Sams Teach Yourself
Windows® 8.1 Apps
with XAML and C#

in 24 Hours

Adam Nathan
Sams Teach Yourself Windows® 8.1 Apps with XAML and C# in 24 Hours
Copyright © 2014 by Pearson Education

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

ISBN-10: 0-672-33836-X
Library of Congress Control Number: 2014931708
Printed in the United States of America
First Printing June 2014

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

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

Special Sales
For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.
For government sales inquiries, please contact governmentsales@pearsoned.com.
For questions about sales outside the U.S., please contact international@pearsoned.com.
Contents at a Glance

Introduction .............................................................................................................. 1

Part I: Getting Started

   HOUR  1  Preparing for Development ................................................................. 5
   2  Discovering the XAML Language ................................................................. 23

Part II: Building an App

   3  Arranging UI Elements .................................................................................. 57
   4  Mastering Layout .............................................................................................. 79
   5  Handling Input: Touch, Mouse, Pen, and Keyboard ........................................ 113
   6  Managing the App Lifecycle ........................................................................... 159
   7  Threading, Windows, and Pages ..................................................................... 181

Part III: Understanding Controls

   8  Content Controls .............................................................................................. 201
   9  Items Controls ................................................................................................ 233
   10 Text ................................................................................................................. 259
   11 Images ............................................................................................................ 293
   12 Audio, Video, and Speech .............................................................................. 335
   13 Other Controls ................................................................................................ 371

Part IV: Leveraging the Richness of XAML

   14 Vector Graphics .............................................................................................. 411
   15 Animation ........................................................................................................ 447
   16 Styles, Templates, and Visual States ............................................................... 493
   17 Data Binding ................................................................................................... 525
Part V: Exploiting Windows 8.1

18 Working with Data ........................................... 551
19 Supporting Charms ........................................... 579
20 Leveraging Contracts ......................................... 611
21 Reading from Sensors ....................................... 647
22 Controlling Devices .......................................... 665
23 Live Tiles, Notifications, and the Lock Screen ........ 693
24 Earning Money .................................................. 733
Index ....................................................................... 763
# Table of Contents

Introduction .......................................................................................................1

## Part I: Getting Started

**HOUR 1** Preparing for Development ...............................................................5
  - Creating, Deploying, and Profiling an App .................................................5
  - Understanding the App Package .................................................................8
  - Submitting to the Windows Store ...............................................................18
  - Further Exploration ....................................................................................20
  - Summary ..................................................................................................20
  - Q&A .........................................................................................................21
  - Workshop ..................................................................................................22

**HOUR 2** Discovering the XAML Language ....................................................23
  - What Is XAML, Exactly? ..........................................................................23
  - Elements and Attributes ..........................................................................25
  - Namespaces ...............................................................................................25
  - Property Elements ....................................................................................27
  - Type Converters .........................................................................................28
  - Markup Extensions ....................................................................................29
  - Children of Object Elements ...................................................................31
  - Mixing XAML with C# ..............................................................................35
  - Updating the Hour1 Project .......................................................................35
  - Making Your App World-Ready .................................................................42
  - Making Your App Accessible .....................................................................48
  - Further Exploration ..................................................................................53
  - Summary ..................................................................................................54
  - Q&A .........................................................................................................54
  - Workshop ..................................................................................................54
  - Exercises ..................................................................................................55
Part II: Building an App

HOUR 3  Arranging UI Elements ........................................ 57
  Controlling Size ............................................................. 58
  Controlling Position .......................................................... 60
  Applying 2D Transforms ...................................................... 64
  Applying 3D Transforms ...................................................... 70
  Further Exploration .......................................................... 73
  Summary ........................................................................... 73
  Q&A ................................................................................ 73
  Workshop ......................................................................... 77
  Exercises ......................................................................... 77

HOUR 4  Mastering Layout .................................................. 79
  Discovering Your Window Size and Location ......................... 80
  Panels ............................................................................... 84
  Handling Content Overflow ................................................ 97
  Further Exploration .......................................................... 110
  Summary ........................................................................... 110
  Q&A ................................................................................ 110
  Workshop ......................................................................... 112

HOUR 5  Handling Input: Touch, Mouse, Pen, and Keyboard .... 113
  Touch Input ......................................................................... 114
  Mouse Input ......................................................................... 138
  Pen Input ............................................................................. 141
  Keyboard Input .................................................................... 150
  Further Exploration .......................................................... 156
  Summary ........................................................................... 156
  Q&A ................................................................................ 156
  Workshop ......................................................................... 158

HOUR 6  Managing the App Lifecycle .................................... 159
  Killing .............................................................................. 161
  Suspending ........................................................................ 162
  Resuming .......................................................................... 164
### Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminating</td>
<td>165</td>
</tr>
<tr>
<td>Launching</td>
<td>165</td>
</tr>
<tr>
<td>Activating</td>
<td>169</td>
</tr>
<tr>
<td>Managing Session State with SuspensionManager</td>
<td>171</td>
</tr>
<tr>
<td>Programmatically Launching Apps</td>
<td>174</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>177</td>
</tr>
<tr>
<td>Summary</td>
<td>178</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>179</td>
</tr>
<tr>
<td>Workshop</td>
<td>180</td>
</tr>
<tr>
<td><strong>HOUR 7</strong> Threading, Windows, and Pages</td>
<td>181</td>
</tr>
<tr>
<td>Understanding the Threading Model</td>
<td>181</td>
</tr>
<tr>
<td>Displaying Multiple Windows</td>
<td>186</td>
</tr>
<tr>
<td>Navigating Between Pages</td>
<td>190</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>198</td>
</tr>
<tr>
<td>Summary</td>
<td>198</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>198</td>
</tr>
<tr>
<td>Workshop</td>
<td>199</td>
</tr>
<tr>
<td><strong>Part III: Understanding Controls</strong></td>
<td></td>
</tr>
<tr>
<td><strong>HOUR 8</strong> Content Controls</td>
<td>201</td>
</tr>
<tr>
<td>Button</td>
<td>204</td>
</tr>
<tr>
<td>AppBarButton</td>
<td>207</td>
</tr>
<tr>
<td>HyperlinkButton</td>
<td>214</td>
</tr>
<tr>
<td>RepeatButton</td>
<td>215</td>
</tr>
<tr>
<td>ToggleButton</td>
<td>215</td>
</tr>
<tr>
<td>AppBarToggleButton</td>
<td>216</td>
</tr>
<tr>
<td>CheckBox</td>
<td>217</td>
</tr>
<tr>
<td>RadioButton</td>
<td>217</td>
</tr>
<tr>
<td>ToolTip</td>
<td>219</td>
</tr>
<tr>
<td>App Bars</td>
<td>222</td>
</tr>
<tr>
<td>Summary</td>
<td>229</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>229</td>
</tr>
<tr>
<td>Workshop</td>
<td>231</td>
</tr>
<tr>
<td>Exercises</td>
<td>231</td>
</tr>
</tbody>
</table>
### HOUR 9  Items Controls

