## 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

**Part I  Windows Phone App Development Fundamentals**

1  **Introduction to Windows Phone App Development**  
1.1 Installing the Windows Phone SDK  
1.2 Creating Your First XAML for Windows Phone App  
1.3 The Application Class  
1.4 Summary  

2  **Fundamental Concepts in Windows Phone Development**  
2.1 Understanding the Role of XAP Files  
2.2 The Application Deployment Tool  
2.3 The Windows Phone Capabilities Model  
2.4 Determining App Capabilities Using the Marketplace Test Kit  
2.5 The Threading Model for XAML-Based Graphics and Animation in Windows Phone  
2.6 Performance and Element Visibility  
2.7 Deciding Between Visibility and Opacity  
2.8 Understanding the Frame Rate Counter  
2.9 The Windows Phone Application Analysis Tool  
2.10 External Events  
2.11 Frame Rate Graph  
2.12 CPU Usage Graph  
2.13 Memory Usage MB Graph  
2.14 Storyboards  
2.15 Image Loads  
2.16 GC Events  
2.17 Viewing Detailed Profiling Information  
2.18 Reading Device Information  
2.19 Calculating Available Memory  
2.20 DeviceStatus Events  
2.21 Applying the Model-View-ViewModel Pattern to a Windows Phone App  
2.22 Implementing the MVVM Pattern  
2.23 ViewModelBase Class  
2.24 Property Change Notification  
2.25 Implementing INotifyPropertyChanged: The Traditional Approach
Implementing INotifyPropertyChanged: An Alternative Approach
                                                                                                    35
Using Commands                                                                                         38
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                                                                                       67
    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
Part II Essential Elements

5 Using Content Controls, Items Controls, and Range Controls

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 ...................................................................... 124
Hyperlink Button ...................................................................... 124
Repeat and Toggle Buttons ......................................................... 124
Radio Button ............................................................................ 128
Check Box ............................................................................... 131
ToolTip ..................................................................................... 138
Items Controls ........................................................................... 138
ListBox ................................................................................... 140
ComboBox ................................................................................ 142
Range Controls .......................................................................... 142
ProgressBar ............................................................................. 144
Progress Indicator ..................................................................... 145
Slider ....................................................................................... 151
ScrollBar ................................................................................ 152
Summary ................................................................................... 152

6 Mastering Text Elements and Fonts ......................................... 153
Understanding Text Element Types ............................................. 154
Displaying Text Using the TextBlock ........................................ 155
Using the Run and LineBreak Objects ....................................... 155
Multiline TextBlock Text ........................................................... 158
Controlling Font Properties ....................................................... 158
Understanding the Built-In Fonts .............................................. 160
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 ......................................................................... 298
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 332
Pivot Load Events 333
Hosting Multiple Application Bars with the Pivot 333
Populating a Pivot Via a Data Bound Collection 344
Using the Panorama Control 347
Panorama Anatomy 347
PanoramaItem Control 349
Sample Panorama Application 350
Things to Avoid When Using the Panorama and Pivot 355
Windows Phone Toolkit Lockable Pivot 355
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 .................................................... 386
  Sizing and Spacing Constraints ................................................. 387
  General Guidelines ................................................................... 387
Summary ........................................................................................ 388

13 Portraying Your App with Live Tiles 389
Getting Started with Live Tiles ...................................................... 389
  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 ......................... 399
    Setting the Lock Screen’s Notification Text ............................ 401
    Creating a Lock Screen Icon .................................................. 403
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.Completing 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
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowing the User to Review Your App Using the</td>
<td>426</td>
</tr>
<tr>
<td>MarketplaceReviewTask</td>
<td></td>
</tr>
<tr>
<td>Searching the Marketplace with the MarketplaceSearchTask</td>
<td>427</td>
</tr>
<tr>
<td>Playing a Media File Using the MediaPlayerLauncher</td>
<td>428</td>
</tr>
<tr>
<td>Placing a Call with the PhoneCallTask</td>
<td>435</td>
</tr>
<tr>
<td>Selecting a Phone Number with the PhoneNumberChooserTask</td>
<td>437</td>
</tr>
<tr>
<td>Saving a Contact’s Phone Number with the</td>
<td>439</td>
</tr>
<tr>
<td>SavePhoneNumberTask</td>
<td></td>
</tr>
<tr>
<td>Searching the Web with the SearchTask</td>
<td>442</td>
</tr>
<tr>
<td>Sending Contacts a Link Using the ShareLinkTask</td>
<td>444</td>
</tr>
<tr>
<td>Posting a Status Update to Social Networks Using the</td>
<td>445</td>
</tr>
<tr>
<td>ShareStatusTask</td>
<td></td>
</tr>
<tr>
<td>Preparing an SMS With the SmsComposeTask</td>
<td>445</td>
</tr>
<tr>
<td>Navigating to a Web Page Using the WebBrowserTask</td>
<td>446</td>
</tr>
<tr>
<td>Selecting a Contact’s Address Using the AddressChooserTask</td>
<td>447</td>
</tr>
<tr>
<td>Saving a Contact to the Phone’s Contact List Using the</td>
<td>449</td>
</tr>
<tr>
<td>SaveContactTask</td>
<td></td>
</tr>
<tr>
<td>Taking a Photo with the CameraCaptureTask</td>
<td>450</td>
</tr>
<tr>
<td>Inviting Game Players with the GameInviteTask</td>
<td>453</td>
</tr>
<tr>
<td>Selecting a Photo from the Photo Hub Using the</td>
<td>454</td>
</tr>
<tr>
<td>PhotoChooserTask</td>
<td></td>
</tr>
<tr>
<td>Sharing Images with Other Devices Using NFC or with Other Registered Phone Services</td>
<td>460</td>
</tr>
<tr>
<td>Creating a Ringtone with an Audio File Using the</td>
<td>461</td>
</tr>
<tr>
<td>SaveRingtoneTask</td>
<td></td>
</tr>
<tr>
<td>Contacts and Appointments</td>
<td>464</td>
</tr>
<tr>
<td>Retrieving Contacts</td>
<td>464</td>
</tr>
<tr>
<td>Storing Contacts in the Custom Contact Store</td>
<td>469</td>
</tr>
<tr>
<td>Detecting Changes in Stored Contacts</td>
<td>477</td>
</tr>
<tr>
<td>Converting a Stored Contact to and from a vCard</td>
<td>478</td>
</tr>
<tr>
<td>Retrieving Appointments</td>
<td>478</td>
</tr>
<tr>
<td>Saving an Appointment Using the SaveAppointmentTask</td>
<td>483</td>
</tr>
<tr>
<td>Summary</td>
<td>484</td>
</tr>
<tr>
<td>15 Receiving Messages from the Cloud Using Push Notification</td>
<td>485</td>
</tr>
<tr>
<td>Push Notifications Types</td>
<td>485</td>
</tr>
<tr>
<td>Benefits of Push Notification</td>
<td>486</td>
</tr>
<tr>
<td>Understanding Push Notification</td>
<td>487</td>
</tr>
<tr>
<td>Getting Started with Push Notification</td>
<td>489</td>
</tr>
<tr>
<td>Subscribing to Push Notification</td>
<td>489</td>
</tr>
<tr>
<td>Binding to the Shell</td>
<td>490</td>
</tr>
<tr>
<td>HttpNotificationChannel Events</td>
<td>491</td>
</tr>
<tr>
<td>Handling Channel Errors</td>
<td>492</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Power Management and Push Notification</td>
<td>493</td>
</tr>
<tr>
<td>Sending Push Notifications</td>
<td>495</td>
</tr>
<tr>
<td>Toast Notifications</td>
<td>495</td>
</tr>
<tr>
<td>Receiving a Toast Notification from Within an Application</td>
<td>496</td>
</tr>
<tr>
<td>Sending a Toast Notification</td>
<td>497</td>
</tr>
<tr>
<td>Tile Notifications</td>
<td>500</td>
</tr>
<tr>
<td>Sending a Tile Notification</td>
<td>501</td>
</tr>
<tr>
<td>Updating an Application Tile Using a Shell Tile Schedule</td>
<td>504</td>
</tr>
<tr>
<td>Raw Notifications</td>
<td>505</td>
</tr>
<tr>
<td>Sending a Raw Notification</td>
<td>505</td>
</tr>
<tr>
<td>Receiving a Raw Notification</td>
<td>508</td>
</tr>
<tr>
<td>Identifying Notifications in an HttpWebResponse</td>
<td>509</td>
</tr>
<tr>
<td>Notification Classes</td>
<td>509</td>
</tr>
<tr>
<td>Cloud Service Authentication</td>
<td>511</td>
</tr>
<tr>
<td>Authenticating a Cloud Service</td>
<td>511</td>
</tr>
<tr>
<td>Creating a Notification Channel for an Authenticated Cloud Service</td>
<td>511</td>
</tr>
<tr>
<td>Building a Stock Ticker Application</td>
<td>511</td>
</tr>
<tr>
<td>Sample App Notifications</td>
<td>520</td>
</tr>
<tr>
<td>Summary</td>
<td>524</td>
</tr>
</tbody>
</table>

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 | 561 |
| Wi-Fi Triangulation | 562 |
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 ........................... 610
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  620
- The Resx Localizability Sample  621
  - Controlling the UI Culture from the `LocalizabilityViewModel`  622
  - Displaying Localized Text and Images Within the `LocalizabilityViewPage`  625
- RTL Support  629

## Summary

> 630

## Extending the Windows Phone Picture Viewer

- 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

## Capturing Images and Video with the Camera

- 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

## Recording Audio with the Microphone

- Recording Audio with the Microphone  691
  - Creating a Helium Voice App  692

## Creating Speech Driven Experiences

- User Input with Speech Recognition  700
  - Getting Started with Speech Recognition  700
  - Overview of Recognition Grammars  701
  - Recognizing Speech Using the Dictation and Web Search Grammars  702
  - Recognizing Speech with Phrase List Grammars  703
<table>
<thead>
<tr>
<th>Recognizing Speech with SRGS Grammars</th>
<th>710</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech Recognizer Settings</td>
<td>717</td>
</tr>
<tr>
<td>Providing a Custom Speech Recognition UI</td>
<td>718</td>
</tr>
<tr>
<td>Recognizing Speech in Other Languages</td>
<td>721</td>
</tr>
<tr>
<td>Launching Your App via Voice Commands</td>
<td>723</td>
</tr>
<tr>
<td>Understanding the Structure of the VCD File</td>
<td>724</td>
</tr>
<tr>
<td>Installing a VCD</td>
<td>726</td>
</tr>
<tr>
<td>Updating a Phrase List Programatically</td>
<td>731</td>
</tr>
<tr>
<td>Making the Phone Speak with Text-to-Speech</td>
<td>732</td>
</tr>
<tr>
<td>Getting Started with TTS</td>
<td>732</td>
</tr>
<tr>
<td>Creating a Speech Synthesis Markup Language File</td>
<td>733</td>
</tr>
<tr>
<td>Summary</td>
<td>735</td>
</tr>
</tbody>
</table>

