, 792-796 showing directions via, 611 updating, 802-804 showing locations via, 413, 610-611 required capabilities for wallet integration, Marketplace 775 apps allowing user reviews, 426 launching, 425-426

745-751

navigating to, 422-425 Metro. See Microsoft Design Language searching, 427-428 microphone, 691 Marketplace Test Kit, determining app capabilihelium voice app, creating, 692-698 ties via, 17 recording audio, 691-692 MarketplaceDetailTask, choosers/launchers Microsoft Design Language, 153 and, 422-425 Microsoft Expression Blend for Windows Phone, MarketplaceHubTask, choosers/launchers and, 425-426 Microsoft Phone Info DeviceStatus class. See MarketplaceReviewTask, choosers/launchers DeviceStatus class and, 426 minimizing MarketplaceSearchTask, choosers/launchers Icon Button Tray (application bar), 243 and, 427-428 Windows Phone Emulator window, 4-5 media files, playing, 428-435 motion sensors, 525, 552-557 media library, saving/deleting music, 205-207 mouse events (touch input), 357-359, 361-362 MediaElement, playing audio/video, 193-194 MPNS (Microsoft Push Notification Service), controlling audio output, 194 487-488, 509-511 sample code, 194-200 MSIL (Microsoft Intermediate Language), XAP streaming content, 194 files, 16-17 XNA SoundEffect class, 201-202 multiline text, 155-158 MediaPlayerLauncher, choosers/launchers and. multiple languages/cultures (internationaliza-428-435 tion), 613 memberships (club), Wallet hub and, 773 globalization, 613-614 introduction to, 774-775 internationalization, 613 membership information localizability storing, 792-796 defining, 614 updating, 802-804 dynamic localizability, 618-620 required capabilities for wallet integration, images, 620-621, 625-629 775 resx files, 614-618, 620-630 memory usage RTL support, 629-630 determining (apps), 27-29 sample code, 621-630 Memory Usage MB Graph (Windows Phone text, 625-629 Application Analysis tool), 26 UI updates when cultures change, menu items (application bar) 618-620 built-in application bar, customizing, speech-driven apps, 721-723 234-235 text support, 160-161 custom application bar, 237-238 MultiScaleImage API, 210-212 runtime, retrieving at, 236-237 MultiScaleImage element, viewing high-resolu-MessageService property and IMessageService tion images, 207-208 interface, 41-44 metadata, UTF (Unit Test Framework) and,

Deep Zoom technology, 207-217	external navigation via HyperlinkButton
Deep Zoom Viewer sample code,	control, 70
212-217	hosting web content within apps, 70-71
image creation, 208-209	internal URI, 69
image tiling, 208	URI mapping, 73-74
MultiScaleImage API, 210-212	navigation history stacks, 72-73
multitasking, background processing and, 997	.NET Isolated Storage API, 883-885
music, saving to/deleting from media library, 205-207	reading/writing data to isolated storage, 884-886
Mutex, shared resources and, 1038-1041	serialization performance, 886-887
MVVM (Model-View-ViewModel) pattern, 31	network connectivity, monitoring, 850-855
AutoCompleteBox component (Toolkit) and,	network services, 849
260-264 ContextMenu component (Toolkit) and,	Data Sense, monitoring data usage via, 877-879
273-275	hosting, 850
implementing, 31-32	HTTP services, 850
property change notification, 33	network connectivity, monitoring, 850-855
INPC implementation, alternative approach, 35-37	OData, 850, 855
INPC implementation, traditional approach, 33-35	eBay consumer application build, 862-874
ViewModelBase class, 32	EbaySearchView page, 867-871
Viewwiodelbase class, 32	EbaySearchViewModel class, 864-867
	extending OData entity classes, 874
N	fetching data when user scrolls to the end of a list, 871-873
navigating nagae 07.00	generating proxies, 858
navigating pages, 67-69	LINQ and, 855-856
canceling, 77	query options, 858-860
cross-page communication, 77-78	ScrollViewerMonitor class, 871-873
handling, 76	URI structure, 856-858
hardware Back button, 79-80	using proxies, 860-861
navigation history stacks, 72-73	wrapper creation, 862-864
NavigationService property, 75	REST services, 850
backward navigation, 76	Simulation Dashboard, 876
forward navigation, 76	SOAP services, 850
page redirection, 78-79	NFC, sharing images, 460-461
query strings, passing page arguments via, 71-73	notification text, lock screen customization, 401-403
unmapped URI, 69	notifications (push), 485
external navigation via button control,	benefits of, 486-487