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items in the Control</td>
<td>234</td>
</tr>
<tr>
<td>Items Panels</td>
<td>236</td>
</tr>
<tr>
<td>ComboBox</td>
<td>239</td>
</tr>
<tr>
<td>ListBox</td>
<td>241</td>
</tr>
<tr>
<td>ListView</td>
<td>242</td>
</tr>
<tr>
<td>GridView</td>
<td>246</td>
</tr>
<tr>
<td>FlipView</td>
<td>248</td>
</tr>
<tr>
<td>SemanticZoom</td>
<td>251</td>
</tr>
<tr>
<td>MenuFlyout</td>
<td>255</td>
</tr>
<tr>
<td>Summary</td>
<td>256</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>256</td>
</tr>
<tr>
<td>Workshop</td>
<td>257</td>
</tr>
<tr>
<td>Exercises</td>
<td>258</td>
</tr>
</tbody>
</table>

### HOUR 10  Text

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextBlock</td>
<td>259</td>
</tr>
<tr>
<td>RichTextBlock</td>
<td>272</td>
</tr>
<tr>
<td>TextBox</td>
<td>277</td>
</tr>
<tr>
<td>RichEditBox</td>
<td>284</td>
</tr>
<tr>
<td>PasswordBox</td>
<td>287</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>289</td>
</tr>
<tr>
<td>Summary</td>
<td>289</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>289</td>
</tr>
<tr>
<td>Workshop</td>
<td>291</td>
</tr>
<tr>
<td>Exercises</td>
<td>291</td>
</tr>
</tbody>
</table>

### HOUR 11  Images

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Image Element</td>
<td>293</td>
</tr>
<tr>
<td>Multiple Files for Multiple Environments</td>
<td>302</td>
</tr>
<tr>
<td>Decoding Images</td>
<td>308</td>
</tr>
<tr>
<td>Encoding Images</td>
<td>318</td>
</tr>
<tr>
<td>Rendering PDF Content as an Image</td>
<td>326</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>331</td>
</tr>
<tr>
<td>Summary</td>
<td>331</td>
</tr>
</tbody>
</table>
Q&A .......................................................... 332
Workshop .................................................. 332
Exercises .................................................. 333

HOUR 12  Audio, Video, and Speech .................. 335
Playing Media ........................................... 336
Capturing Media ........................................ 347
Transcoding Media .................................... 359
Speech Synthesis ....................................... 364
Further Exploration ................................... 366
Summary .................................................. 367
Q&A .......................................................... 368
Workshop .................................................. 369

HOUR 13  Other Controls ............................... 371
Range Controls .......................................... 371
SearchBox .............................................. 374
Popup Controls ........................................ 381
Hub ......................................................... 391
Date and Time Controls ............................. 397
ProgressRing .......................................... 400
ToggleSwitch .......................................... 400
WebView ............................................... 402
Further Exploration .................................. 408
Summary .................................................. 408
Workshop .................................................. 408

Part IV: Leveraging the Richness of XAML

HOUR 14  Vector Graphics ............................ 411
Shapes ..................................................... 411
Geometries ............................................. 419
Brushes ................................................... 427
Further Exploration .................................. 443
Summary .................................................. 443
### HOUR 21  Reading from Sensors

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>647</td>
</tr>
<tr>
<td>Gyrometer</td>
<td>651</td>
</tr>
<tr>
<td>Inclinometer</td>
<td>651</td>
</tr>
<tr>
<td>Compass</td>
<td>651</td>
</tr>
<tr>
<td>Light Sensor</td>
<td>652</td>
</tr>
<tr>
<td>Orientation</td>
<td>652</td>
</tr>
<tr>
<td>Location</td>
<td>653</td>
</tr>
<tr>
<td>Proximity</td>
<td>660</td>
</tr>
<tr>
<td>Summary</td>
<td>663</td>
</tr>
<tr>
<td>Workshop</td>
<td>663</td>
</tr>
</tbody>
</table>

### HOUR 22  Controlling Devices

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingerprint Readers</td>
<td>666</td>
</tr>
<tr>
<td>Image Scanners</td>
<td>666</td>
</tr>
<tr>
<td>Barcode Scanners</td>
<td>670</td>
</tr>
<tr>
<td>Magnetic Stripe Readers</td>
<td>674</td>
</tr>
<tr>
<td>Custom Bluetooth Devices</td>
<td>676</td>
</tr>
<tr>
<td>Custom Bluetooth Smart Devices</td>
<td>679</td>
</tr>
<tr>
<td>Custom USB Devices</td>
<td>682</td>
</tr>
<tr>
<td>Custom HID Devices</td>
<td>686</td>
</tr>
<tr>
<td>Custom Wi-Fi Direct Devices</td>
<td>688</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>690</td>
</tr>
<tr>
<td>Summary</td>
<td>690</td>
</tr>
<tr>
<td>Workshop</td>
<td>691</td>
</tr>
<tr>
<td>Exercises</td>
<td>691</td>
</tr>
</tbody>
</table>