## 24 Unit Testing Apps

Automated Testing ................................................. 738
Unit Testing ......................................................... 738
Integration Testing ............................................... 739
Coded UI Testing .................................................... 739
Introduction to the Windows Phone Unit Test Framework 739
Creating a Test Project ............................................. 740
Creating a Test Class .............................................. 741
Tag Expressions ....................................................... 743
Setting the Tag Expression Programatically .............. 744
Metadata and Assertions .......................................... 744
TestClass Attribute ............................................... 744
TestMethod Attribute .............................................. 745
Metadata for Test Initialization and Cleanup ............. 745
Miscellaneous Metadata .......................................... 747
WorkItemTest: The Base TestClass Type ..................... 751
Verifying Conditions with Assertions ..................... 751
Verifying Collection Conditions with CollectionAssert 752
Verifying String Conditions with StringAssert ........... 752
Hiding the Expressions Editor ................................. 752
Testing Multiple Assemblies .................................... 753
Testing Nonpublic Members ...................................... 753
A Testable Chat Client ............................................ 754
Building the View ................................................. 758
Code Driven UI Testing ......................................... 760
Using Automation Peers to Manipulate UI Elements at Runtime 763
Inversion of Control (IoC) ....................................... 765
A Custom IoC Container and DI Framework .................. 766
Testing Trial Conditions ......................................... 767
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 ...................................................... 867
Fetching Data When the User Scrolls to the End of a List .......... 871
    Extending OData Entity Classes ...................................... 874
Simulating Real-World Conditions with the Simulation Dashboard ..... 876
Moderating Data Usage with Data Sense .................................. 877
    Using the Data Sense API ................................................ 877
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 ...................................... 889
Building an Automatic State Preservation System ................. 890
    Customizing ViewModel State Preservation ....................... 892
    Automatic State Preservation Inner Workings ................... 893
    Unwinding a Property Lambda Expression ....................... 901
    Creating Property Accessor Delegates .............................. 902
Summary ............................................................................ 903

29 Storing App Data in a Local Database 905

SQL Server Compact .............................................................. 906
Deployment of Local Databases .............................................. 906
LINQ to SQL on the Phone ................................................... 908
LINQ to SQL Platform Differences ......................................... 909
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started with Local Databases</td>
<td>909</td>
</tr>
<tr>
<td>Code-First Data Model Creation</td>
<td>909</td>
</tr>
<tr>
<td>Sample Twitter Timeline Viewer</td>
<td>910</td>
</tr>
<tr>
<td>Using the <code>Column</code> Attribute</td>
<td>912</td>
</tr>
<tr>
<td>Data Context Change Tracking</td>
<td>915</td>
</tr>
<tr>
<td>TwitterUser Class</td>
<td>915</td>
</tr>
<tr>
<td>Multiplicity and the <code>Association</code> Attribute</td>
<td>917</td>
</tr>
<tr>
<td>Twitter DataContext</td>
<td>918</td>
</tr>
<tr>
<td>Database Utilities</td>
<td>919</td>
</tr>
<tr>
<td>Connection Strings</td>
<td>920</td>
</tr>
<tr>
<td>Leveraging a Custom Twitter Service</td>
<td>924</td>
</tr>
<tr>
<td>Gathering the User’s Credentials with the Sign-In View</td>
<td>929</td>
</tr>
<tr>
<td>Viewing Tweets with the Timeline View</td>
<td>932</td>
</tr>
<tr>
<td>Viewing a Local Database Schema</td>
<td>937</td>
</tr>
<tr>
<td>File Explorers for Isolated Storage</td>
<td>938</td>
</tr>
<tr>
<td>Viewing and Modifying an SQL CE Database File</td>
<td>940</td>
</tr>
<tr>
<td>Database-First Using SqlMetal</td>
<td>942</td>
</tr>
<tr>
<td>Deploying a Database to Isolated Storage</td>
<td>943</td>
</tr>
<tr>
<td>Abstracting the Navigation Service</td>
<td>946</td>
</tr>
<tr>
<td>Observing LINQ to SQL Queries with a Custom Log</td>
<td>948</td>
</tr>
<tr>
<td>Updating a Database Schema</td>
<td>951</td>
</tr>
<tr>
<td>- <code>AddColumn</code></td>
<td>952</td>
</tr>
<tr>
<td>- <code>AddTable</code></td>
<td>952</td>
</tr>
<tr>
<td>- <code>AddIndex</code></td>
<td>954</td>
</tr>
<tr>
<td>- <code>AddAssociation</code></td>
<td>955</td>
</tr>
<tr>
<td>- Schema Versioning</td>
<td>957</td>
</tr>
<tr>
<td>Mapping an Inheritance Hierarchy</td>
<td>958</td>
</tr>
<tr>
<td>Concurrency</td>
<td>962</td>
</tr>
<tr>
<td>- <code>ColumnAttribute.UpdateCheck</code></td>
<td>965</td>
</tr>
<tr>
<td>Summary</td>
<td>966</td>
</tr>
</tbody>
</table>

### 30 Auto-Launching with File and Protocol Associations  
967

- Understanding File and Protocol Associations                          | 968  |
- A Note About the Sample                                               | 968  |
- Auto-Launching with a File Extension Association                      | 968  |
  - Registering a File Association                                      | 968  |
  - Receiving a File Launch Request                                     | 970  |
  - Launching a File                                                    | 973  |
- How File and Protocol Associations Affect the User Experience          | 976  |
- Auto-Launching with a Protocol Association                            | 977  |
  - Registering for a Protocol Association                              | 977  |
  - Receiving a Protocol Launch Request                                 | 977  |
Contents
xxi

34 Coordinating Background Audio Playback 1059
   Backing Up the Local Database ................................. 1052
   Restoring the Local Database ..................................... 1055
   Summary .................................................................. 1058

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).

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):

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.
This page intentionally left blank
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
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).

![Application Deployment Tool](image)

**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.
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 7.5\(^1\) 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 `UIElement`’s `RenderTransform` and `Projection` properties. Typically these animations include the following from the `System.Windows.Media` namespace:

<table>
<thead>
<tr>
<th>Transform Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlaneProjection</td>
</tr>
<tr>
<td>RotateTransform</td>
</tr>
<tr>
<td>ScaleTransform</td>
</tr>
<tr>
<td>TranslateTransform</td>
</tr>
</tbody>
</table>

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 `UIElement.Opacity` and `UIElement.Clip` 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 `UIElement.Opacity` 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 `Visibility` property, the element is still retained in visual memory.

**NOTE**

Avoid manipulating the `UIElement.Opacity` property without enabling bitmap caching. Set the `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 `UIElement.Visibility` 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:

```csharp
if (System.Diagnostics.Debugger.IsAttached)
{
    // Display the current frame rate counters.
}
```
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.

![Frame Rate Counter](image)

**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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition (Render) Thread Frame Rate (FPS)</td>
<td>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.</td>
</tr>
<tr>
<td>User Interface Thread Frame Rate (FPS)</td>
<td>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.</td>
</tr>
<tr>
<td>Texture Memory Usage</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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 480x800 pixels. The recommended value is about 2.5. The text in this counter turns red when displaying a value higher than 3.


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.
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.
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

<table>
<thead>
<tr>
<th>Color</th>
<th>Thread</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>UI Thread</td>
<td>Green shading indicates screen updates and touch input. You should aim to keep the UI thread to less than 50% of CPU usage.</td>
</tr>
<tr>
<td>Purple</td>
<td>App Threads</td>
<td>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.</td>
</tr>
<tr>
<td>Gray</td>
<td>System Threads</td>
<td>Gray indicates activity that is independent of your app, such as background agent activity.</td>
</tr>
<tr>
<td>White</td>
<td>Idle Threads</td>
<td>White indicates the available CPU percentage. The higher the idle thread percentage, the more responsive the app should be.</td>
</tr>
</tbody>
</table>
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.
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationCurrentMemoryUsage</td>
<td>The memory usage of the current application in bytes.</td>
</tr>
<tr>
<td>ApplicationMemoryUsageLimit</td>
<td>The maximum additional amount of memory, in bytes, that your application process can allocate.</td>
</tr>
<tr>
<td>ApplicationPeakMemoryUsage</td>
<td>The peak memory usage of the current application in bytes.</td>
</tr>
<tr>
<td>DeviceFirmwareVersion</td>
<td>The firmware version running on the device.</td>
</tr>
<tr>
<td>DeviceHardwareVersion</td>
<td>The hardware version running on the device.</td>
</tr>
<tr>
<td>DeviceManufacturer</td>
<td>The device manufacturer name.</td>
</tr>
<tr>
<td>DeviceName</td>
<td>The device name.</td>
</tr>
<tr>
<td>DeviceTotalMemory</td>
<td>The physical RAM size of the device in bytes.</td>
</tr>
<tr>
<td>IsKeyboardDeployed</td>
<td>If true the user has deployed the physical hardware keyboard of the device.</td>
</tr>
</tbody>
</table>
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 comprehensible (see Figure 2.8).

![DeviceStatusView](image.png)

**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:

```csharp
long requiredBytesEstimate = 10 * 1048576; /* 1048576 bytes equals 1 megabyte. */
if (DeviceStatus.ApplicationMemoryUsageLimit >= DeviceStatus.ApplicationCurrentMemoryUsage + requiredBytesEstimate)
{
    /* Perform expensive task. */
}
```

**NOTE**

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

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:

```csharp
DeviceStatus.KeyboardDeployedChanged += HandleKeyboardDeployedChanged;
```
The event handler can be used to determine whether the keyboard is deployed using the `DeviceStatus` class, as shown:

```csharp
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 `PowerSourceChanged` event allows you to detect when the user attaches or detaches an external power supply.

The `PowerSourceChanged` event can be subscribed to as shown:

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

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

```csharp
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 DataContext 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 DataContext property:

```csharp
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 NotifyPropertyChangedBase class, which provides for property change notification, discussed in the next section.
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 DependencyObject 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:

```csharp
public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(
```
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:

```csharp
public event PropertyChangedEventHandler PropertyChanged = delegate {}
```

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

```csharp
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 UnauthorizedAccessException will ensue. Source properties can be set from non-UI threads using the application’s Dispatcher as shown in the following excerpt:

```
    delegate
    {
        Foo = "bah";
    });
```

There are a number of reasons why peppering your code with BeginInvoke 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 OnPropertyChanged method to every class that implements INotifyPropertyChanged (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 INotifyPropertyChanging (as well as INotifyPropertyChanged).