cloud service authentication, 511	numbers (phone)
enabling, 489	saving, 439-442
MPNS and, 487-488, 509-511	selecting, 437-438
notification classes, 509-511	
power management and, 493-494	
raw notifications, 486, 505	0
notification classes, 510-511	O
notifications in HttpWebResponse, identifying, 509	OData, 850, 855
receiving, 508-509	eBay consumer application, building, 862
sending, 505-507	EbaySearchView page, 867-871
stock ticker application example,	EbaySearchViewModel class, 864-867
521-523	extending OData entity classes, 874
sending, 495	fetching data when user scrolls to the end of a list, 871-873
SLA and, 488	OData wrapper creation, 862-864
stock ticker application example, 511-524	ScrollViewerMonitor class, 871-873
subscribing to, 489-490	LINQ and, 855-856
binding to shell, 490-491	proxies
channel errors, handling, 492-493	generating, 858
HttpNotificationChannel events, 491-492	using, 860-861
tile notifications, 486, 500-501	query options, 858-860
notification classes, 510-511	URI structure, 856-858
sending, 501-504	opacity
stock ticker application example, 520	application bar, changing in
updating via shell tile schedules, 504-505	built-in application bar, 234
	custom application bar, 244
toast notifications, 485, 495-496	XAML element visibility and performance, 20
notification classes, 510-511	OpenType fonts, 159
receiving, 496-497 sending, 497-500	orientation
3.	page orientation
stock ticker application example, 520 Windows Phone Marketplace requirements, 489	animating entire pages when orientation changes, 108-110
notifications (scheduled), 998-1001	animating page elements when orientation changes, 106-107
alarm registration, 1001	PhoneApplicationPage class, 99-104
alarm sample, 1002-1004	setting at runtime, 105-106
reminder registration, 1005-1006	Panoramaltem control, 350
reminder sample, 1006-1010	Windows Phone Emulator display, changing, 4-5

components of, 347-348

I	
	Items layer, 349
page navigation. See <i>also</i> Pivot control; Panorama control, 67-69, 325	Panoramaltem control, 349-350
canceling, 77	Pivot control, differences and similarities, 326
cross-page communication, 77-78	sample code, 350-354
handling, 76	style assumptions, 326
hardware Back button, 79-80	things to avoid, 355
navigation history stacks, 72-73	Title layer, 349
NavigationService property, 75	Windows Phone 8 performance, 329
backward navigation, 76	Windows Phone FCL, placement within, 328
forward navigation, 76	PasswordBox element, 154
page redirection, 78-79	passwords, receiving via PasswordBox control,
query strings, passing page arguments via,	175-176
71-73	payments, making via Wallet hub, 773
unmapped URI, 69	introduction to, 774-775
external navigation via button control,	payment instruments, storing in, 775-777
69-70	Payment Instruments sample app,
external navigation via HyperlinkButton	777-789
control, 70	updating payment instruments, 789-792
hosting web content within apps, 70-71	required capabilities for wallet integration,
internal URI, 69	775
URI mapping, 73-74	performance
page orientation	apps, Windows Phone Marketplace require- ments, 21
animating	Frame Rate Counter, 20-21
entire pages when orientation changes, 108-110	Windows Phone 8, Pivot and Panorama con
page elements when orientation	trols, 329
changes, 106-107	Windows Phone Application Analysis tool,
PhoneApplicationPage class, 99-100	21, 26
OrientationChanged event, 100-102	CPU Usage Graph, 25
PhoneApplicationPage orientation prop-	execution profiling, 21-23
erty, 102-104	External Events view, 25
runtime, setting at, 105-106	Frame Rate Graph, 25
page transitions (animated), Windows Phone	GC Events, 26
Toolkit, 110-114	Image Loads, 26
panning/zooming via Map control, 590	launching, 21-23
Panorama control. See <i>also</i> Pivot control; page navigation, 325, 327-328, 347	Memory Usage MB Graph, 26
Background layer, 348-349	Performance Warnings view, 26-27
Daungrounu layer, 340-349	Storyboards, 26

D

viewing performance metrics, 24 thumbnail photos, viewing in PhotoCamera app, 672-678 XAML element visibility and performance. 19-20 phrase list grammars (speech recognition), 701-710, 731-732 Performance Warnings view (Windows Phone Application Analysis tool), 26-27 picker pages, customizing, 282-286 periodic tasks, 1013-1014 picture viewer, 631 persistent application state. See also transient photo extras applications, 631 application state, 49, 56, 97 creating, 632-645 phone calls, placing, 435-436 edge tracing extras application, 634-645 phone numbers photo share applications, 631 saving, 439-442 creating, 645-654 selecting, 437-438 photo upload share application, 647-654 PhoneApplicationPage class, page orientation, pixel density/picture quality, 632 99-100 pinch gestures (touch input), 378-380, OrientationChanged event, 100-102 383-384 PhoneApplicationPage orientation property, pitch, adjusting in Map control, 593 102-104 Pivot control. See also Panorama control: page PhoneApplicationPage.State dictionary, state navigation, 325, 327, 329 preservation and, 49 application bars, hosting multiple, 333-343 PhoneCallTask, choosers/launchers and, components of, 331-332 435-436 load events, 333 PhoneNumberChooserTask, choosers/launchers and, 437-438 LockablePivot control (Toolkit), 355 Photo Hub, selecting photos from, 454-459 Panorama control, differences and similarities, 326 PhotoChooserTask, choosers/launchers and, 454-459 placing Pivot on a page, 329-331 photos populating via data bound collections, 344-347 cameras, 655 setting active PivotItem presenters, 332 lenses, 686-689 style assumptions, 326 PhotoCamera class, 655-656, 678 things to avoid, 355 Silverlight webcam API, 655-656, 678-685 Visual Studio New Project dialog, creating Pivot applications in, 331 picture viewer, 631 Windows Phone 8 performance, 329 edge tracing extras application, 634-645 Windows Phone FCL, placement within, 328 photo extras applications, 631-645 pixel density/picture quality (picture viewer), photo share applications, 631, 645-654 632 photo upload share application, 647-654 playCommand constructor (iCommand interpixel density/picture quality, 632 face), 38-39 sharing, 460-461 position changes, monitoring (location aware taking, 450-452 apps), 566-584