### HOUR 23  Live Tiles, Notifications, and the Lock Screen

<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Tiles</td>
<td>693</td>
</tr>
<tr>
<td>Badges</td>
<td>708</td>
</tr>
<tr>
<td>Secondary Tiles</td>
<td>710</td>
</tr>
<tr>
<td>Toast Notifications</td>
<td>713</td>
</tr>
<tr>
<td>Setting Up Push Notifications</td>
<td>718</td>
</tr>
<tr>
<td>The Lock Screen</td>
<td>726</td>
</tr>
<tr>
<td>Further Exploration</td>
<td>729</td>
</tr>
</tbody>
</table>
Summary ............................................................................ 730
Workshop ............................................................................ 730
Exercises ............................................................................. 731

**HOUR 24  Earning Money** .......................................................... 733
Adding Advertisements to Your App ........................................... 734
Supporting a Free Trial .......................................................... 740
Supporting In-App Purchases ................................................... 744
Validating Windows Store Receipts .......................................... 753
Testing Windows Store Features .............................................. 755
Further Exploration .................................................................. 760
Summary ................................................................................ 760
Q&A .......................................................................................... 761
Workshop ............................................................................. 762

Index ..................................................................................... 763
Adam Nathan is a principal software architect for Microsoft and a best-selling technical author. He introduced XAML to countless developers through his books on a variety of Microsoft technologies. Currently a part of the Windows division, Adam has previously worked on Visual Studio and the Common Language Runtime. He was the founding developer and architect of Popfly, Microsoft’s first Silverlight-based product, named by PCWorld as one of its year’s most innovative products. He is also the founder of PINVOKE.NET, the online resource for .NET developers who need to access Win32. His apps have been featured on Lifehacker, Gizmodo, ZDNet, ParentMap, and other enthusiast sites.

Adam’s books are considered required reading by many inside Microsoft and throughout the industry. Adam is the author of *Windows 8.1 Apps with XAML and C# Unleashed* (Sams, 2013), *101 Windows Phone 7 Apps* (Sams, 2011), *Silverlight 1.0 Unleashed* (Sams, 2008), *WPF Unleashed* (Sams, 2006), *WPF 4 Unleashed* (Sams, 2010), *WPF 4.5 Unleashed* (Sams, 2013), and *.NET and COM: The Complete Interoperability Guide* (Sams, 2002); a coauthor of *ASP.NET: Tips, Tutorials, and Code* (Sams, 2001); and a contributor to books including *.NET Framework Standard Library Annotated Reference, Volume 2* (Addison-Wesley, 2005) and *Windows Developer Power Tools* (O’Reilly, 2006). You can find Adam online at www.adamnathan.net, or @adamnathan on Twitter.
Dedication

To Rose Rudberg.

Acknowledgments

First, I thank Eileen Chan for the encouragement and patience that enabled me to complete this book. I’d also like to give special thanks to J. Boyd Nolan, Ashish Shetty, Tim Heuer, Mark Rideout, Jonathan Russ, Joe Duffy, Chris Brumme, Eric Rudder, Loretta Yates, Neil Rowe, Betsy Gratner, Ginny Munroe, Bill Chiles, and Valery Sarkisov. As always, I thank my parents for having the foresight to introduce me to Basic programming on our IBM PCjr when I was in elementary school.
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.

Email: consumer@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.
Introduction

If you ask me, it has never been a better time to be a software developer. Not only are programmers in high demand—due in part to an astonishingly low number of computer science graduates each year—but app stores make it easier than ever to broadly distribute your own software and even make money from it!

When I was in junior high school, I released a few shareware games and asked for $5 donations. I earned $15 total. One of the three donations was from my grandmother, who didn’t even own a computer! These days, of course, adults and kids alike can make money on simple apps and games without relying on kind and generous individuals going to the trouble of mailing a check!

The Windows Store is an app store like no other, and it keeps getting better. When you consider the number of people who use Windows 8.1 (and Windows RT) compared to the number of people who use any other operating system on the planet, you realize what a unique and enormous opportunity the Windows Store provides. That’s one of the reasons that the Windows Store is the fastest-growing app store in history.

When you write a Windows Store app, you have three main choices for programming language and UI framework pairings:

- JavaScript with an HTML user interface
- C#, Visual Basic, or C++ with a XAML user interface
- C++ with a DirectX user interface

You can also leverage a number of features and componentization techniques to mix and match these languages and UI frameworks within the same app.

C# and XAML has been a very popular choice for writing Windows Store apps. It is the choice for apps such as Netflix, Hulu Plus, Fresh Paint, SkyDrive, Evernote Touch, Reader, Alarms, Movie Moments, Maps, OneNote, Lync, and many, many more. It is also the implementation choice for many core experiences in Windows, such as the PC Settings app, the Search app, and Contact/Calendar functionality in Windows 8.1. The XAML team has stated that their goal is to be the high fidelity, high performance framework for any scenario.
Then why does Microsoft provide so many choices? The idea is to enable you to work with whatever is most comfortable for you, or whatever best leverages your existing assets, or whatever most naturally consumes the third-party SDK you must use.

Your choice can have other benefits. HTML tends to be the best choice if you need to support versions of your app on non-Microsoft platforms or a website. XAML is best at interoperability, as it’s easy to mix both HTML and DirectX content in a XAML app. DirectX, the best choice for hardcore games, provides the most potential for getting the highest performance.

Although your choice of language is generally dictated by your choice of UI Framework, each language has its strengths. JavaScript benefits from a large community that produces interesting libraries. C# has the best features for writing concise asynchronous code. C++ can provide high performance.