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:

```csharp
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 `NotifyPropertyChangedBase` 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).

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 `Dispatcher` 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:

```csharp
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:

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

```csharp
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.

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:

```xml
<Image Source="/Foo.png"
    c:Commanding.Command="{Binding ViewCommand}"
    c:Commanding.CommandParameter="{Binding FullScreen}" />
```

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.
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 `ArgumentValidator.AssertNotNull` method is as follows:

```csharp
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:

```csharp
public static double AssertLessThan(
    double comparisonValue, double value, string parameterName)
{
    if (value >= comparisonValue)
    {
        throw new ArgumentOutOfRangeException(
            "Parameter should be less than ",
            + comparisonValue, parameterName);
    }

    return value;
}
```

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

```csharp
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).

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).
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 technology-agnostic enum values.

**LISTING 2.1** MessageService Class (excerpt)

```csharp
public partial class MessageService : MessageServiceBase
{
    public override MessageResult ShowCustomDialog(
        string message,
        string caption,
        MessageBoxButton 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 IMessageService in greater detail in the next chapter.
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:

```xml
<site name="WPUnleashed.Web" id="22">
  <application path="/" applicationPool="Clr4IntegratedAppPool">
    <virtualDirectory path="/" physicalPath="C:\Development\Source\WP8Unleashed\Source\MainExamples\Web" />
  </application>
  <bindings>
    <binding protocol="http" bindingInformation="*:27571:localhost" />
    <binding protocol="http" bindingInformation="*:27571:YourMachineName" />
  </bindings>
</site>
```