power management, push notifications and, auto-launching apps via protocol associa-493-494 tions, 977 PowerSource property (DeviceStatus class). launching protocol URI, 979-980 28.30 receiving protocol launch requests, PowerSourceChanged event (DeviceStatus 977-979 class), 29-30 registering protocol associations, 977 preserving app state/settings, 47-48 reserved protocol names, 981 automatic state preservation system, builduser experience, effects on, 976 ing, 890-893 purchases, making via Wallet hub, 773 binary serialization, 898-901 introduction to, 774-775 creating property accessor delegates, payment instruments, storing in, 775-777 902-903 Payment Instruments sample app. customizing ViewModel state preserva-777-789 tion, 892-893 updating payment instruments, 789-792 identifying stateful ViewModels, 893-898 required capabilities for wallet integration, Silverlight Serializer and, 898-901 775 unwinding property lambda expressions, push notifications, 485 901-902 benefits of, 486-487 IsolatedStorageSettings class, 889-890 cloud service authentication, 511 local folder, 881-882 enabling, 489 managed storage, 882 MPNS and, 487-488, 509-511 .NET Isolated Storage API, 883-885 notification classes, 509-511 reading/writing data to isolated storage, 884-886 power management and, 493-494 serialization performance, 886-887 raw notifications, 486, 505 WinPRT Windows Storage API, 883, 887-888 notification classes, 510-511 application settings, 888-889 notifications in HttpWebResponse, identifying, 509 measuring available free space, 888 receiving, 508-509 serialization requirements, 889 sending, 505-507 previewing fonts (text), 163 stock ticker application example, product lists, displaying (Bookshop sample 521-523 app), 86-92 sending, 495 ProductsView class, Bookstore sample app, 85-86 SLA and, 488 ProductsViewModel class, Bookshop sample stock ticker application example, 511-524 app, 84-85 subscribing to, 489-490 progress indicators, 145-151 binding to shell, 490-491 ProgressBar control, 144 channel errors, handling, 492-493 property change notification (MVVM), 33-37 HttpNotificationChannel events, 491-492 protocol associations, 967-968

tile notifications, 486, 500-501
notification classes, 510-511
sending, 501-504
stock ticker application example, 520
updating via shell tile schedules,
504-505
toast notifications, 485, 495-496
notification classes, 510-511
receiving, 496-497
sending, 497-500
stock ticker application example, 520
Windows Phone Marketplace requirements,
489

Q - R

class

query strings, passing page arguments via (page navigation), 71-73
quick cards, search integration and, 983-984
application manifests, 985-987
captions, customizing, 987-989
landing pages, creating to display quick card information, 994-995
quick card target pages, defining, 989-994

Radio button, 128-131
range controls
progress indicators, 145-151
ProgressBar control, 144
RangeBase class properties, 143-144
scrollbars, 152
sliders, 151-152
raw notifications, 486, 505
notification classes, 510-511
receiving, 508-509
sending, 505-507
stock ticker application example, 521-523
reading device information. See DeviceStatus

ready-to-use grammars (speech recognition), 701-703 recording audio, 691-692 redirecting pages, page navigation, 78-79 reminders (scheduled notifications) registering, 1005-1006 sample code, 1006-1010 Repeat button, 124-126 resolution live tiles, 391 Windows Phone Emulator, changing settings resource intensive tasks, 1014 REST services, 850 restoring transient state, 55 retitling apps, Windows Phone Emulator, 6 retrieving appointments, 478-483 contacts, 464-469 device information. See DeviceStatus class reusing code, IMessageService interface, 41-44 reviews (apps), allowing in Marketplace, 426 rich content, displaying via RichTextBox control, 176-179 ringtones, creating, 461-463 Rotate Orientation button (Windows Phone Emulator), 5 routes (maps), calculating displaying via visual states in Map control, 604-610 route calculator, 601-602 searching for routes via RouteSearchView, 602-604 shortest routes between two points, 598-601 RTL, localizability support, 629-630 Run objects, TextBlock element, 155-158 Rx. 580

S	saving to-do items, 1029-1030
	todoListView XAML, 1025-1028
SaveContactTask, choosers/launchers and, 449-450	updating tiles via Scheduled Task Agent, 1035-1036
SaveEmailAddressTask, choosers/launchers and, 418-421	Screen Fill Rate Counter field (Frame Rate Counter), 22
SavePhoneNumberTask, choosers/launchers	scrollbars, 152
and, 439-442	search integration, 983
SaveRingtoneTask, choosers/launchers and, 461-463	App Instant Answer, 996
saving	configuring, 985
appointments, 483-484	application manifests, 985-987
contacts	creating landing pages to display quick card information, 994-995
contact lists, 449-450	customizing captions, 987-989
Custom Contacts Store, 469-477	defining quick card target page, 989-994
email addresses, 418-421	searching
music to media library, 205	Marketplace, 427-428
phone numbers, 439-442	web, 442-443
transient state, 53-54	SearchTask, choosers/launchers and, 442-443
scaling, Fit to Screen button (Windows Phone Emulator display window), 5	Segoe font (text), 159
scheduled notifications, 998-1001	Selector class, items controls, 139
alarm registration, 1001	semantic validation, 806
alarm sample, 1002-1004	sensors
reminder registration, 1005-1006	accelerometers, 525
reminder sample, 1006-1010	Accelerometer class, 528-529
scheduled tasks, 1010-1011, 1014-1018	calibrating, 533-536
API limitations, 1036-1038	shake detection, 537-539
background agents, 1011-1014	simulating acceleration via emulator,
debugging, 1025	529-530
periodic tasks, 1013-1014	smoothing readings, 530-533
registering, 1018-1019	compass sensors, 525, 539-545
resource intensive tasks, 1014	calibrating, 546-548
ScheduledTaskAgent, 1011-1012	compass orientation, 545-546
to-do list example, 1019-1024	gyroscope sensors, 525, 548-552
creating to-do item shell tiles,	motion sensors, 525, 552-557
1028-1029	overview of, 525-527
debugging scheduled tasks, 1025	shake detection, accelerometers and, 537-539
editing existing to-do items, 1030-1035	ShareLinkTask, choosers/launchers and, 444