The key to the multiple language support is the Windows Runtime, or WinRT for short. WinRT APIs are automatically projected into the programming language you use, so they look natural for that language. Projections are more than just exposing the raw APIs, however. Core WinRT data types such as `String`, collection types, and a few others are mapped to appropriate data types for the target environment. For C# or other .NET languages, this means exposing them as `System.String`, `System.Collections.Generic.IList<T>`, and so on. To match conventions, member names are even morphed to be Camel-cased for JavaScript and Pascal-cased for other languages, which makes the MSDN reference documentation occasionally look goofy.

In the set of APIs exposed by Windows:

- Everything under the `Windows.UI.Xaml` namespace is XAML-specific.
- Everything under the `Windows.UI.WebUI` namespace is for HTML apps.
- Everything under `System` is .NET-specific.
- Everything else (which is under `Windows`) is general-purpose WinRT functionality.

As you dig into the framework, you notice that the XAML-specific and .NET-specific APIs are indeed the most natural to use from C# and XAML.

**Who Should Read This Book?**

This book is for software developers who are interested in creating apps for the Windows Store, whether they are for tablets, laptops, or desktops. It does not teach you how to program, nor does it teach the basics of the C# language. However, it is designed to be understandable even for folks who are new to .NET, and does not require previous experience with XAML.
When building an app, one of the first things you must do is arrange a bunch of elements in its window. This sizing and positioning of elements is called layout. XAML apps are provided a feature-rich layout system that covers everything from placing elements at exact coordinates to building experiences that scale and rearrange across a wide range of screen resolutions and aspect ratios. This is essential for handling the diversity of Windows devices, as well as intelligently handling when your app isn’t the only one on the screen.

In XAML apps, layout boils down to interactions between parent elements and their child elements. Parents and their children work together to determine their final sizes and positions. Parent elements that support the arrangement of multiple children are known as panels, and they derive from a class called Panel. All the elements involved in the layout process (both parents and children) derive from UIElement.

Because layout is such an important topic, the next two hours are dedicated to it. This hour focuses on the children, examining the common ways that you can control layout on a child-by-child basis. Several properties control these aspects, most of which are summarized in Figure 3.1 for an arbitrary element inside an arbitrary panel. Size-related properties are shown in blue, and position-related properties are shown in red. In addition, elements can have transforms applied to them (shown in green) that can affect both size and position.
The next hour continues the layout story by examining the variety of built-in parent panels, each of which arranges its children in unique ways.

**Controlling Size**

Every time layout occurs, such as when an app’s window is resized or the screen is rotated, child elements tell their parent panel their desired size. Elements tend to *size to their content*, meaning that they try to be large enough to fit their content and no larger. This size can be influenced on individual instances of children via several straightforward properties.

**Height and Width**

All FrameworkElements have simple Height and Width properties (of type double), and they also have MinHeight, MaxHeight, MinWidth, and MaxWidth properties that can be used to specify a range of acceptable values. Any or all of these can be easily set on elements in C# or in XAML.

An element naturally stays as small as possible, so if you use MinHeight or MinWidth, it is rendered at that height/width unless its content forces it to grow. In addition, that growth can be limited by using MaxHeight and MaxWidth—as long as these values are larger than their Min counterparts. When using an explicit Height and Width at the same time as their Min and Max counterparts, Height and Width take precedence as long as they are in the range from Min to Max. The default value of MinHeight and MinWidth is 0, and the default value of
MaxHeight and MaxWidth is Double.PositiveInfinity (which can be set in XAML as simply "Infinity").

**CAUTION**

**Avoid setting explicit sizes!**

Giving controls explicit sizes makes it difficult to adapt to different screen sizes and orientations, and could cause text to be cut off if you ever translate it into other languages, as demonstrated in the preceding hour. Therefore, you should avoid setting explicit sizes unless absolutely necessary. Fortunately, setting explicit sizes is rarely necessary, thanks to the panels described in the next hour.

Unlike the six properties that are input to the layout process, FrameworkElement exposes read-only ActualHeight and ActualWidth properties representing output from the layout process: the final size of the element after layout is complete. That's right—whether an element specified an explicit size, specified a range of acceptable sizes, or didn’t specify anything at all, the behavior of the parent can alter an element's final size on the screen. These properties are, therefore, useful for advanced scenarios in which you need to programmatically act on an element's size. The values of the other size-related properties, on the other hand, aren't very interesting to base logic on. For example, when not set explicitly, the value of Height and Width are Double.NaN, regardless of the element’s true size.

**CAUTION**

**Be careful when writing code that uses ActualHeight and ActualWidth!**

Every time the layout process occurs, it updates the values of each element’s ActualHeight and ActualWidth. However, you can’t rely on the values of these properties at all times. It's safe to rely on their values within an event handler for the LayoutUpdated event defined on FrameworkElement. At the same time, you must not alter layout from within a LayoutUpdated handler. If you do, an exception will be thrown pointing out that you introduced a cycle. UIElement defines an UpdateLayout method to force any pending layout updates to occur, but you should avoid using this method. Besides the fact that frequent calls to UpdateLayout can harm performance because of the excess layout processing, there’s no guarantee that the elements you’re using properly handle the potential reentrancy in their layout-related methods.

**Margin and Padding**

Margin and Padding are two similar properties that are also related to an element’s size. All FrameworkElements have a Margin property, and all Controls (plus many other elements) have a Padding property. Their only difference is that Margin controls how much extra space gets placed around the outside edges of the element, whereas Padding controls how much extra space gets placed around the inside edges of the element.
Both Margin and Padding are of type Thickness, an interesting class that can represent one, two, or four double values. Here is how the values are interpreted when set in XAML:

- When set to a list of four values, the numbers represent the left, top, right, and bottom edges, respectively.
- When set to a list of two values, the first number is used for the left and right edges and the second number is used for the top and bottom edges. So "12,24" is a shortcut for "12,12,12,12".
- When set to a single value, it is used for all four sides. So "12" is a shortcut for "12,12,12,12", which is a shortcut for "12,24,12,24".
- Negative values may be used for margins (and often are), but are not allowed for padding.
- The commas are optional. You can use spaces instead of, or in addition to, commas. "12,24" is the same as "12 24" and "12, 24".