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:
**Consuming Local Web Applications**

```
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**

<table>
<thead>
<tr>
<th>Path</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:[Path to Samples]\WP8Unleashed\Source\MainExamples\Web\</td>
<td>27571</td>
</tr>
<tr>
<td>C:[Path to Samples]\WP8Unleashed\Source\Wallet\WalletWcfServices\</td>
<td>14122</td>
</tr>
<tr>
<td>C:[Path to Samples]\WP8Unleashed\Source\BackgroundAgents\Web\</td>
<td>60182</td>
</tr>
</tbody>
</table>

**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.
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 orientation changes, 106-107
Page transitions, 110-114
XAML
  composition thread, 18-20
  UI thread, 18-20
App Instant Answer, 996
App List (Windows Phone Emulator), 6
App.RootFrame_Navigated method (Fast App Resume), 60-61
App.RootFrame_Navigating method (Fast App Resume), 59-61
Application Analysis tool, 21, 26
  CPU Usage Graph, 25
  execution profiling, 21-23
  External Events view, 25
  Frame Rate Graph, 25
  GC Events, 26
  Image Loads, 26
  launching, 21-23
  Memory Usage MB Graph, 26
  performance metrics, viewing, 24
  Performance Warnings view, 26-27
Storyboards, 26
application bar, 231, 252
  built-in application bar, 231-233
    customizing colors, 234
    expanding, 233
    hiding, 233
    icon buttons, 234-236
    menu items, 234-237
    minimizing Icon Button Tray, 234
    system tray dimensions, 234
  custom application bar, 237
    AppBar control, 248-252
    customizing colors, 246-247
    customizing system tray, 247-248
    disabling application bar menu, 242-243
    extending icon base classes, 250-252
    extending menu item base classes, 250-252
    icon buttons, 237-238
    item visibility, 250
    menu items, 237-238
    minimizing Icon Button Tray, 243
    opacity, 244
    sample code, 238-242
    switching to full-screen, 244-246
  multiple applications bars, hosting via Pivot control, 333-343
Application class, 7-8
Application Deployment tool, 15
application state, 49
  persistent state, 56, 97
  transient state, 97
    requirements, 54-55
    restoring, 55
    saving, 53-54
ApplicationCurrentMemoryUsage property (Device Status class), 28
ApplicationMemoryUsageLimit property (DeviceStatus class), 27-29
ApplicationPeakMemoryUsage property (DeviceStatus class), 28
appointments, 464
  retrieving, 478-483
  saving, 483-484
apps
  animated page transitions, 110-114
Application class, 7-8
auto-launching via
  file associations, 968-975
  protocol associations, 977-981
  Windows.System.Launcher class, 981
background file transfers, 1043
  app termination/resubscription to transfer events, 1048
  sample code, 1048-1049
  transfer requests, 1043-1047
background processing, 997
  background tasks, 998
architecture of 1085

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

How can we make this index more useful? Email us at indexes@samspublishing.com
architecture of

showing locations via, 610-611

Marketplace
allowing app user reviews, 426
launching apps from, 425-426
navigating to apps in, 422-425

memory usage, determining, 27-29

page navigation, 67-69
canceling, 77
cross-page communication, 77-78
handling, 76
hardware Back button, 79-80
navigation history stacks, 72-73
NavigationService property, 75-76
page redirection, 78-79
passing page arguments via query strings, 71-73
unmapped URI, 69-71
URI mapping, 73-74

page orientation
animating entire pages when orientation changes, 108-110
animating page elements when orientation changes, 106-107
PhoneApplicationPage class, 99-104
setting at runtime, 105-106

performance, Windows Phone Marketplace requirements, 21

PhotoCamera app, building, 658-668
thumbnails, viewing, 672-678
video effects, 668-671
protocol associations, 967-968
auto-launching apps via protocol associations, 977-981
effects on user experience, 976
retitling, 6
running under Lock screen, 63-64
LockScreenService class, 64-66
LockScreenViewModel class, 66-67

Rx and, 580-584

search integration, 983
App Instant Answer, 996
application manifests, 985-987
configuring, 985-995
creating landing pages to display quick card information, 994-995
customizing captions, 987-989
defining quick card target page, 989-994

speech-driven apps, 699
launching apps via voice commands, 723-732
speech recognition, 700-723
TTS, 732-735
Voice Paint app creation, 703-721

splash screens, 80-83
state preservation, 47-48
automatic state preservation system build, 890-903
IsolatedStorageSettings class, 889-890
local folder, 881-882
managed storage, 882
.NET Isolated Storage API, 883-886
persistent state, 49, 56, 97
transient state, 49, 53-55, 97
WinPRT Windows Storage API, 883, 887-889

terminating apps, 53
testing via UTF, 739-740
AssemblyCleanup attribute, 746
AssemblyInitialize attribute, 745-746
assertions, 751-752
Asynchronous attribute, 749
asynchronous testing, 762-763
Bug attribute, 749-750
ClassCleanup attribute, 746
ClassInitialize attribute, 746
creating a test project, 740-741, 747
creating test classes, 741-743
Description attribute, 748
ExpectedException attribute, 749
hiding Expressions Editor, 752-753
Ignore attribute, 747-748
metadata, 745-751
Owner attribute, 748-749
Priority attribute, 750
Tag attribute, 749
tag expressions, 743-744
testable chat client app example, 754-764
TestClass attribute, 744
testing multiple assemblies, 753
testing nonpublic members, 753
TestInitialize attribute, 747
TestMethod attribute, 745
TestProperty attribute, 747
Timeout attribute, 747
WorkItemTest class, 751
unit testing, 737-738
WCF services, consuming, 44-45
web content, hosting within apps, 70-71
Windows Phone Application Analysis tool, 21, 26
    CPU Usage Graph, 25
    execution profiling, 21-23
    External Events view, 25
    Frame Rate Graph, 25
    GC Events, 26
    Image Loads, 26
    launching, 21-23
    Memory Usage MB Graph, 26
    Performance Warnings view, 26-27
    Storyboards, 26
    viewing performance metrics, 24
Windows Phone Marketplace
    capability discovery phase, 16-17
    submitting apps to, 15-16
WMAppManifest.xml files, defining app capabilities, 16

argument validation, 39-40
assertions (UTF), 751
    collection conditions, verifying, 752
    CollectionAssert, 752
    string conditions, verifying, 752
    StringAssert, 752
associations
    file associations, 967-968
        auto-launching apps via file associations, 968-975
        effects on user experience, 976
    protocol associations, 967-968
        auto-launching apps via protocol associations, 977-981
        effects on user experience, 976
asynchronous testing (UTF), 762-763
asynchronous validation, 824
decoupling validation, 830
example of, 836-840
group validation, 844-847
INotifyDataErrorInfo interface, 824, 842-844
    provisioning for, 827-830
    reusable NotifyDataErrorInfo interface, 825-827
    validating all properties, 833-836
    validating changing properties, 830-833
    ValidationSummary control, adding
        INotifyDataErrorInfo support for, 842-844
audio
    background audio playback, 1059
        audio streaming agents, 1076-1079
        AudioPlayerAgent class, 1060, 1064-1075
        AudioStreamingAgent class, 1060
        AudioTrack class, 1061
        BackgroundAudioPlayer class, 1060-1061
        controlling playback from foreground app, 1070-1075

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

testing multiple assemblies, 753
testing nonpublic members, 753

TestInitialize attribute, 747

TestMethod attribute, 745

TestProperty attribute, 747

Timeout attribute, 748

WorkItemTest class, 751

B

Back button, page navigation via, 79-80

background agents, 1011-1012, 1059-1060

background audio playback, 1059

assembly resources, playing via MediaStreamSource, 1078-1079

audio streaming agents, 1076-1079

AudioPlayerAgent class, 1060, 1064-1075

AudioStreamingAgent class, 1060

AudioTrack class, 1061

BackgroundAudioPlayer class, 1060-1061

custom audio player agent, creating, 1061-1064

diagram of foreground app, controlling playback from, 1070-1075

overview of, 1060

background file transfers, 590

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

background images (lock screen), setting, 399-401

Background layer (Panorama control), 348-349

background processing, 997

background tasks, 998

battery life and, 997

multitasking and, 997

Mutex, shared resources and, 1038-1041

scheduled notifications, 998-1001

alarm registration, 1001

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

debugging, 1025

periodic tasks, 1013-1014

registering, 1018-1019

resource intensive tasks, 1014

scheduled task agents, 1011-1012, 1017-1018, 1035-1036

to-do list example, 1019-1035

banking, Wallet hub and, 773

introduction to, 774-775

payment instruments, storing in, 775-777

Payment Instruments sample app, 777-789

updating payment instruments, 789-792

required capabilities for wallet integration, 775

battery life (power management)

background processing and, 997

push notifications and, 493-494

BingMapsDirectionTask, choosers/launchers and, 409-412

BingMapsTask, choosers/launchers and, 413

bold text, 159-160

Bookshop sample app, 83

design-time data, 92-93

image caching, 93-95

product lists, displaying, 86-92

How can we make this index more useful? Email us at indexes@samspublishing.com
productsView class, 85-86
ProductsViewModel class, 84-85
WCF services, 95-97
Boolean option, providing via ToggleSwitch component (Toolkit), 293-295
localizing ToggleSwitch component, 295-297
switch color, changing, 297
built-in fonts (text), 160-162
buttons
  Button Click mode, 124
  check boxes, 131-138
  Hyperlink button, 124
  Icon Button Tray (application bar), minimizing, 234
  icon buttons (application bar)
    custom application bar, 237-238
    customizing, 234-235
    disabling, 235
    retrieving at runtime, 236-237
Radio button, 128-131
Repeat button, 124-126
sizing, 121-123
tap and click events, 123
Toggle button, 124, 126-128
buying items, Wallet hub and, 773
  introduction to, 774-775
  payment instruments, storing in, 775-777
    Payment Instruments sample app, 777-789
  updating payment instruments, 789-792
required capabilities for wallet integration, 775
cameras, 655
lenses, 686-687
  app registration, 687-689
  icon creation, 689
PhotoCamera class, 655-656
  building PhotoCamera app, 658-678
  selection criteria, 656
  thumbnails, viewing in PhotoCamera app, 672-678
  video effects, adding to PhotoCamera app, 668-671
Silverlight webcam API, 655-656, 678-685
taking photos, 450-452
canceling page navigation, 77
CanExecute (object) method (ICommand interface), 38
CanExecuteChanged method (ICommand interface), 38
capabilities (apps)
  app submissions to Windows Phone Marketplace, capability discovery phase, 16-17
defining, 16
ID_CAP_NETWORK capability, 16
ID_HW_FFCCAMERA capability, 16
Marketplace Test Kit, determining via, 17
cartographic modes (Map control), 589-590
cell tower triangulation and location aware apps, 561-562
certification requirements, launching events, 50
chat client app UTF testing example, 754-758
  automation peers, manipulating UI elements at runtime, 763-764
code driven UI testing, 760-763
  views, building, 758-760
check boxes, 131-138
choosers, 405, 408-409, 484
  AddressChooserTask, 447-449
API overview, 405-406
application execution model, 406-408
appointments, 464
  retrieving, 478-483
  saving, 483-484
automated testing, 769-771
BingMapsDirectionTask, 409-412
BingMapsTask, 413
CameraCaptureTask, 450-452
ChooserBase.Completed events, 406-408
connection setting pages, navigating to, 413-414
ConnectionSettingsTask, 413-414
contacts, 464
  converting stored contacts to/from vCards, 478
  deleting changes to stored contacts, 477-478
  retrieving, 464-469
  saving to contact lists, 449-450
  selecting addresses, 447-449
  storing in Custom Contacts Store, 469-477
driving directions, retrieving, 409-412
e-mail
  composing, 417-418
  saving contact addresses, 418-421
  selecting email addresses, 414-417
EmailAddressChooserTask, 414-417
EmailComposeTask, 417-418
GameInviteTask, 453
games, inviting players to, 453
images, sharing, 460-461
links, sending to contacts, 444
map locations, displaying, 413
Marketplace
  allowing app user reviews, 426
  launching apps from, 425-426
  navigating to apps in, 422-425
  searching, 427-428
MarketplaceDetailTask, 422-425
MarketplaceHubTask, 425-426
MarketplaceReviewTask, 426
MarketplaceSearchTask, 427-428
media files, playing, 428-435
MediaPlayerLauncher, 428-435
phone calls, placing, 435-436
phone numbers
  saving, 439-442
  selecting, 437-438
PhoneCallTask, 435-436
PhoneNumberChooserTask, 437-438
Photo Hub, selecting photos from, 454-459
PhotoChooserTask, 454-459
photos, taking, 450-452
ringtones, creating, 461-463
SaveContactTask, 449-450
SaveEmailAddressTask, 418-421
SavePhoneNumberTask, 439-442
SaveRingtoneTask, 461-463
SearchTask, 442-443
ShareLinkTask, 444
ShareMediaTask, 460-461
ShareStatusTask, 445
SMS, composing, 445
SmsComposeTask, 445
status updates, posting to social networks, 445
tombstoning and, 406-408
web pages, navigating, 446-447
web searches, 442-443
WebBrowserTask, 446-447
click events (buttons), 123
clipboard, placing text on, 179-180
Close button (Windows Phone Emulator), 5