ShareMediaTask, choosers/launchers and,	SOAP services, 850
460-461	social networks, posting status updates to,
ShareStatusTask, choosers/launchers and, 445	445
sharing	Solution Explorer, 3
images, 460-461	sound
	background audio playback, 1059
links with contacts, 444	audio streaming agents, 1076-1079
status updates in social networks, 445	AudioPlayerAgent class, 1060,
shell tiles. See live tiles	1064-1075
Show Resolution Settings button (Windows Phone Emulator), 5	AudioStreamingAgent class, 1060
Silverlight	AudioTrack class, 1061
Deep Zoom technology, 207-217	BackgroundAudioPlayer class, 1060-1061
Silverlight Serializer, building an automatic	controlling playback from foreground app,
state preservation system, 898-901	1070-1075
Silverlight webcam API, 655-656, 678-685	creating custom audio player agent,
Windows Phone FCL	1061-1064
Silverlight controls not in FCL, 118-119	overview of, 1060
unsupported Silverlight controls, 119	playing assembly resources via MediaStreamSource, 1078-1079
XAML, 1	live tiles, 391
XAML plus .NET, 1	media files, playing, 428-435
Simulation Dashboard, 63, 876	ringtones, creating, 461-463
SIP (Software Input Panel), 165	sound effects, 201-202
dimensions of, 165	special offers/deals, Wallet hub and, 796-802
dismissing programmatically, 167-168	speech-driven apps, 699
keyboard layouts, 165-166	speech recognition, 700-701
opening programmatically, 165-167	custom speech recognition UI, 718-721
sizing	dictation grammars, 701-703
buttons, 121-123	internationalization, 721-723
images, 185-186	
live tiles, 390-391	list grammars, 701-710, 731-732
text, 159-160	phrase list grammars, 701-710, 731-732
touch input sizing/spacing constraints, 387	ready-to-use grammars, 701-703
Windows Phone Emulator displays, 4	speech recognizer settings, 717-718
SLA (Service-Level Agreements), push notifica-	SRGS grammars, 702, 710-717
tions and, 488	web search grammars, 701-703
slider controls, 151-152	XML grammars, 701-702, 710-717
SMS, composing, 445	TTS, 732
SmsComposeTask, choosers/launchers and, 445	selecting speaking voice, 732-733 SSML files, 733-735

WinPRT Windows Storage API, 883, 887-888 voice commands, launching apps via, 723-724 application settings, 888-889 installing VCD files, 726-730 measuring available free space, 888 updating phrase lists in VCD files, serialization requirements, 889 731-732 status updates, posting to social networks, VCD file structure, 724-725 445 Voice Paint app, creating, 703-721 stock ticker application example, push notificasplash screens, 80-83 tions and, 511-524 SOL CE database files, viewing/modifying. storage 940-942 isolated storage SQL Server Compact, local databases and data deploying databases to, 943-946 storage, 906 IsolatedStorageSettings class, 889-890 SqlMetal, database-first data model creation, .NET Isolated Storage API, 883-886 942-943 SQL CE database files, 940-942 SRGS grammars (speech recognition), 702, 710-717 storing/browsing web content in, 225-228 SSML (Speech Synthesis Markup Language) files, 733-735 Windows Phone SDK Isolated Storage Explorer, 938-939 Start Experience (Windows Phone Emulator), 6, 389-390 WP7 Isolated Storage Explorer, 939-940 state preservation, 47-48 local databases, data storage and, 905 automatic state preservation system, buildcode-first data model creation, 909-910 ing. 890-893 deploying databases, 906-907 binary serialization, 898-901 deploying databases to isolated storage, creating property accessor delegates, 943-946 902-903 LINQ to SQL, 905, 908-909, 948-950, customizing ViewModel state preserva-958-965 tion, 892-893 mapping inheritance hierarchies, identifying stateful ViewModels, 893-898 958-962 Silverlight Serializer and, 898-901 navigation services, abstracting, 946-948 unwinding property lambda expressions, SQL Server Compact, 906 901-902 SqlMetal and database-first data model IsolatedStorageSettings class, 889-890 creation, 942-943 local folder, 881-882 Twitter timeline viewer example, 910-937 .NET Isolated Storage API, 883-886 updating database schemas, 951-957 WinPRT Windows Storage API, 883, versioning database schemas, 957-958 887-889 viewing database schemas, 937-942 managed storage, 882 viewing/modifying SQL CE database files, .NET Isolated Storage API, 883-885