When creating a Thickness in C#, you can use its constructor that accepts either a single value or all four values:

```csharp
this.TextBox.Margin = new Thickness(12);  // Margin="12" in XAML
this.TextBox.Margin = new Thickness(12,24,12,24);  // Margin="12,24" in XAML
```

Note that the handy two-number syntax is a shortcut only available through XAML. Thickness does not have a two-parameter constructor.

**Controlling Position**

This section doesn’t discuss positioning elements with (X,Y) coordinates, as you might expect. Parent panels define their own unique mechanisms for enabling children to position themselves, and those are discussed in the next hour. A few mechanisms are common to all FrameworkElement children, however, and that’s what this section examines. These mechanisms are related to alignment and a concept called flow direction.

**Alignment**

The HorizontalAlignment and VerticalAlignment properties enable an element to control what it does with any extra space that its parent panel gives it. Each property has a corresponding enumeration with the same name, giving the following options:

- **HorizontalAlignment**—Left, Center, Right, and Stretch
- **VerticalAlignment**—Top, Center, Bottom, and Stretch
Controlling Position

Stretch is the default value for both properties, although various controls override the setting. The effects of HorizontalAlignment can easily be seen by placing a few Buttons in a StackPanel (a panel described further in the next hour) and marking them with each value from the enumeration:

```xml
<StackPanel>
  <Button HorizontalAlignment="Left" Content="Left" Background="Red"/>
  <Button HorizontalAlignment="Center" Content="Center" Background="Orange"/>
  <Button HorizontalAlignment="Right" Content="Right" Background="Green"/>
  <Button HorizontalAlignment="Stretch" Content="Stretch" Background="Blue"/>
</StackPanel>
```

Notice that an enumeration value such as HorizontalAlignment.Left is able to be specified in XAML as simply Left. This is thanks to a type converter that is able to handle any enumeration. The rendered result appears in Figure 3.2.

![Figure 3.2](image)

**FIGURE 3.2**
The effects of HorizontalAlignment on Buttons in a StackPanel

These two properties are useful only when a parent panel gives the child element more space than it needs. For example, adding VerticalAlignment values to elements in the StackPanel used in Figure 3.2 would make no difference, because each element is already given the exact amount of height it needs (no more, no less).

**NOTE**
When an element uses Stretch alignment (horizontally or vertically), an explicit Height or Width setting still takes precedence. MaxHeight and MaxWidth also take precedence, but only when their values are smaller than the natural stretched size. Similarly, MinHeight and MinWidth take precedence only when their values are larger than the natural stretched size. When Stretch is used in a context that constrains the element’s size, it acts like an alignment of Center (or Left if the element is too large to be centered in its parent).

**Content Alignment**

In addition to HorizontalAlignment and VerticalAlignment properties, the Control class also has HorizontalContentAlignment and VerticalContentAlignment properties. These
properties determine how a control’s content fills the space within the control. (Therefore, the relationship between alignment and content alignment is somewhat like the relationship between Margin and Padding.)

The content alignment properties are of the same enumeration types as the corresponding alignment properties, so they provide the same options. However, the default value for HorizontalContentAlignment is Left, and the default value for VerticalContentAlignment is Top. Some elements implicitly choose different defaults. Buttons, for example, center their content in both dimensions by default.

Figure 3.3 demonstrates the effects of HorizontalContentAlignment, simply by taking the previous XAML snippet and changing the property name as follows:

```xml
<StackPanel>
  <Button HorizontalContentAlignment="Left" HorizontalAlignment="Stretch"
    Content="Left" Background="Red"/>
  <Button HorizontalContentAlignment="Center" HorizontalAlignment="Stretch"
    Content="Center" Background="Orange"/>
  <Button HorizontalContentAlignment="Right" HorizontalAlignment="Stretch"
    Content="Right" Background="Green"/>
  <Button HorizontalContentAlignment="Stretch" HorizontalAlignment="Stretch"
    Content="Stretch" Background="Blue"/>
</StackPanel>
```

Each Button also has its HorizontalAlignment set to Stretch rather than its default of Left so the differences in HorizontalContentAlignment are visible. Without this, each Button would auto-size to its content and there would be no extra space for the content to move within.

**Figure 3.3**
The effects of HorizontalContentAlignment on Buttons in a StackPanel

In Figure 3.3, the Button with HorizontalContentAlignment="Stretch" might not appear as you expected. Its inner TextBlock is technically stretched, but it’s meaningless because TextBlock (which is not a Control) doesn’t have the same notion for stretching its inner text. For other types of content, Stretch can indeed have the intended effect.
FlowDirection

FlowDirection is a property on FrameworkElement (and several other classes) that can reverse the way an element’s inner content flows. It applies to some panels and their arrangement of children, and it also applies to the way content is aligned inside child controls. The property is of type FlowDirection, with two values: LeftToRight (FrameworkElement’s default) and RightToLeft.

The idea of FlowDirection is that it should be set to RightToLeft when the current culture corresponds to a language that is read from right to left. This reverses the meaning of left and right for settings such as content alignment. The following XAML demonstrates this, with Buttons that force their content alignment to Top and Left but then apply each of the two FlowDirection values:

```xml
<StackPanel>
  <Button FlowDirection="LeftToRight"
         HorizontalContentAlignment="Left" Width="320"
         Background="Red">LeftToRight</Button>
  <Button FlowDirection=".RightToLeft"
         HorizontalContentAlignment="Left" Width="320"
         Background="Orange">RightToLeft</Button>
</StackPanel>
```

The result is shown in Figure 3.4.

![Figure 3.4](image)

**Figure 3.4**
The effects of FlowDirection on Buttons with Left content alignment

Notice that FlowDirection does not affect the flow of letters within these Buttons. English letters always flow left to right, and Arabic letters always flow right to left, for example. But FlowDirection reverses the notion of left and right for other pieces of the user interface, which typically need to match the flow direction of letters.