cloud services, authenticating, 511
club memberships, Wallet hub and, 773
  introduction to, 774-775

How can we make this index more useful? Email us at indexes@samspublishing.com
club memberships, Wallet hub and membership information
storing, 792-796
updating, 802-804
required capabilities for wallet integration, 775
coded UI testing, 739
CodePlex.com, Toolkit source code, 254
color
application bar color, customizing
built-in application bar, 234
custom application bar, 246-247
colored fonts (text), 160
switch color, changing via ToggleSwitch component (Toolkit), 297
ComboBox control, 142
Commanding.Event property (iCommand interface), 39
commands, iCommand interface
CanExecute (object) method, 38
CanExecuteChanged method, 38
Commanding.Event property, 39
DelegateCommand constructor, 38-39
Execute (object) method, 38
playCommand constructor, 38-39
compass sensors, 525, 539-545
calibrating, 546-548
compass orientation, 545-546
compatibility, Windows Mobile 6.5 apps and Windows Phone 8.1
composite validation
decoupling validation, 830
element of, 836-840
group validation, 844-847
INotifyDataErrorInfo interface, 824, 842-844
provisioning for, 827-830
reusable INotifyDataErrorInfo interface, 825-827
validating
all properties, 833-836
changing properties, 830-833
ValidationSummary control, adding
INotifyDataErrorInfo support for, 842-844
Composition (Render) Thread Frame Rate (FPS) field (Frame Rate Counter), 22
composition thread (XAML animation/graphics), 18-20
concurrency, LINQ to SQL and, 962-965
connection setting pages, navigating to, 413-414
ConnectionSettingsTask, choosers/launchers and, 413-414
connectivity (network), monitoring, 850-855
contacts, 464
contact lists, saving contacts to, 449-450
Custom Contacts Store, storing in, 469-477
retrieving, 464-469
stored contacts
converting to/from vCards, 478
deleting changes to, 477-478
content controls, Windows Phone FCL, 119-121
ContextMenu component (Toolkit)
displaying, 271-273
hosting in a ListBoxItem, 275-277
MVVM and, 273-275
control headers, adding to TimePicker component (Toolkit), 281
controls
buttons
Button Click mode, 124
check boxes, 131-138
Hyperlink button, 124
Radio button, 128-131
Repeat button, 124-126
sizing, 121-123
tap and click events, 123
ToggleButton button, 124, 126-128
content controls, 119-121
FrameworkElement class, 116
items controls
  ComboBox control, 142
  ListBox control, 140-142
  Selector class members, 139
range controls
  progress indicators, 145-151
  ProgressBar control, 144
  RangeBase class properties, 143-144
  scrollbars, 152
  sliders, 151-152
sensors
  accelerometers, 525, 527-539
  compass sensors, 525, 539-548
  gyroscope sensors, 525, 548-552
  motion sensors, 525, 552-557
  overview of, 525-527
ToolTip control, 138
touch input
  designing for, 386-388
  double tap gestures, 370
  hold gestures, 371-372
  manipulation events, 357, 363-367
  mouse events, 357-359, 361-362
  tap gestures, 369-370
  Toolkit gestures, 291-293, 372-386
  TouchPoint class, 357, 360-363
  UIElement gesture events, 357-358, 368-372
  Windows Phone Emulator, 358
Windows Phone FCL
  control types within, 117
  Silverlight controls not in FCL, 118-119
  unsupported Silverlight controls, 119
CPU Usage Graph (Windows Phone Application Analysis tool), 25
credit/debit cards, Wallet hub and, 773
  introduction to, 774-775
  payment instruments, storing in, 789-792
data binding
  AutoCompleteBox component (Toolkit), 270-271
  Pivot control, populating via data bound collections, 344-347
Data Sense, monitoring data usage via, 877-879
databases (local), data storage and, 1093
  required capabilities for wallet integration, 775
  storing in Wallet hub, 775-789
cross-page communication and page navigation, 77-78
cross-platform development, IMessageService interface, 41-44
Custom Contacts Store, storing contacts in, 469-477
customizing
  application bar, 234, 246-247
  DatePicker/TimePicker components
    full-screen picker pages, 282-286
    value formats in, 281-282
  maps, 604-610
  picker pages, 282-286
  switch color, changing via ToggleSwitch component (Toolkit), 297
  system tray, custom application bar, 247-248
cycle tiles, 390-394

D

data binding
  AutoCompleteBox component (Toolkit), 270-271
  Pivot control, populating via data bound collections, 344-347
Data Sense, monitoring data usage via, 877-879
databases (local), data storage and, 905
  database schemas
    updating, 951-957
    versioning, 957-958
    viewing, 937-942
deploying databases, 906-907
  isolated storage, deploying databases to, 943-946
databases (local), data storage and

LINQ to SQL, 905, 908
  code-first data model creation, 909-910
  concurrency, 962-965
  mapping inheritance hierarchies, 958-962
  platform differences, 909
  viewing queries via custom log, 948-950

navigation services, abstracting, 946-948

SQL Server Compact, 906

SqlMetal and database-first data model creation, 942-943

Twitter timeline viewer example, 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

XAP files, 906-907

DatePicker/TimePicker components (Toolkit), 277-281
  control headers, adding to TimePicker component (Toolkit), 281
  customizing
    full-screen picker pages, 282-286
    value formats, 281-282

deactivating events (life cycle of apps), 52-53
deals/special offers, Wallet hub and, 796-802
debit/credit cards, Wallet hub and, 773
  introduction to, 774-775
  payment instruments, storing in, 789-792

required capabilities for wallet integration, 775
  storing in Wallet hub, 775-789

debugging
  Lock screen, engaging while debugging via Simulation Dashboard, 63
  scheduled tasks, 1025

Deep Zoom technology, viewing high-resolution images via MultiScaleImage element, 207-217

DelegateCommand constructor (ICommand interface), 38-39
deleting
  changes to stored contacts, 477-478
  music to media library, 205-207

deploying apps via Application Deployment tool, 15

DeviceExtendedProperties class. See DeviceStatus class

DeviceFirmwareVersion property (DeviceStatus class), 28

DeviceHardwareVersion property (DeviceStatus class), 28

DeviceManufacturer property (DeviceStatus class), 28

DeviceName property (DeviceStatus class), 28

DeviceStatus class, 27

ApplicationCurrentMemoryUsage property, 28

ApplicationMemoryUsageLimit property, 27-29

ApplicationPeakMemoryUsage property, 28

DeviceFirmwareVersion property, 28

DeviceHardwareVersion property, 28

DeviceManufacturer property, 28

DeviceName property, 28

DeviceTotalMemory property, 28

IsKeyboardDeployed property, 28

IsKeyboardPresent property, 28

KeyboardDeployedChanged event, 29-30

PowerSource property, 28, 30
PowerSourceChanged event, 29-30
properties, viewing, 27
DeviceTotalMemory property (DeviceStatus class), 28
dictation grammars (speech recognition), 701-703
directions
driving directions, retrieving, 409-412
Maps app, showing in, 611
disabling
application bar menu, 242-243
icon buttons (application bar), 235
double tap gestures (touch input), 370, 374, 382
drag gestures (touch input), 375-377, 383
drawing surface, presenting via InkPresenter element, 186-193
driving directions, retrieving, 409-412

E

eBay OData consumer application, building, 862
  EbaySearchView page, 867-871
  EbaySearchViewModel class, 864-867
  extending OData entity classes, 874
  fetching data when user scrolls to the end of a list, 871-873
  OData wrapper creation, 862-864
  ScrollViewerMonitor class, 871-873
edge tracing photo extras application, creating, 634-645
e-mail
  composing, 417-418
  contact addresses, saving, 418-421
  email addresses, selecting, 414-417
EmailAddressChooserTask, choosers/launchers and, 414-417
EmailComposeTask, choosers/launchers and, 417-418
embedding fonts (text), 162-164
emulators. See Windows Phone Emulator
events (life cycle of apps)
deactivating events, 52-53
launching, 50-51
subscribing to, 51
tombstoning, 52-53
Execute (object) method (iCommand interface), 38
execution model, 48
  application life cycle, 49
    deactivating events, 52-53
    launching events, 50-51
    subscribing to events, 51
    tombstoning, 52-53
  application state, 49
    restoring transient state, 55
    saving persistent state, 56
    saving transient state, 53-54
    transient state requirements, 54-55
apps
  exiting programmatically, 53
  terminating, 53
execution profiling (Windows Phone Application Analysis tool), 21-23
exiting apps programmatically, 53
expanding application bar, 233
Expression Blend for Windows Phone, 2
Expressions Editor (UTF), hiding, 752-753
External Events view (Windows Phone Application Analysis tool), 25

F
FAS (Fast App Switching), 56
Fast App Resume, 56, 63
  App.RootFrame_Navigated method, 60-61
  App.RootFrame_Navigating method, 59-61
enabling, 56-57
FastAppResumeViewModel class, 61-62
launching, 57-59
optimizing, 57
recommended resume behaviors, 58
file associations, 967-968
auto-launching apps via file associations, 968
launching files, 973-975
receiving file launch requests, 970-973
registering file associations, 968-970
user experience, effects on, 976
file transfers (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
financial information, storing in Wallet hub, 773
introduction to, 774-775
payment instruments, storing in, 775-777
Payment Instruments sample app, 777-789
updating payment instruments, 789-792
required capabilities for wallet integration, 775
Fit to Screen button (Windows Phone Emulator), 5
flat lists, 305-306
flexible layouts via WrapPanel component (Toolkit), 297
child element spacing, 298
ListBox, using with, 300-302
sample code, 298-299
flick gestures (touch input), 377-378, 384-386
flip tiles, 390, 394-395
floating menu (Windows Phone Emulator), 4
fonts (text)
assigning, 164
bold text, 159-160
built-in fonts, 160-162
colored fonts, 160
embedding fonts, 162-164
language support, 160-161
OpenType fonts, 159
properties of, 158-160
Segoe font, 160
selecting, 159
sizing, 159-160
stretching, 159
styling, 159
third-party fonts, 162-164
weight adjustments, 159-160
Windows Font Preview tool, 163
foreground apps
background audio playback, controlling, 1070-1075
background file transfers, 1043
multitasking and, 997
Mutex and shared resources, 1038-1041
foreign languages/cultures. See internationalization
Frame Rate Counter, 20-21
Frame Rate Graph (Windows Phone Application Analysis tool), 25
FrameworkElement class, 116
full-screen, switching application bar to, 244-246
full-screen picker pages, customizing, 282-286
G

game loops, controlling via GameTimer class, 202-203
  update intervals, controlling, 203
  XNA environment initialization, 203-204
GameInviteTask, choosers/launchers and, 453
  games, inviting players to, 453
GC Events (Windows Phone Application Analysis tool), 26
geographic location API, 563
  DesiredAccuracy property, 565
  DesiredAccuracyInMeters property, 565
  Geolocator class, 563
    location simulator sample code, 571-575
      testing, 570-571
  monitoring position changes, 566
  MovementThreshold property, 565-566
  PositionChanged events, 566-584
  retrieving current location, 563-565
  StatusChanged events, 567-568
geographically aware apps. See location aware apps
gesture events (touch input), 357-358, 368-369
  double tap gestures, 370
  GestureBegin events, 380-381
  GestureCompleted events, 380-381
  hold gestures, 371-372
  tap gestures, 369-370
globalization, 613-614
graphics/animation
  page orientation
    animating entire pages when orientation changes, 108-110
    animating page elements when orientation changes, 106-107
  page transitions, 110-114
  XAML
    composition thread, 18-20

H

headers
  control headers, adding to TimePicker component (Toolkit), 281
  Pivot headers, 331-332
heading, adjusting in Map control, 593
helium voice app, creating, 692-698
hiding
  application bar, 233
  Expressions Editor (UTF), 752-753
high-resolution images, viewing via MultiScaleImage element, 207-208
  Deep Zoom technology, 207-217
  Deep Zoom Viewer sample code, 212-217
  image creation, 208-209
  image tiling, 208
  MultiScaleImage API, 210-212
hold gestures (touch input), 371-375, 375, 382
HTTP services, 850
Hyperlink button, 124

I

iCommand interface, 38
  CanExecute (object) method, 38
  CanExecuteChanged method, 38
  Commanding, Event property, 39

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

1099

Integration testing, 739

1099

IntelliSense support, 168

Internationalization, 613
defining, 613
globalization, 613-614
localizability
defining, 614
dynamic localizability, 618-620
images, 620-621, 625-629
resx files, 614-618, 620-630
RTL support, 629-630
sample code, 621-630
text, 625-629
UI updates when cultures change, 618-620
speech-driven apps, 721-723
text support, 160-161

Internet
links, sending to contacts, 444
web pages, navigating, 446-447
web searches, 442-443

Invitations (games), 453

IoC (Inversion of Control), 765-767

IsKeyboardDeployed property (DeviceStatus class), 28

IsKeyboardPresent property (DeviceStatus class), 28

Isolated storage
IsolatedStorageSettings class, 889-890
local databases, deploying to, 943-946
SQL CE database files, 940-942
storing/browsing web content in, 225-228

Windows Phone SDK Isolated Storage Explorer, 938-939

WP7 Isolated Storage Explorer, 939-940

Item visibility, custom application bar, 250