reading/writing data to isolated storage,

serialization performance, 886-887

884-886

940-942

Explorer, 938-939

Windows Phone SDK Isolated Storage

WP7 Isolated Storage Explorer, 939-940	integration testing, 739
XAP files, 906-907	loC, 765-767
managed storage, state preservation and, 882	launchers, testing via, 769-771 trial conditions, testing, 767-769
WinPRT Windows Storage API, 883	unit testing, 737-738
Storyboards (Windows Phone Application Analysis tool), 26	UTF, 739-740
streaming audio/video content, 194	AssemblyCleanup attribute, 746
stretching text, 159	AssemblyInitialize attribute, 745-746
styling text, 159	assertions, 751-752
submitting apps to Windows Phone Marketplace, 15-16	Asynchronous attribute, 749 asynchronous testing, 762-763
subscribing to	Bug attribute, 749-750
events (life cycle of apps), 51	ClassCleanup attribute, 746
push notifications, 489-490	ClassInitialize attribute, 746
binding to shell, 490-491	creating a test project, 740-741, 747
channel errors, handling, 492-493	creating test classes, 741-743
HttpNotificationChannel events, 491-492	Description attribute, 748
Surface Counter field (Frame Rate Counter), 22	ExpectedException attribute, 749
switch color, changing via ToggleSwitch compo-	hiding Expressions Editor, 752-753
nent (Toolkit), 297	Ignore attribute, 747-748
synchronous validation	metadata, 745-751
decoupling validation, 830	Owner attribute, 748-749
provisioning for, 827-828	Priority attribute, 750
validating changing properties, 830-833	Tag attribute, 749
syntactic validation, 806	tag expressions, 743-744
	testable chat client app example, 754-764
т	TestClass attribute, 744
Т	testing multiple assemblies, 753
tag expressions in UTF (Unit Test Framework),	testing nonpublic members, 753
743-744	TestInitialize attribute, 747
tap and click events (buttons), 123	TestMethod attribute, 745
tap gestures (touch input), 369-370, 373-374,	TestProperty attribute, 747
382	Timeout attribute, 748
templates (Windows Phone App), 3	WorkItemTest class, 751
terminating apps, 53	text
testing (automated), 738-739	clipboard, placing text on, 179-180
ad hoc testing, 737	displaying, 155
choosers, testing via, 769-771	

TextBlock element, 154-155 element types, understanding, 154-155 fonts displaying text, 155 assigning, 164 LineBreak objects, 155-158 bold text, 159-160 multiline text. 158 built-in fonts, 160-162 Run objects, 155-158 colored fonts, 160 TextBox element, 154-155 embedding fonts, 162-164 TTS, 732 language support, 160-161 selecting speaking voice, 732-733 OpenType fonts, 159 SSML files, 733-735 properties of, 158-160 whitespace, handling, 155 Segoe font, 159 Texture Memory Usage field (Frame Rate Counter), 22 selecting, 159 third-party fonts (text), 162-164 sizing, 159-160 thumbnail images, viewing in PhotoCamera app, stretching, 159 672-678 styling, 159 tile notifications, 486, 500-501 third-party fonts, 162-164 notification classes, 510-511 weight adjustments, 159-160 sending, 501-504 Windows Font Preview tool, 163 stock ticker application example, 520 formatting at runtime, 176-179 updating via shell tile schedules, 504-505 inputting via TextBox control, 165 Tile options (Windows Phone Emulator), 6 input scope, 168-174 tiling images, 208 IntelliSense support, 168 TiltEffect component (Toolkit), 291-293 SIP. 165-168 TimePicker/DatePicker components (Toolkit), word prediction, 168 277-281 line breaks, 155-157 control headers, adding to TimePicker component (Toolkit), 281 localizability (internationalization), 625-629 customizing Microsoft Design Language, 153 multiline text, 155-158 full-screen picker pages, 282-286 value formats, 281-282 notification text. lock screen customization. 401-403 Title laver (Panorama control), 349 PasswordBox element, 154 titles (apps), retitling via Windows Phone Emulator, 6 passwords, receiving via PasswordBox control, 175-176 toast notifications, 485, 495-496 rich content, displaying via RichTextBox connotification classes, 510-511 trol, 176-179 receiving, 496-497 suggesting via AutoCompleteBox component sending, 497-500 (Toolkit), 259-266 stock ticker application example, 520 dynamically populating, 266-267