You must explicitly set FlowDirection to match the current culture, but fortunately you can do this on a single, top-level element. Windows doesn’t automatically change FlowDirection on your behalf in order for the behavior to be predictable and easily testable. The idea is that you should specify FlowDirection appropriately inside the .resw file for each distinct culture you support. For example, if you include a resource with the name Root.FlowDirection, then you can mark a Page’s root element with x:Uid="Root" to control FlowDirection on a per-culture basis. The resource just needs to be given the value of LeftToRight or RightToLeft.
Applying 2D Transforms

The XAML UI Framework contains a handful of built-in two-dimensional transform classes (derived from Transform) that enable you to change the size and position of elements independently from the previously discussed properties. Some also enable you to alter elements in more exotic ways, such as by rotating or skewing them.

All UIElements have a RenderTransform property that can be set to any Transform in order to change its appearance after the layout process has finished (immediately before the element is rendered). They also have a handy RenderTransformOrigin property that represents the starting point of the transform (the point that remains stationary). Figure 3.5 demonstrates the impact of setting RenderTransformOrigin to five different (x,y) values when used with one of the Transform objects that performs rotation.

![Figure 3.5](image)

**Figure 3.5**

Five common RenderTransformOrigins used on rotated Buttons rendered on top of unrotated Buttons

RenderTransformOrigin can be set to a Windows.Foundation.Point, with (0,0) being the default value. This represents the top-left corner, shown by the first button in Figure 3.5. An origin of (0,1) represents the bottom-left corner, (1,0) is the top-right corner, and (1,1) is the bottom-right corner. You can use numbers greater than 1 to set the origin to a point outside the bounds of an element, and you can use fractional values. Therefore, (.5,.5) represents the middle of the object. The reason the corner-pivoting appears slightly off in Figure 3.5 is an artifact of the default appearance of Buttons. They have an invisible three-pixel-wide region around their visible rectangle. If you imagine each button extending three pixels in each direction, the pivoting of the first four buttons would be exactly on each corner.

The value for RenderTransformOrigin can be specified in XAML with two comma-delimited numbers (and no parentheses). For example, a Button rotated around its center, like the one at the far right of Figure 3.5, can be created as follows:

```xml
<Button RenderTransformOrigin=".5,.5">
    <Button.RenderTransform>
        <RotateTransform Angle="45"/>
    </Button.RenderTransform>
</Button>
```
This section looks at all the built-in 2D transforms, all in the Windows.UI.Xaml.Media namespace:

- RotateTransform
- ScaleTransform
- SkewTransform
- TranslateTransform
- CompositeTransform
- TransformGroup
- MatrixTransform

**RotateTransform**

RotateTransform, which was just demonstrated, rotates an element according to the values of three double properties:

- **Angle**—Angle of rotation, specified in degrees (default value = 0)
- **CenterX**—Horizontal center of rotation (default value = 0)
- **CenterY**—Vertical center of rotation (default value = 0)

The default (CenterX, CenterY) point of (0,0) represents the top-left corner.

Whereas Figure 3.5 shows rotated Buttons, Figure 3.6 demonstrates what happens when RotateTransform is applied to the inner content of a Button. To achieve this, the simple string that typically appears inside a Button is replaced with an explicit TextBlock as follows:

```xml
<Button Background="Orange">
  <TextBlock RenderTransformOrigin=".5,.5">
    <TextBlock.RenderTransform>
      <RotateTransform Angle="45"/>
    </TextBlock.RenderTransform>
  </TextBlock>
</Button>
```

**FIGURE 3.6**

Using RotateTransform on the content of a Button
**ScaleTransform**

ScaleTransform enlarges or shrinks an element horizontally, vertically, or in both directions. This transform has four straightforward double properties:

- **ScaleX**—Multiplier for the element’s width (default value = 1)
- **ScaleY**—Multiplier for the element’s height (default value = 1)
- **CenterX**—Origin for horizontal scaling (default value = 0)
- **CenterY**—Origin for vertical scaling (default value = 0)

A ScaleX value of 0.5 shrinks an element’s rendered width in half, whereas a ScaleX value of 2 doubles the width. CenterX and CenterY work the same way as with RotateTransform.

Listing 3.1 applies ScaleTransform to three Buttons in a StackPanel, demonstrating the ability to stretch them independently in height or in width. Figure 3.7 shows the result.

**LISTING 3.1  Applying ScaleTransform to Buttons in a StackPanel**

```xml
<StackPanel Width="200">
  <Button Background="Red">No Scaling</Button>
  <Button Background="Orange">
    <Button.RenderTransform>
      <ScaleTransform ScaleX="2"/>
    </Button.RenderTransform>
    X
  </Button>
  <Button Background="Yellow">
    <Button.RenderTransform>
      <ScaleTransform ScaleX="2" ScaleY="2"/>
    </Button.RenderTransform>
    X + Y
  </Button>
  <Button Background="Lime">
    <Button.RenderTransform>
      <ScaleTransform ScaleY="2"/>
    </Button.RenderTransform>
    Y
  </Button>
</StackPanel>
```

Figure 3.8 displays the same Buttons from Listing 3.1 (and Figure 3.7) but with explicit CenterX and CenterY values set. The point represented by each pair of these values is displayed in each Button’s text. Notice that the lime Button isn’t moved to the left like the orange Button, despite being marked with the same CenterX of 70. That’s because CenterX is relevant only when ScaleX is a value other than 1, and CenterY is relevant only when ScaleY is a value other than 1.
FIGURE 3.7
The scaled Buttons from Listing 3.1

FIGURE 3.8
The Buttons from Listing 3.1 but with explicit scaling centers

**SkewTransform**

SkewTransform slants an element according to the values of four double properties:

- **AngleX**—Amount of horizontal skew (default value = 0)
- **AngleY**—Amount of vertical skew (default value = 0)
- **CenterX**—Origin for horizontal skew (default value = 0)
- **CenterY**—Origin for vertical skew (default value = 0)

These properties behave much like the properties of the previous transforms. Figure 3.9 demonstrates SkewTransform applied as a RenderTransform on several Buttons, using the default center of the top-left corner.