How can we make this index more useful? Email us at indexes@samspublishing.com
items controls

ComboBox control, 142
ListBox control, 140-142
Selector class members, 139
Items layer (Panorama control), 349
itineraries, displaying via visual states in Map control, 604-610

J - K - L

JavaScript, web page behaviors, 222-223

KeyboardDeployedChanged event (DeviceStatus class), 29-30
keyboards, SIP layouts, 165-166

languages (internationalization), 613
globalization, 613-614
internationalization, 613
localizability
defining, 614
dynamic localizability, 618-620
images, 620-621, 625-629
resx files, 614-618, 620-630
RTL support, 629-630
sample code, 621-630
text, 625-629
UI updates when cultures change, 618-620
speech-driven apps, 721-723
text support, 160-161
launchers, 405, 408-409, 484
AddressChooserTask, 447-449
API overview, 405-406
appointments, 464
retrieving, 478-483
saving, 483-484
automated testing, 769-771
BingMapsDirectionTask, 409-412
BingMapsTask, 413
CameraCaptureTask, 450-452
connection setting pages, navigating to, 413-414
ConnectionSettingsTask, 413-414
contacts, 464
converting stored contacts to/from vCards, 478
deleting changes to stored contacts, 477-478
retrieving, 464-469
saving to contact lists, 449-450
selecting addresses, 447-449
storing in Custom Contacts Store, 469-477
driving directions, retrieving, 409-412
driving directions, retrieving, 409-412
e-mail
 composing, 417-418
 saving contact addresses, 418-421
 selecting email addresses, 414-417
EmailAddressChooserTask, 414-417
EmailComposeTask, 417-418
games, inviting players to, 453
images, sharing, 460-461
links, sending to contacts, 444
map locations, displaying, 413
Marketplace
 allowing app user reviews, 426
launching apps from, 425-426
navigating to apps in, 422-425
searching, 427-428
MarketplaceDetailTask, 422-425
MarketplaceHubTask, 425-426
MarketplaceReviewTask, 426
MarketplaceSearchTask, 427-428
media files, playing, 428-435
MediaPlayerLauncher, 428-435
phone calls, placing, 435-436
live tiles

phone numbers
    saving, 439-442
    selecting, 437-438
PhoneCallTask, 435-436
PhoneNumberChooserTask, 437-438
Photo Hub, selecting photos from, 454-459
PhotoChooserTask, 454-459
photos, taking, 450-452
ringtones, creating, 461-463
SaveContactTask, 449-450
SaveEmailAddressTask, 418-421
SavePhoneNumberTask, 439-442
SaveRingtoneTask, 461-463
SearchTask, 442-443
ShareLinkTask, 444
ShareMediaTask, 460-461
ShareStatusTask, 445
SMS, composing, 445
SmsComposeTask, 445
status updates, posting to social networks, 445
    web pages, navigating, 446-447
    web searches, 442-443
WebBrowserTask, 446-447
launching events (life cycle of apps), 50-51
layouts (flexible) via WrapPanel component (Toolkit)
    child element spacing, 298
    flexible layouts via WrapPanel component (Toolkit), 297
ListBox, using with, 300-302
    sample code, 298-299
lenses (cameras), 686-687
    app registration, 687-689
    icons, creating, 689
lists
    flat lists, 305-306
    grouped lists, 307
    Marketplace app list sample, 307-314
    user contacts list sample, 316-323
    LongListSelector, 303-304
live tiles, 389
    animation, 391
    changing templates, 391
    configuring, 390
    cycle tiles, 390-394
    flip tiles, 390, 394-395
    iconic tiles, 390, 395-396
    images, 390
    large tiles, 390
    lock screen customization, 397-399
        background images, 399-401
        creating lock screen icons, 403-404
        notification text, 401-403
LINQ
    maps and, 597
    OData and, 855-856
LINQ to SQL
    concurrency, 962-965
    inheritance hierarchies, mapping, 958-962
    local databases, data storage and, 905, 908
        code-first data model creation, 909-910
        platform differences, 909
    queries, viewing via custom log, 948-950
list grammars (speech recognition), 701-710, 731-732
ListBox control, 140-142
ListPicker component (Toolkit), 255-258
How can we make this index more useful? Email us at indexes@samspublishing.com
resolution, 391
schedules, updating tile notifications, 504-505
sizing, 390-391
sound, 391
Start Experience (Windows Phone Emulator), 389-390
updating, 391, 396-397
local databases, data storage and, 905
database schemas
  updating, 951-957
  versioning, 957-958
  viewing, 937-942
deploying databases, 906-907
isolated storage, deploying databases to, 943-946
LINQ to SQL, 905, 908
code-first data model creation, 909-910
  concurrency, 962-965
  mapping inheritance hierarchies, 958-962
  platform differences, 909
  viewing queries via custom log, 948-950
navigation services, abstracting, 946-948
SQL Server Compact, 906
SqlMetal and database-first data model creation, 942-943
Twitter timeline viewer example, 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
XAP files, 906-907
local folder, state preservation and, 881-882
  .NET Isolated Storage API, 883-886
  WinPRT Windows Storage API, 883, 887-889
localizability
defining, 614
dynamic localizability, 618-620
  images, 620-621, 625-629
  resx files, 614-618
  image localization, 620-621
  localizability sample code, 621-630
  RTL support, 629-630
  sample code, 621-630
  text, 625-629
  UI updates when cultures change, 618-620
location aware apps. See also maps, 559
A-GPS and, 560-561
architecture of, 562-563
background location tracking, 569-570
cell tower triangulation and, 561-562
determining location, 559-562
geographic location API, 563
  DesiredAccuracy property, 565
  DesiredAccuracyInMeters property, 565
  Geolocator class, 563, 570-575
  monitoring position changes, 566
  MovementThreshold property, 565-566
  PositionChanged events, 566-584
  retrieving current location, 563-565
  StatusChanged events, 567-568
  hardware layer, 563
  location simulator, 570-575
  location viewer sample code, 575
civic address resolution, 580
displaying location via GeoLocationView page, 577-579
GeoLocationViewModel class, 575-577
managed layer, 563
native code layer, 563
Rx and, 580-584
Wi-Fi triangulation and, 562
location tracking via Map control, 594-596
lock screen
customizing via live tiles, 397-399
background images, 399-401
creating lock screen icons, 403-404
notification text, 401-403
running apps under, 63-64
LockScreenService class, 64-66
LockScreenViewModel class, 66-67
Windows Phone Emulator, engaging Lock screen within, 63
LockablePivot control (Toolkit), 355
LongListSelector, 303-304
flat lists, 305-306
grouped lists, 307
Marketplace app list sample, 307-314
user contacts list sample, 316-323
visual structure of, 304
looping lists, 287-291
LoopingSelector component (Toolkit), 287-291
low light conditions, adjusting Map control for, 588-589
loyalty cards, Wallet hub and, 773
introduction to, 774-775
membership information
storing, 792-796
updating, 802-804
required capabilities for wallet integration, 775
maps. See also location aware apps, 559
LINQ and, 597
Map control, 585-587
Application ID, 587
AuthenticationToken, 587
calculating shortest routes between two points, 598-601
cartographic modes, 589-590
centering to current location, 590-591
customizing page elements via visual states, 604-610
heading adjustments, 593
location tracking, 594-596
low light conditions, 588-589
overlaying content, 596-597
panning/zooming, 590
pitch adjustments, 593
route calculator, 601-602
sample Map View, 587-588
searching for routes via RouteSearchView, 602-604
setting viewable area of map, 592-593
showing directions via Maps app, 611
showing locations via Maps app, 610-611
Maps app
downloading maps, 612
showing directions via, 611
showing locations via, 413, 610-611
Marketplace apps
allowing user reviews, 426
launching, 425-426
Marketplace

navigating to, 422-425
searching, 427-428
Marketplace Test Kit, determining app capabilities via, 17
MarketplaceDetailTask, choosers/launchers and, 422-425
MarketplaceHubTask, choosers/launchers and, 425-426
MarketplaceReviewTask, choosers/launchers and, 426
MarketplaceSearchTask, choosers/launchers and, 427-428
media files, playing, 428-435
media library, saving/deleting music, 205-207
MediaElement, playing audio/video, 193-194
controlling audio output, 194
sample code, 194-200
streaming content, 194
XNA SoundEffect class, 201-202
MediaPlayerLauncher, choosers/launchers and, 428-435
memberships (club), Wallet hub and, 773
introduction to, 774-775
membership information
storing, 792-796
updating, 802-804
required capabilities for wallet integration, 775
memory usage
determining (apps), 27-29
Memory Usage MB Graph (Windows Phone Application Analysis tool), 26
menu items (application bar)
built-in application bar, customizing, 234-235
custom application bar, 237-238
runtime, retrieving at, 236-237
MessageService property and IMessageService interface, 41-44
metadata, UTF (Unit Test Framework) and, 745-751
Metro. See Microsoft Design Language
microphone, 691
helium voice app, creating, 692-698
recording audio, 691-692
Microsoft Design Language, 153
Microsoft Expression Blend for Windows Phone, 2
Microsoft.Phone.Info.DeviceStatus class. See DeviceStatus class
minimizing
Icon Button Tray (application bar), 243
Windows Phone Emulator window, 4-5
motion sensors, 525, 552-557
mouse events (touch input), 357-359, 361-362
MPNS (Microsoft Push Notification Service), 487-488, 509-511
MSIL (Microsoft Intermediate Language), XAP files, 16-17
multiline text, 155-158
multiple languages/cultures (internationalization), 613
globalization, 613-614
internationalization, 613
localizability
defining, 614
dynamic localizability, 618-620
images, 620-621, 625-629
resx files, 614-618, 620-630
RTL support, 629-630
sample code, 621-630
text, 625-629
UI updates when cultures change, 618-620
speech-driven apps, 721-723
text support, 160-161
MultiScaleImage API, 210-212
MultiScaleImage element, viewing high-resolution images, 207-208
Deep Zoom technology, 207-217
  Deep Zoom Viewer sample code, 212-217
  image creation, 208-209
image tiling, 208
  MultiScaleImage API, 210-212
multitasking, background processing and, 997
music, saving to/deleting from media library, 205-207
Mutex, shared resources and, 1038-1041
MVVM (Model-View-ViewModel) pattern, 31
  AutoCompleteBox component (Toolkit) and, 260-264
  ContextMenu component (Toolkit) and, 273-275
  implementing, 31-32
  property change notification, 33
    INPC implementation, alternative approach, 35-37
    INPC implementation, traditional approach, 33-35
  ViewModelBase class, 32
ViewModelBase class, 32

N
navigating pages, 67-69
  canceling, 77
  cross-page communication, 77-78
  handling, 76
  hardware Back button, 79-80
  navigation history stacks, 72-73
NavigationService property, 75
  backward navigation, 76
  forward navigation, 76
  page redirection, 78-79
query strings, passing page arguments via, 71-73
unmapped URI, 69
  external navigation via button control, 69-70

external navigation via HyperlinkButton control, 70
hosting web content within apps, 70-71
internal URI, 69
URI mapping, 73-74
navigation history stacks, 72-73
.NET Isolated Storage API, 883-885
  reading/writing data to isolated storage, 884-886
  serialization performance, 886-887
network connectivity, monitoring, 850-855
network services, 849
  Data Sense, monitoring data usage via, 877-879
  hosting, 850
  HTTP services, 850
  network connectivity, monitoring, 850-855
OData, 850, 855
  eBay consumer application build, 862-874
  EbaySearchView page, 867-871
  EbaySearchViewModel class, 864-867
  extending OData entity classes, 874
  fetching data when user scrolls to the end of a list, 871-873
  generating proxies, 858
  LINQ and, 855-856
  query options, 858-860
  ScrollViewerMonitor class, 871-873
  URI structure, 856-858
  using proxies, 860-861
  wrapper creation, 862-864
REST services, 850
Simulation Dashboard, 876
SOAP services, 850
NFC, sharing images, 460-461
notification text, lock screen customization, 401-403
notifications (push), 485
  benefits of, 486-487

How can we make this index more useful? Email us at indexes@samspublishing.com
cloud service authentication, 511
enabling, 489
MPNS and, 487-488, 509-511
notification classes, 509-511
power management and, 493-494
raw notifications, 486, 505
notification classes, 510-511
notifications in HttpWebResponse, identifying, 509
receiving, 508-509
sending, 505-507
stock ticker application example, 521-523
sending, 495
SLA and, 488
stock ticker application example, 511-524
subscribing to, 489-490
binding to shell, 490-491
channel errors, handling, 492-493
HttpNotificationChannel events, 491-492
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
notifications (scheduled), 998-1001
alarm registration, 1001
alarm sample, 1002-1004
reminder registration, 1005-1006
reminder sample, 1006-1010
numbers (phone)
saving, 439-442
selecting, 437-438
O
OData, 850, 855
eBay consumer application, building, 862
EbaySearchView page, 867-871
EbaySearchViewModel class, 864-867
extending OData entity classes, 874
fetching data when user scrolls to the end of a list, 871-873
OData wrapper creation, 862-864
ScrollViewerMonitor class, 871-873
LINQ and, 855-856
proxies
generating, 858
using, 860-861
query options, 858-860
URI structure, 856-858
opacity
application bar, changing in
built-in application bar, 234
custom application bar, 244
XAML element visibility and performance, 20
OpenType fonts, 159
orientation
page orientation