styling, 268-270

to-do list scheduled task example, 1019-1024 creating to-do item shell tiles, 1028-1029	customizing full-screen picker pages, 282-286
debugging scheduled tasks, 1025	customizing value formats, 281-282
editing existing to-do items, 1030-1035	flexible layouts via WrapPanel component, 297
saving to-do items, 1029-1030	child element spacing, 298
todoListView XAML, 1025-1028	sample code, 298-299
updating tiles via Scheduled Task Agent, 1035-1036	using with ListBox, 300-302
Toggle button, 124, 126-128	in-place selection, 255-258
ToggleSwitch component (Toolkit)	installing, 254
Boolean option, providing, 293-295	ListPicker component, 255-258
localizing, 295-297	LockablePivot control, 355
switch color, changing, 297	looping lists, 287-291
tombstoning, 52-53, 406-408	LoopingSelector component, 287-291
Toolkit, 253	suggesting text, 259-266
AutoCompleteBox component, 259-260	dynamically populating, 266-267
data binding, 270-271	styling, 268-270
defining custom filters, 264-266	TiltEffect component, 291-293
dynamically populating suggested lists, 266-267	Toolkit gestures (touch input), 372-373 double tap gestures, 374, 382
ItemFilter property, 264-266	drag gestures, 375-377, 383
MVVM and, 260-264	flick gestures, 377-378, 384-386
styling suggested lists, 268-270	GestureBegin events, 380-381
Boolean option, providing via ToogleSwitch	GestureCompleted events, 380-381
component, 293-295	hold gestures, 375, 382
changing switch color, 297	pinch gestures, 378-380, 383-384
localizing ToggleSwitch component,	sample code, 381-386
295-297	tap gestures, 373-374, 382
CodePlex.com, 254	touch feedback, providing, 291-293
components of, 254-255	updating, 254
ContextMenu component	ToolTip control, 138
displaying, 271-273	touch feedback, providing via Toolkit, 291-293
hosting in a ListBoxItem, 275-277	touch input
MVVM and, 273-275	designing for, 386
data binding, 270-271	design guidelines, 387-388
DatePicker/TimePicker components, 277-281	sizing/spacing constraints, 387
adding control headers to TimePicker component, 281	touch components, 386-387
	double tap gestures, 370, 374, 382
	drag gestures, 375-377, 383

flick gestures, 377-378, 384-386 hold gestures, 371-372, 375, 382 manipulation events, 357, 363-367 mouse events, 357-359, 361-362 pinch gestures, 378-380, 383-384 tap gestures, 369-370, 373-374, 382 Toolkit gestures, 372-373 double tap gestures, 374, 382 drag gestures, 375-377, 383 flick gestures, 377-378, 384-386 GestureBegin events, 380-381 GestureCompleted events, 380-381 hold gestures, 375, 382 pinch gestures, 378-380, 383-384 sample code, 381-386 tap gestures, 373-374, 382 Touch class, 360-361 TouchPoint class, 357, 360-361 mouse events, 361-362 Touch.FrameReported events, 362-363 UIElement gesture events, 357-358, 368-369 double tap gestures, 370 hold gestures, 371-372 tap gestures, 369-370 Windows Phone Emulator, 358 tracking location via Map control, 594-596 transferring files (background), 1043 app termination/resubscription to transfer events, 1048 sample code, 1048-1049 backing up local databases, 1052-1055 restoring local databases, 1055-1058 retrieving user Windows Live Anonymous IDs, 1050-1052 URL rerouting with WCF services, 1049-1050 transfer requests, 1043-1047

transient application state. See also persistent application state, 49, 97 requirements, 54-55 restoring, 55 saving, 53-54 transitions (animated), Windows Phone Toolkit, 110-114 Trigger Reminders button (Simulation Dashboard), 877 TTS (Text-to-Speech), 732 selecting speaking voice, 732-733 SSML files, 733-735 Twitter timeline viewer example (local databases). 910-912 Association attribute, 917-918 Column attribute, 912-914 connection strings, 920-924 database utilities, 919-920 leveraging custom Twitter services, 924-929 multiplicity, 917-918 one-to-many relationships, 917-918 tracking data context changes, 915 TwitterDataContext class, 918-919 TwitterUser class, 915-917 user credentials, gathering in Sign In view, 929-932 viewing tweets in Timeline view, 932-937

U

UI testing (coded), 739

UI thread (XAML animation/graphics), 18-20

UIElement gesture events (touch input), 357-358, 368-369

double tap gestures, 370

hold gestures, 371-372

tap gestures, 369-370

unit testing, 737-738

update intervals (XNA game loops), controlling, 203	items controls ComboBox control, 142
updating	ListBox control, 140-142
deals/special offers, Wallet hub and, 796-802	Selector class members, 139
live tiles, 391, 396-397	range controls
local databases schemas, 951-957	progress indicators, 145-151
localizability (internationalization), 618-620	ProgressBar control, 144
membership information, Wallet hub and, 802-804	RangeBase class properties, 143-144 scrollbars, 152
payment instruments in Wallet hub,	sliders, 151-152
789-792	sensors
phrase lists in VCD files, 731-732	accelerometers, 525, 527-539
status updates, posting to social networks, 445	compass sensors, 525, 539-548
	gyroscope sensors, 525, 548-552
stock ticker application example, 523-524 tile notifications via shell tile schedules.	motion sensors, 525, 552-557
504-505	overview of, 525-527
to-do item tiles via Scheduled Task Agent,	ToolTip control, 138
1035-1036	touch feedback, providing via Toolkit, 291-293
Toolkit, 254	touch input
upload share application (photos), creating, 647-654	designing for, 386-388
user input (controls)	double tap gestures, 370, 374, 382
buttons	drag gestures, 375-377, 383
Button Click mode, 124	flick gestures, 377-378, 384-386
check boxes, 131-138	hold gestures, 371-372, 375, 382
Hyperlink button, 124	manipulation events, 357, 363-367
icon buttons (application bar), 234-237	mouse events, 357-359, 361-362
icon buttons (custom application bar),	pinch gestures, 378-380, 383-384
237-238	tap gestures, 369-370, 373-374, 382
minimizing Icon Button Tray (application	Toolkit gestures, 372-386
bar), 234, 243	TouchPoint class, 357, 360-363
Radio button, 128-131	UIElement gesture events, 357-358,
Repeat button, 124-126	368-372
sizing, 121-123	Windows Phone Emulator, 358
tap and click events, 123	validating, 805
Toggle button, 124, 126-128	asynchronous validation, 824-847
content controls, 119-121	composite validation, 824-847
drawing surface, presenting via InkPresenter element, 186-193	defining validation visual states, 810-82