**TranslateTransform**

TranslateTransform simply moves an element according to two double properties:

- **X**—Amount to move horizontally (default value = 0)
- **Y**—Amount to move vertically (default value = 0)
TranslateTransform is an easy way to “nudge” elements one way or another. Most likely, you’d do this dynamically based on user actions (and perhaps in an animation). With all the panels described in the next hour, it’s unlikely that you’d need to use TranslateTransform to arrange a static user interface.

Combining Transforms

If you want to transform an element multiple ways simultaneously, such as rotate and scale it, a few different options are available:

- CompositeTransform
- TransformGroup
- MatrixTransform

CompositeTransform

The CompositeTransform class is the easiest way to combine transforms. It has all the properties of the previous four transforms, although some have slightly different names: Rotation, ScaleX, ScaleY, SkewX, SkewY, TranslateX, TranslateY, CenterX, and CenterY.

Figure 3.10 shows several transforms being applied to a single Button as follows:

```xml
<Button Background="Orange">
  <Button.RenderTransform>
    <!-- The composite transform order is always scale, then skew, then rotate, then translate -->
    <CompositeTransform Rotation="45" ScaleX="5" ScaleY="1" SkewX="30"/>
  </Button.RenderTransform>
</Button>
```

FIGURE 3.9
SkewTransform applied to Buttons in a StackPanel
Applying 2D Transforms

FIGURE 3.10
A Button scaled, skewed, and rotated with a single CompositeTransform

It can be handy to always use CompositeTransform instead of the previous transforms, even if you’re only performing one type of transform.

TransformGroup
CompositeTransform always applies its transforms in the same order: scale, skew, rotate, and then translate. If you require a nonstandard order, you can use a TransformGroup instead then put its child transforms in any order. For example, the following XAML looks like it might have the same effect as the previous XAML, but Figure 3.11 shows that the result is much different:

```xml
<Button Background="Orange">
    <Button.RenderTransform>
        <TransformGroup>
            <!-- First rotate, then scale, then skew! -->
            <RotateTransform Angle="45"/>
            <ScaleTransform ScaleX="5" ScaleY="1"/>
            <SkewTransform AngleX="30"/>
        </TransformGroup>
    </Button.RenderTransform>
</Button>
```

FIGURE 3.11
This time, the Button is rotated, scaled, and then skewed.

TransformGroup is just another Transform-derived class, so it can be used wherever any transform is used.
For maximum performance, the system calculates a combined transform out of a TransformGroup's children and applies it as a single transform, much as if you had used CompositeTransform. Note that you can apply multiple instances of the same transform to a TransformGroup. For example, applying two separate 45° RotateTransforms would result in a 90° rotation.

MatrixTransform

MatrixTransform is a low-level mechanism that can be used to represent all combinations of rotation, scaling, skewing, and translating. MatrixTransform has a single Matrix property (of type Matrix) representing a 3x3 affine transformation matrix. (Affine means that straight lines remain straight.) Its Matrix property has the following subproperties representing 6 values in a 3x3 matrix:

\[
\begin{bmatrix}
M11 & M12 & 0 \\
M21 & M22 & 0 \\
OffsetX & OffsetY & 1
\end{bmatrix}
\]

The final column’s values cannot be changed.

NOTE

MatrixTransform is the only transform that can be specified as a simple string in XAML. For example, you can translate a Button 10 units to the right and 20 units down with the following syntax:

```xml
<Button RenderTransform="1,0,0,1,10,20"/>
```

The comma-delimited list represents the M11, M12, M21, M22, OffsetX, and OffsetY values, respectively. The values 1, 0, 0, 1, 0, 0 give you the identity matrix (meaning no transform is done), so making MatrixTransform act like TranslateTransform is as simple as starting with the identity matrix and then using OffsetX and OffsetY as TranslateTransform's X and Y values. Scaling can be done by treating the first and fourth values (the 1s in the identity matrix) as ScaleX and ScaleY, respectively. Rotation and skewing are more complicated because they involve sin, cos, and angles specified in radians.

If you’re comfortable with the matrix notation, representing transforms with this concise (and less-readable) syntax can be a time saver when you’re writing XAML by hand.

Applying 3D Transforms

Although you can integrate DirectX into a XAML app to work with a full 3D graphics engine, the XAML UI Framework does not directly expose full 3D capabilities. It does, however, enable you to perform the most common 3D effects with perspective transforms. These transforms escape
Applying 3D Transforms

the limitations of the 2D transforms by enabling you to rotate and translate an element in any or all of the three dimensions.

Perspective transforms are normally done with a class called *PlaneProjection*, which defines RotationX, RotationY, and RotationZ properties. The X and Y dimensions are defined as usual, and the Z dimension extends into and out of the screen, as illustrated in Figure 3.12. X increases from left-to-right, Y increases from top-to-bottom, and Z increases from back-to-front.

**FIGURE 3.12**
The three dimensions, relative to the screen

Although plane projections act like transforms, they derive from a class called *Projection* rather than Transform. Therefore, one cannot be assigned to an element via its RenderTransform property, but rather a separate property called Projection. The following plane projections are marked on playing card images, producing the result in Figure 3.13:

```xml
<StackPanel Orientation="Horizontal">
  <Image Source="Images/CardHA.png" Width="150" Margin="12">
    <Image.Projection>
      <PlaneProjection RotationX="55"/>
    </Image.Projection>
  </Image>
  <Image Source="Images/CardH2.png" Width="150">
    <Image.Projection>
      <PlaneProjection RotationY="55"/>
    </Image.Projection>
  </Image>
  <Image Source="Images/CardH3.png" Width="150" Margin="36">
    <Image.Projection>
      <PlaneProjection RotationZ="55"/>
    </Image.Projection>
  </Image>
</StackPanel>
```
FIGURE 3.13
Using a plane projection to rotate the card around the X, Y, and Z axes, then all three axes

Notice that rotating around only the Z axis is like using a 2D RotateTransform, although the direction is reversed.