animating entire pages when orientation changes, 108-110
animating page elements when orientation changes, 106-107
PhoneApplicationPage class, 99-104
setting at runtime, 105-106
PanoramaItem control, 350
Windows Phone Emulator display, changing, 4-5
page navigation. See also Pivot control; Panorama control, 67-69, 325
canceling, 77
cross-page communication, 77-78
handling, 76
hardware Back button, 79-80
navigation history stacks, 72-73
NavigationService property, 75
backward navigation, 76
forward navigation, 76
page redirection, 78-79
query strings, passing page arguments via, 71-73
unmapped URI, 69
external navigation via button control, 69-70
external navigation via HyperlinkButton control, 70
hosting web content within apps, 70-71
internal URI, 69
URI mapping, 73-74
page orientation
animating
entire pages when orientation changes, 108-110
page elements when orientation changes, 106-107
PhoneApplicationPage class, 99-100
OrientationChanged event, 100-102
PhoneApplicationPage orientation property, 102-104
runtime, setting at, 105-106
page transitions (animated), Windows Phone Toolkit, 110-114
panning/zooming via Map control, 590
Panorama control. See also Pivot control; page navigation, 325, 327-328, 347
Background layer, 348-349
components of, 347-348
Items layer, 349
PanoramaItem control, 349-350
Pivot control, differences and similarities, 326
sample code, 350-354
style assumptions, 326
things to avoid, 355
Title layer, 349
Windows Phone 8 performance, 329
Windows Phone FCL, placement within, 328
PasswordBox element, 154
passwords, receiving via PasswordBox control, 175-176
payments, making via Wallet hub, 773
introduction to, 774-775
payment instruments, storing in, 775-777
Payment Instruments sample app, 777-789
updating payment instruments, 789-792
required capabilities for wallet integration, 775
performance
apps, Windows Phone Marketplace requirements, 21
Frame Rate Counter, 20-21
Windows Phone 8, Pivot and Panorama controls, 329
Windows Phone Application Analysis tool, 21, 26
CPU Usage Graph, 25
execution profiling, 21-23
External Events view, 25
Frame Rate Graph, 25
GC Events, 26
Image Loads, 26
launching, 21-23
Memory Usage MB Graph, 26
Performance Warnings view, 26-27
Storyboards, 26
viewing performance metrics, 24

XAML element visibility and performance, 19-20

Performance Warnings view (Windows Phone Application Analysis tool), 26-27

periodic tasks, 1013-1014

persistent application state. See also transient application state, 49, 56, 97

phone calls, placing, 435-436

phone numbers
   saving, 439-442
   selecting, 437-438

PhoneApplicationPage class, page orientation, 99-100
   OrientationChanged event, 100-102
   PhoneApplicationPage orientation property, 102-104

PhoneApplicationPage.State dictionary, state preservation and, 49

PhoneCallTask, choosers/launchers and, 435-436

PhoneNumberChooserTask, choosers/launchers and, 437-438

Photo Hub, selecting photos from, 454-459

PhotoChooserTask, choosers/launchers and, 454-459

photos
   cameras, 655
      lenses, 686-689
   PhotoCamera class, 655-656, 678
   Silverlight webcam API, 655-656, 678-685

picture viewer, 631
   edge tracing extras application, 634-645
   photo extras applications, 631-645
   photo share applications, 631, 645-654
   photo upload share application, 647-654
   pixel density/picture quality, 632
   sharing, 460-461
   taking, 450-452

thumbnail photos, viewing in PhotoCamera app, 672-678

phrase list grammars (speech recognition), 701-710, 731-732

picker pages, customizing, 282-286

picture viewer, 631
   photo extras applications, 631
      creating, 632-645
      edge tracing extras application, 634-645
   photo share applications, 631
      creating, 645-654
   photo upload share application, 647-654

pinch gestures (touch input), 378-380, 383-384

pitch, adjusting in Map control, 593

Pivot control. See also Panorama control; page navigation, 325, 327, 329
   application bars, hosting multiple, 333-343
   components of, 331-332
   load events, 333
   LockablePivot control (Toolkit), 355
   Panorama control, differences and similarities, 326
   placing Pivot on a page, 329-331
   populating via data bound collections, 344-347
   setting active PivotItem presenters, 332
   style assumptions, 326
   things to avoid, 355
   Visual Studio New Project dialog, creating Pivot applications in, 331
   Windows Phone 8 performance, 329
   Windows Phone FCL, placement within, 328

pixel density/picture quality (picture viewer), 632

playCommand constructor (iCommand interface), 38-39

position changes, monitoring (location aware apps), 566-584
power management, push notifications and, 493-494
PowerSource property (DeviceStatus class), 28, 30
PowerSourceChanged event (DeviceStatus class), 29-30
preserving app state/settings, 47-48
  automatic state preservation system, building, 890-893
    binary serialization, 898-901
    creating property accessor delegates, 902-903
    customizing ViewModel state preservation, 892-893
    identifying stateful ViewModels, 893-898
    Silverlight Serializer and, 898-901
    unwinding property lambda expressions, 901-902
IsolatedStorageSettings class, 889-890
local folder, 881-882
managed storage, 882
.NET Isolated Storage API, 883-885
  reading/writing data to isolated storage, 884-886
  serialization performance, 886-887
WinPRT Windows Storage API, 883, 887-888
  application settings, 888-889
  measuring available free space, 888
  serialization requirements, 889
previewing fonts (text), 163
product lists, displaying (Bookshop sample app), 86-92
ProductsView class, Bookstore sample app, 85-86
ProductsViewModel class, Bookshop sample app, 84-85
progress indicators, 145-151
ProgressBar control, 144
property change notification (MVVM), 33-37
protocol associations, 967-968
auto-launching apps via protocol associations, 977
  launching protocol URI, 979-980
  receiving protocol launch requests, 977-979
  registering protocol associations, 977
  reserved protocol names, 981
user experience, effects on, 976
push notifications, 485
  benefits of, 486-487
  cloud service authentication, 511
  enabling, 489
  MPNS and, 487-488, 509-511
  notification classes, 509-511
  power management and, 493-494
  raw notifications, 486, 505
    notification classes, 510-511
    notifications in HttpWebResponse, identifying, 509
    receiving, 508-509
    sending, 505-507
  stock ticker application example, 521-523
  sending, 495
SLA and, 488
stock ticker application example, 511-524
subscribing to, 489-490
  binding to shell, 490-491
  channel errors, handling, 492-493
  HttpNotificationChannel events, 491-492
push notifications

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

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 class
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 in, 5
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
saving appointments, 483-484
contacts
custom contact lists, 449-450
custom contacts store, 469-477
e-mail addresses, 418-421
music to media library, 205
phone numbers, 439-442
transient state, 53-54
scaling, fit to screen button (windows phone emulator display window), 5
scheduled notifications, 998-1001
alarm registration, 1001
alarm sample, 1002-1004
reminder registration, 1005-1006
reminder sample, 1006-1010
scheduled tasks, 1010-1011, 1014-1018
api limitations, 1036-1038
background agents, 1011-1014
debugging, 1025
periodic tasks, 1013-1014
registering, 1018-1019
resource intensive tasks, 1014
scheduledtaskagent, 1011-1012
to-do list example, 1019-1024
creating to-do item shell tiles, 1028-1029
debugging scheduled tasks, 1025
editing existing to-do items, 1030-1035
saving to-do items, 1029-1030
todolistview xaml, 1025-1028
updating tiles via scheduled task agent, 1035-1036
screen fill rate counter field (frame rate counter), 22
scrollbars, 152
search integration, 983
app instant answer, 996
configuring, 985
application manifests, 985-987
creating landing pages to display quick card information, 994-995
customizing captions, 987-989
defining quick card target page, 989-994
searching
marketplace, 427-428
web, 442-443
search task, choosers/launchers and, 442-443
segoe font (text), 159
selector class, items controls, 139
semantic validation, 806
sensors
accelerometers, 525
accelerometer class, 528-529
calibrating, 533-536
shake detection, 537-539
simulating acceleration via emulator, 529-530
smoothing readings, 530-533
compass sensors, 525, 539-545
calibrating, 546-548
compass orientation, 545-546
gyroscope sensors, 525, 548-552
motion sensors, 525, 552-557
overview of, 525-527
shake detection, accelerometers and, 537-539
sharelinktask, choosers/launchers and, 444
ShareMediaTask, choosers/launchers and,
ShareStatusTask, choosers/launchers and, 445

sharing
- images, 460-461
- links with contacts, 444
- status updates in social networks, 445

shell tiles. See live tiles

Show Resolution Settings button (Windows Phone Emulator), 5

Silverlight
- Deep Zoom technology, 207-217
- Silverlight Serializer, building an automatic state preservation system, 898-901
- Silverlight webcam API, 655-656, 678-685
- Windows Phone FCL
  - Silverlight controls not in FCL, 118-119
  - unsupported Silverlight controls, 119

XAML
- 1
- XAML plus .NET, 1

Simulation Dashboard, 63, 876

SIP (Software Input Panel), 165
- dimensions of, 165
- dismissing programmatically, 167-168
- keyboard layouts, 165-166
- opening programmatically, 165-167

sizing
- buttons, 121-123
- images, 185-186
- live tiles, 390-391
- text, 159-160
- touch input sizing/spacing constraints, 387
- Windows Phone Emulator displays, 4

SLA (Service-Level Agreements), push notifications and, 488

slider controls, 151-152

SMS, composing, 445

SmsComposeTask, choosers/launchers and, 445

SOAP services, 850

social networks, posting status updates to, 445

Solution Explorer, 3

sound
- background audio playback, 1059
  - AudioPlayerAgent class, 1060, 1064-1075
  - AudioStreamingAgent class, 1060
  - AudioTrack class, 1061
  - BackgroundAudioPlayer class, 1060-1061
- controlling playback from foreground app, 1070-1075
- creating custom audio player agent, 1061-1064
- overview of, 1060
- playing assembly resources via MediaStreamSource, 1078-1079

live tiles, 391

media files, playing, 428-435

ringtones, creating, 461-463

sound effects, 201-202

special offers/deals, Wallet hub and, 796-802

speech-driven apps, 699

speech recognition, 700-701
  - custom speech recognition UI, 718-721
  - dictation grammars, 701-703
  - internationalization, 721-723
  - 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

TTS, 732

selecting speaking voice, 732-733

SSML files, 733-735
voice commands, launching apps via, 723-724
installing VCD files, 726-730
updating phrase lists in VCD files, 731-732
VCD file structure, 724-725
Voice Paint app, creating, 703-721
splash screens, 80-83
SQL CE database files, viewing/modifying, 940-942
SQL Server Compact, local databases and data storage, 906
SqlMetal, database-first data model creation, 942-943
SRGS grammars (speech recognition), 702, 710-717
SSML (Speech Synthesis Markup Language) files, 733-735
Start Experience (Windows Phone Emulator), 6, 389-390
state preservation, 47-48
automatic state preservation system, building, 890-893
binary serialization, 898-901
creating property accessor delegates, 902-903
customizing ViewModel state preservation, 892-893
identifying stateful ViewModels, 893-898
Silverlight Serializer and, 898-901
unwinding property lambda expressions, 901-902
IsolatedStorageSettings class, 889-890
local folder, 881-882
.NET Isolated Storage API, 883-886
WinPRT Windows Storage API, 883, 887-889
managed storage, 882
.NET Isolated Storage API, 883-885
reading/writing data to isolated storage, 884-886
serialization performance, 886-887
WinPRT Windows Storage API, 883, 887-888
application settings, 888-889
measuring available free space, 888
serialization requirements, 889
status updates, posting to social networks, 445
stock ticker application example, push notifications and, 511-524
storage
isolated storage
deploying databases to, 943-946
IsolatedStorageSettings class, 889-890
.NET Isolated Storage API, 883-886
SQL CE database files, 940-942
storing/browsing web content in, 225-228
Windows Phone SDK Isolated Storage Explorer, 938-939
WP7 Isolated Storage Explorer, 939-940
local databases, data storage and, 905
code-first data model creation, 909-910
deploying databases, 906-907
deploying databases to isolated storage, 943-946
LINQ to SQL, 905, 908-909, 948-950, 958-965
mapping inheritance hierarchies, 958-962
navigation services, abstracting, 946-948
SQL Server Compact, 906
SqlMetal and database-first data model creation, 942-943
Twitter timeline viewer example, 910-937
updating database schemas, 951-957
versioning database schemas, 957-958
viewing database schemas, 937-942
viewing/modifying SQL CE database files, 940-942
Windows Phone SDK Isolated Storage Explorer, 938-939

How can we make this index more useful? Email us at indexes@samspublishing.com
WP7 Isolated Storage Explorer, 939-940
XAP files, 906-907
managed storage, state preservation and, 882
WinPRT Windows Storage API, 883
Storyboards (Windows Phone Application Analysis tool), 26
streaming audio/video content, 194
stretching text, 159