code driven UI testing, 760-763 property setters, 806-810 semantic validation, 806 TestClass attribute, 744 syntactic validation, 806 TestCleanup attribute, 747 Windows Phone FCL testing control types within, 117 multiple assemblies, 753 Silverlight controls not in FCL, 118-119 nonpublic members, 753 unsupported Silverlight controls, 119 TestInitialize attribute, 747 User Interface Thread Frame Rate (FPS) field TestMethod attribute, 745 (Frame Rate Counter), 22 TestProperty attribute, 747 user reviews (apps), allowing in Marketplace, Timeout attribute, 748 426 WorkItemTest class, 751 UTF (Unit Test Framework), 739-740 AssemblyCleanup attribute, 746 AssemblyInitialize attribute, 745-746 assertions, 751 CollectionAssert, 752 validating StringAssert, 752 arguments, 39-40 verifying collection conditions, 752 group validation, 818-820, 844-847 verifying string conditions, 752 user input, 805 Asynchronous attribute, 749 asynchronous validation, 824-847 asynchronous testing, 762-763 composite validation, 824-847 Bug attribute, 749-750 defining validation visual states, 810-824 ClassCleanup attribute, 746 property setters, 806-810 ClassInitialize attribute, 746 semantic validation, 806 Description attribute, 748 syntactic validation, 806 ExpectedException attribute, 749 vCards, converting stored contacts to/from, Expressions Editor, hiding, 752-753 478 Ignore attribute, 747-748 VCD (Voice Command Definition) files, 723-724 metadata, 745-751 installing, 726-730 Owner attribute, 748-749 structure of, 724-725 Priority attribute, 750 updating phrase lists in, 731-732 Tag attribute, 749 video tag expressions, 743-744 media files, playing, 428-435 test classes, creating, 741-743 PhotoCamera app, adding video effects to, test project, creating, 740-741 668-671 testable chat client app example, 754-758 playing via MediaElement, 193-194 automation peers, manipulating UI elecontrolling audio output, 194 ments at runtime, 763-764 sample code, 194-200 building views, 758-760

streaming content, 194	installing VCD files, 726-730
XNA SoundEffect class, 201-202	updating phrase lists in VCD files,
Silverlight webcam API, displaying video in, 684-685	731-732 VCD file structure, 724-725
ViewModelBase class (MVVM), 32	Voice Paint app, creating, 703-721
visibility	
item visibility in custom application bar, 250	
XAML element visibility and performance, 19-20	W
visual states, customizing maps via, 604-610 Visual Studio Express 2012, 2 Add New Project dialog, 2 OData, 850 Simulation Dashboard, 63, 876 SOAP services, 850 Solution Explorer, 3 Windows Phone App template, 3 XAML creating apps, 2-3	Wallet hub, 773 deals/special offers storing information, 796-802 updating information, 796-802 introduction to, 774-775 membership information storing, 792-796 updating, 802-804 payment instruments, storing in, 773, 775-777, 789-792
XAML design view, 4	required capabilities for wallet integration,
Visual Studio New Project dialog, creating Pivot applications in, 331	775 WCF services
voice-driven apps, 699	Bookshop sample app, 95-97
speech recognition, 700-701 custom speech recognition UI, 718-721 dictation grammars, 701-703 list grammars, 701-710, 731-732 phrase list grammars, 701-710, 731-732 ready-to-use grammars, 701-703 speech recognizer settings, 717-718 SRGS grammars, 702, 710-717 web search grammars, 701-703 XML grammars, 701-702, 710-717, 721-723 TTS, 732	consuming, 44-45 web (world wide) links, sending to contacts, 444 searching, 442-443 web content apps, hosting within, 70-71 WebBrowser element, displaying in, 217 communicating with web pages, 218-222 injecting behavior into web pages, 222-223 monitoring/canceling navigation, 217-218
	storing/browsing web content in isolated
selecting speaking voice, 732-733 SSML files, 733-735	storage, 225-228
voice commands, launching apps via, 723-724	web pages, navigating, 446-447

web search grammars (speech recognition),	Application class, 7-8
701-703	Close button, 5
WebBrowserTask, choosers/launchers and, 446-447	display orientation, changing, 4-5
webcam API (Silverlight), 655-656	display size, changing, 4
weight (text), adjusting, 159-160	display window, scaling via Fit to Screen but- ton, 5
whitespace, handling in text, 155	floating menu, 4
Wi-Fi triangulation and location aware apps, 562	launching, 3-4
Windows Font Preview tool, 163	Lock screen, engaging, 63
Windows Live Anonymous IDs, retrieving (back-	minimizing window, 4-5
ground file transfers), 1050-1052	resolution settings, changing, 5
Windows Mobile 6.5 apps, Windows Phone 8	Show Resolution Settings button, 5
compatibility, 1	Start Experience, 6
Windows Phone 8	Tile options, 6
performance, Pivot and Panorama controls, 329	touch input, 358
Windows Mobile 6.5 app compatibility, 1	WCF services, consuming, 44-45
Windows Phone App template, 3	XAML apps
Windows Phone Application Analysis tool, 21,	customizing, 5-7
26	debugging, 3-7
CPU Usage Graph, 25	running, 3-7
execution profiling, 21-23	Windows Phone execution model. See execution model
External Events view, 25	Windows Phone FCL
Frame Rate Graph, 25	controls
GC Events, 26	Button Click mode, 124
Image Loads, 26	check boxes, 131-138
launching, 21-23	
Memory Usage MB Graph, 26	ComboBox control, 142 content controls, 119-121 control types within FCL, 117
performance metrics, viewing, 24	
Performance Warnings view, 26-27	Hyperlink button, 124
Storyboards, 26	
Windows Phone Certification Requirements, launching events, 50	items controls, 138-142 ListBox control, 140-142
Windows Phone Emulator, 2	progress indicators, 145-151
Additional Tools window, opening, 4	ProgressBar control, 144
app customization	Radio button, 128-131
MainPage, 6, 9-11	range controls, 142-152
retitling apps, 6	Repeat button, 124-126
App List, 6	scrollbars, 152

Silverlight controls not in FCL, 118-119	Windows Phone App template, 3
sizing buttons, 121-123	XAML design view, 4
sliders, 151-152	Windows Phone Emulator, 2, 5
tap and click events, 123	App List, 6
Toggle button, 124, 126-128	Application class, 7-8
ToolTip control, 138	changing display orientation, 4-5
unsupported Silverlight controls, 119	changing display size, 4
Panorama control placement within, 328	changing resolution settings, 5
Pivot control placement within, 328	customizing XAML apps, 5-7
Windows Phone Marketplace	debugging XAML apps, 3-7
Marketplace Test Kit, determining app capa-	floating menu, 4
bilities via, 17	launching, 3-4
push notification requirements, 489	MainPage, 6, 9-11
submitting apps to	minimizing window, 4-5
app performance, 21	opening Additional Tools window, 4
capability discovery phase, 16-17	retitling apps, 6
XAP files, 15-16	running XAML apps, 3-7
Windows Phone SDK 8.0	scaling display window, 5
Frame Rate Counter, 20-21	Start Experience, 6
installing, 2	Tile options, 6
lock screen/app integration, 397-399	Windows Phone SDK Isolated Storage Explorer,
background images, 399-401	938-939
creating lock screen icons, 403-404	Windows Phone Toolkit. See Toolkit
notification text, 401-403	Windows.System.Launcher class, auto-launch-
Microsoft Expression Blend for Windows	ing apps via, 981
Phone, 2	WinPRT Windows Storage API, 883, 887-888
requirements for, 2	application settings, 888-889
Simulation Dashboard, 876	measuring available free space, 888
support for, 2	serialization requirements, 889
text	WMAppManifest.xml files, defining app capabilities, 16
formatting at runtime, 178-179	word prediction, inputting text, 168
inputting via TextBox control, 165-176	WP7 Isolated Storage Explorer, 939-940
placing on clipboard, 179-180	WrapPanel component (Toolkit), flexible layouts
rich content, displaying via RichTextBox control, 176-179	via, 297
Visual Studio Express 2012, 2	child element spacing, 298
Add New Project dialog, 2	ListBox, using with, 300-302
creating XAML apps, 2-3	sample code, 298-299
Solution Explorer, 3	

701-702,

X	XAML plus .NET, 1
•	XAML plus .NET, 1
XAML	XAP files, 14
Application class, 7-8	Application Deployment tool, 15
apps	components of, 14
creating via Visual Studio Express 2012, 2-3	defining, 14
customizing, 5-7, 9-11	local database deployments, 906-907
argument validation, 39-40	MSIL, 16-17
graphics/animation	purpose of, 14
composition thread, 18-20	requirements for, 14
UI thread, 18-20	Windows Phone Marketplace, submitting apps to, 15-16
iCommand interface, 38	XML grammars (speech recognition), 701-70
CanExecute (object) method, 38	710-717
CanExecuteChanged method, 38	XNA game loops, controlling via GameTimer class, 202-203
Commanding.Event property, 39	update intervals, controlling, 203
DelegateCommand constructor, 38-39	XNA environment initialization, 203-204
Execute (object) method, 38	XNA microphone API, 691
playCommand constructor, 38-39	helium voice app, creating, 692-698
input validation, 805	recording audio, 691-692
asynchronous validation, 824-847	XNA SoundEffect class, 201-202
composite validation, 824-847	, Coana_noot olaco, _c
defining validation visual states, 810-824	
property setters, 806-810	V 7
MVVM, 31	Y - Z
implementing, 31-32	Tooming (naming via Man control ECO
property change notification, 33-37	zooming/panning via Map control, 590
ViewModelBase class, 32	
opacity of elements, 20	
Silverlight in, 1	
todoListView XAML in to-do list scheduled task example, 1025-1028	
Toolkit, components of, 254	
visibility of elements and performance, 19-20	
Windows Phone Emulator	
debugging apps, 3-7	
running apps, 3-7	
XAML design view, 4	