styling text, 159
submitting apps to Windows Phone Marketplace, 15-16
subscribing to
  events (life cycle of apps), 51
  push notifications, 489-490
    binding to shell, 490-491
    channel errors, handling, 492-493
    HttpNotificationChannel events, 491-492
Surface Counter field (Frame Rate Counter), 22
switch color, changing via ToggleSwitch component (Toolkit), 297
synchronous validation
  decoupling validation, 830
  provisioning for, 827-828
  validating changing properties, 830-833
syntactic validation, 806

T

tag expressions in UTF (Unit Test Framework), 743-744
tap and click events (buttons), 123
tap gestures (touch input), 369-370, 373-374, 382
templates (Windows Phone App), 3
terminating apps, 53
testing (automated), 738-739
  ad hoc testing, 737
  choosers, testing via, 769-771
integrated testing, 739
IoC, 765-767
launchers, testing via, 769-771
trial conditions, testing, 767-769
unit testing, 737-738
UTF, 739-740
  AssemblyCleanup attribute, 746
  AssemblyInitialize attribute, 745-746
  assertions, 751-752
  Asynchronous attribute, 749
  asynchronous testing, 762-763
  Bug attribute, 749-750
  ClassCleanup attribute, 746
  ClassInitialize attribute, 746
  creating a test project, 740-741, 747
  creating test classes, 741-743
  Description attribute, 748
  ExpectedException attribute, 749
  hiding Expressions Editor, 752-753
  Ignore attribute, 747-748
  metadata, 745-751
  Owner attribute, 748-749
  Priority attribute, 750
  Tag attribute, 749
  tag expressions, 743-744
  testable chat client app example, 754-764
  TestClass attribute, 744
  testing multiple assemblies, 753
  testing nonpublic members, 753
  TestInitialize attribute, 747
  TestMethod attribute, 745
  TestProperty attribute, 747
  Timeout attribute, 748
  WorkItemTest class, 751

text
  clipboard, placing text on, 179-180
displaying, 155
element types, understanding, 154-155

fonts
  assigning, 164
  bold text, 159-160
  built-in fonts, 160-162
  colored fonts, 160
  embedding fonts, 162-164
  language support, 160-161
  OpenType fonts, 159
  properties of, 158-160
  Segoe font, 159
  selecting, 159
  sizing, 159-160
  stretching, 159
  styling, 159
  third-party fonts, 162-164
  weight adjustments, 159-160
  Windows Font Preview tool, 163

formatting at runtime, 176-179

inputting via TextBox control, 165
  input scope, 168-174
  IntelliSense support, 168
  SIP, 165-168
  word prediction, 168

line breaks, 155-157

localizability (internationalization), 625-629

Microsoft Design Language, 153

multiline text, 155-158

notification text, lock screen customization, 401-403

PasswordBox element, 154

passwords, receiving via PasswordBox control, 175-176

rich content, displaying via RichTextBox control, 176-179

suggesting via AutoCompleteBox component (Toolkit), 259-266
  dynamically populating, 266-267
  styling, 268-270

TextBlock element, 154-155
  displaying text, 155
  LineBreak objects, 155-158
  multiline text, 158
  Run objects, 155-158
  TextBox element, 154-155

TTS, 732
  selecting speaking voice, 732-733
  SSML files, 733-735
  whitespace, handling, 155

Texture Memory Usage field (Frame Rate Counter), 22

third-party fonts (text), 162-164

thumbnail images, viewing in PhotoCamera app, 672-678

tile notifications, 486, 500-501
  notification classes, 510-511
  sending, 501-504
  stock ticker application example, 520
  updating via shell tile schedules, 504-505

Tile options (Windows Phone Emulator), 6

tiling images, 208

TiltEffect component (Toolkit), 291-293

TimePicker/DatePicker components (Toolkit), 277-281
  control headers, adding to TimePicker component (Toolkit), 281
  customizing
    full-screen picker pages, 282-286
    value formats, 281-282

Title layer (Panorama control), 349

titles (apps), retitling via Windows Phone Emulator, 6

toast notifications, 485, 495-496
  notification classes, 510-511
  receiving, 496-497
  sending, 497-500
  stock ticker application example, 520

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

How can we make this index more useful? Email us at indexes@samspublishing.com
update intervals (XNA game loops), controlling

updating

deals/special offers, Wallet hub and,
796-802
live tiles, 391, 396-397
local databases schemas, 951-957
localizability (internationalization), 618-620
membership information, Wallet hub and,
802-804
payment instruments in Wallet hub,
789-792
phrase lists in VCD files, 731-732
status updates, posting to social networks,
445
stock ticker application example, 523-524
tile notifications via shell tile schedules,
504-505
to-do item tiles via Scheduled Task Agent,
1035-1036
Toolkit, 254
upload share application (photos), creating,
647-654
user input (controls)

buttons

Button Click mode, 124
check boxes, 131-138
Hyperlink button, 124
icon buttons (application bar), 234-237
icon buttons (custom application bar),
237-238
minimizing Icon Button Tray (application
bar), 234, 243
Radio button, 128-131
Repeat button, 124-126
sizing, 121-123
tap and click events, 123
Toggle button, 124, 126-128
content controls, 119-121
drawing surface, presenting via InkPresenter
element, 186-193
items controls

ComboBox control, 142
ListBox control, 140-142
Selector class members, 139
range controls

progress indicators, 145-151
ProgressBar control, 144
RangeBase class properties, 143-144
scrollbars, 152
sliders, 151-152
sensors

accelerometers, 525, 527-539
compass sensors, 525, 539-548
gyroscope sensors, 525, 548-552
motion sensors, 525, 552-557
overview of, 525-527
ToolTip control, 138
touch feedback, providing via Toolkit,
291-293
touch input

designing for, 386-388
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, 372-386
TouchPoint class, 357, 360-363
UIElement gesture events, 357-358,
368-372
Windows Phone Emulator, 358
validating, 805

asynchronous validation, 824-847
composite validation, 824-847
defining validation visual states, 810-824
property setters, 806-810
semantic validation, 806
syntactic validation, 806
Windows Phone FCL
control types within, 117
Silverlight controls not in FCL, 118-119
unsupported Silverlight controls, 119
User Interface Thread Frame Rate (FPS) field
(Frame Rate Counter), 22
user reviews (apps), allowing in Marketplace, 426
UTF (Unit Test Framework), 739-740
AssemblyCleanup attribute, 746
AssemblyInitialize attribute, 745-746
assertions, 751
  CollectionAssert, 752
  StringAssert, 752
  verifying collection conditions, 752
  verifying string conditions, 752
Asynchronous attribute, 749
asynchronous testing, 762-763
Bug attribute, 749-750
ClassCleanup attribute, 746
ClassInitialize attribute, 746
Description attribute, 748
ExpectedException attribute, 749
Expressions Editor, hiding, 752-753
Ignore attribute, 747-748
metadata, 745-751
Owner attribute, 748-749
Priority attribute, 750
Tag attribute, 749
tag expressions, 743-744
test classes, creating, 741-743
test project, creating, 740-741
testable chat client app example, 754-758
  automation peers, manipulating UI elements at runtime, 763-764
  building views, 758-760
  code driven UI testing, 760-763
TestClass attribute, 744
TestCleanup attribute, 747
testing
  multiple assemblies, 753
  nonpublic members, 753
TestInitialize attribute, 747
TestMethod attribute, 745
TestProperty attribute, 747
Timeout attribute, 748
WorkItemTest class, 751

validating
  arguments, 39-40
  group validation, 818-820, 844-847
  user input, 805
    asynchronous validation, 824-847
    composite validation, 824-847
    defining validation visual states, 810-824
    property setters, 806-810
    semantic validation, 806
    syntactic validation, 806
vCards, converting stored contacts to/from, 478
VCD (Voice Command Definition) files, 723-724
  installing, 726-730
  structure of, 724-725
  updating phrase lists in, 731-732
video
  media files, playing, 428-435
  PhotoCamera app, adding video effects to, 668-671
  playing via MediaElement, 193-194
    controlling audio output, 194
    sample code, 194-200

How can we make this index more useful? Email us at indexes@samspublishing.com
video

streaming content, 194
XNA SoundEffect class, 201-202
Silverlight webcam API, displaying video in, 684-685
ViewModelBase class (MVVM), 32

visibility
item visibility in custom application bar, 250
XAML element visibility and performance, 19-20
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
XAML design view, 4

Visual Studio New Project dialog, creating Pivot applications in, 331

voice-driven apps, 699

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
selecting speaking voice, 732-733
SSML files, 733-735
voice commands, launching apps via, 723-724

installing VCD files, 726-730
updating phrase lists in VCD files, 731-732
VCD file structure, 724-725
Voice Paint app, creating, 703-721

W

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
required capabilities for wallet integration, 775

WCF services
Bookshop sample app, 95-97
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, 218-221
injecting behavior into web pages, 222-223
monitoring/canceling navigation, 217-218
storing/browsing web content in isolated storage, 225-228

web pages, navigating, 446-447
web search grammars (speech recognition), 701-703
WebBrowserTask, choosers/launchers and, 446-447
webcam API (Silverlight), 655-656
weight (text), adjusting, 159-160
whitespace, handling in text, 155
Wi-Fi triangulation and location aware apps, 562
Windows Font Preview tool, 163
Windows Live Anonymous IDs, retrieving (background file transfers), 1050-1052
Windows Mobile 6.5 apps, Windows Phone 8 compatibility, 1
Windows Phone 8
  performance, Pivot and Panorama controls, 329
  Windows Mobile 6.5 app compatibility, 1
Windows Phone Application Analysis tool, 21, 26
  CPU Usage Graph, 25
  execution profiling, 21-23
  External Events view, 25
  Frame Rate Graph, 25
  GC Events, 26
  Image Loads, 26
  launching, 21-23
  Memory Usage MB Graph, 26
  performance metrics, viewing, 24
  Performance Warnings view, 26-27
  Storyboards, 26
Windows Phone Certification Requirements, launching events, 50
Windows Phone Emulator, 2
  Additional Tools window, opening, 4
  app customization
    MainPage, 6, 9-11
    retitling apps, 6
  App List, 6
  Application class, 7-8
  Close button, 5
  display orientation, changing, 4-5
  display size, changing, 4
  display window, scaling via Fit to Screen button, 5
  floating menu, 4
  launching, 3-4
  Lock screen, engaging, 63
  minimizing window, 4-5
  resolution settings, changing, 5
  Show Resolution Settings button, 5
  Start Experience, 6
  Tile options, 6
  touch input, 358
  WCF services, consuming, 44-45
  XAML apps
    customizing, 5-7
    debugging, 3-7
    running, 3-7
Windows Phone execution model. See execution model
Windows Phone FCL
  controls
    Button Click mode, 124
    check boxes, 131-138
    ComboBox control, 142
    content controls, 119-121
    control types within FCL, 117
    Hyperlink button, 124
    items controls, 138-142
    ListBox control, 140-142
    progress indicators, 145-151
    ProgressBar control, 144
    Radio button, 128-131
    range controls, 142-152
    Repeat button, 124-126
    scrollbars, 152
Silverlight controls not in FCL, 118-119
sizing buttons, 121-123
sliders, 151-152
tap and click events, 123
Toggle button, 124, 126-128
Tooltip control, 138
unsupported Silverlight controls, 119
Panorama control placement within, 328
Pivot control placement within, 328
Windows Phone Marketplace
Marketplace Test Kit, determining app capabilities via, 17
push notification requirements, 489
submitting apps to
  app performance, 21
  capability discovery phase, 16-17
  XAP files, 15-16
Windows Phone SDK 8.0
Frame Rate Counter, 20-21
installing, 2
lock screen/app integration, 397-399
  background images, 399-401
  creating lock screen icons, 403-404
  notification text, 401-403
Microsoft Expression Blend for Windows Phone, 2
requirements for, 2
Simulation Dashboard, 876
support for, 2
text
  formatting at runtime, 178-179
  inputting via TextBox control, 165-176
  placing on clipboard, 179-180
  rich content, displaying via RichTextBox control, 176-179
Visual Studio Express 2012, 2
  Add New Project dialog, 2
  creating XAML apps, 2-3
  Solution Explorer, 3
Windows Phone App template, 3
XAML design view, 4
Windows Phone Emulator, 2, 5
  App List, 6
  Application class, 7-8
  changing display orientation, 4-5
  changing display size, 4
  changing resolution settings, 5
  customizing XAML apps, 5-7
  debugging XAML apps, 3-7
  floating menu, 4
  launching, 3-4
  MainPage, 6, 9-11
  minimizing window, 4-5
  opening Additional Tools window, 4
  retitling apps, 6
  running XAML apps, 3-7
  scaling display window, 5
  Start Experience, 6
  Tile options, 6
Windows Phone SDK Isolated Storage Explorer, 938-939
Windows Phone Toolkit. See Toolkit
Windows.System.Launcher class, auto-launching apps via, 981
WinPRT Windows Storage API, 883, 887-888
  application settings, 888-889
  measuring available free space, 888
  serialization requirements, 889
WMAppManifest.xml files, defining app capabilities, 16
word prediction, inputting text, 168
WP7 Isolated Storage Explorer, 939-940
WrapPanel component (Toolkit), flexible layouts via, 297
  child element spacing, 298
  ListBox, using with, 300-302
  sample code, 298-299