Much like the 2D transform classes, PlaneProjection defines additional properties for changing the center of rotation: CenterOfRotationX, CenterOfRotationY, and CenterOfRotationZ. The first two properties are relative to the size of the element, on a scale from 0 to 1. The CenterOfRotationZ property is always in terms of absolute pixels, as elements never have any size in the Z dimension to enable a relative specification.

PlaneProjection defines six properties for translating an element in any or all dimensions. GlobalOffsetX, GlobalOffsetY, and GlobalOffsetZ apply the translation after
the rotation, so the offsets are relative to the global screen coordinates. LocalOffsetX, LocalOffsetY, and LocalOffsetZ apply the translation before the rotation, causing the rotation to be relative to the rotated coordinate space.

NOTE
One other type of projection exists that can be assigned to an element’s Projection property: Matrix3DProjection, which also derives from the Projection base class. This is a low-level construct that enables you to specify the projection as a 4x4 3D transformation matrix. This can be handy if you are already working with 3D transformation matrices, otherwise the simpler PlaneProjection is all you need to use.

Further Exploration

Probably the best way to get a feel for the various layout properties is to experiment with them in the Visual Studio XAML designer. You can create a simple StackPanel with Buttons, such the ones shown in this hour, then watch how things change as you experiment with different properties and values. This can be especially helpful for understanding complicated properties such as RenderTransform and Projection.

Summary

That concludes our tour of the layout properties that child elements can use to influence the way they appear on the screen. The most important part of layout, however, is the parent panels. This hour repeatedly uses a simple StackPanel for simplicity, but the next hour formally introduces this panel and all the other panels as well.

Q&A

Q. What unit of measurement is used by XAML?

A. The various length properties use units of pixels, although they are considered logical pixels because they do not always map directly to physical pixels on the screen. Windows automatically scales your app content to either 100%, 140%, or 180% of its natural size based on the size and resolution (and therefore dots-per-inch, or DPI) of the screen. This is why Visual Studio provides different screen size options for the same resolutions in both the simulator and the designer. The latter can be controlled via the Device tool window shown in Figure 3.14.
HOUR 3: arranging UI Elements

Figure 3.14
Size matters when it comes to the device’s screen, due to the automatic scaling called out in the “Display” dropdown.

The effect of this scaling can be easily seen with the following XAML placed in a Page:

```xaml
<StackPanel HorizontalAlignment="Left">
    <Rectangle Fill="Red" Height="100" Width="1000"/>
    <TextBlock Name="textBlock" FontSize="40"/>
</StackPanel>
```

with code-behind that sets the TextBlock’s Text to the Page’s current dimensions as reported by the XAML UI Framework:

```csharp
this.textBlock.Text = "Page size: " + this.ActualWidth + "x" + this.ActualHeight;
```

Figure 3.15 shows the result on two different simulator settings. In both cases, the red Rectangle is 1,000 units long as far as your code is concerned. However, it gets scaled larger on the smaller screen (and the Page’s reported bounds are now smaller than the true resolution). Depending on the screen, a user can also bump up the scale from 100% to 140% or 140% to 180% by selecting the “Change the size of apps on the displays that can support it” option in the “Display” section of the PC Settings app.
Q. What's the difference between using the CenterX and CenterY properties on transforms such as RotateTransform versus using the RenderTransformOrigin property on UIElement?

A. When a transform is applied to a UIElement, the CenterX and CenterY properties at first appear to be redundant with RenderTransformOrigin. Both mechanisms control the origin of the transform.

However, CenterX and CenterY enable absolute positioning of the origin rather than the relative positioning of RenderTransformOrigin. Their values are specified as logical pixels, so the top-right corner of an element with a Width of 20 would be specified with CenterX set to 20 and CenterY set to 0 rather than the point (1,0). Also, when multiple
RenderTransforms are grouped together (described later in this hour), CenterX and CenterY on individual transforms enable more fine-grained control.

That said, RenderTransformOrigin is generally more useful than CenterX and CenterY. For the common case of transforming an element around its middle, the relative (.5,.5) RenderTransformOrigin is easy to specify in XAML, whereas accomplishing the same thing with CenterX and CenterY would require writing some C# code to calculate the absolute offsets.

Note that you can use RenderTransformOrigin on an element simultaneously with using CenterX and CenterY on its transform. In this case, the two X values and two Y values are combined to calculate the final origin point.

Q. How do transforms such as ScaleTransform affect FrameworkElement’s ActualHeight and ActualWidth properties?

A. Applying a transform to FrameworkElement never changes the values of these properties. Therefore, because of transforms, these properties can “lie” about the size of an element on the screen. For example, all the Buttons in Figures 3.9 and 3.10 have the same ActualHeight and ActualWidth.

Such “lies” might surprise you, but they’re usually for the best. The point of transforms is to alter an element’s appearance without the element’s knowledge. Giving elements the illusion that they are being rendered normally enables arbitrary controls to be plugged in and transformed without special handling.

Q. How does ScaleTransform affect Margin and Padding?

A. Padding is scaled along with the rest of the content (because Padding is internal to the element), but Margin does not get scaled. As with ActualHeight and ActualWidth, the numeric Padding property value does not change, despite the visual scaling.

Q. Given the impact of transforms, how can I easily get the true position and size of an element?

A. Sometimes you do really need to know an element’s final position and size, taking into account the effects from transforms. For example, several Windows Runtime APIs require the position or bounds of an on-screen element so Windows can display system UI in the right spot.

You can get a window-relative position that accounts for all transforms as follows:

```csharp
static Point GetPositionInWindow(FrameworkElement element)
{
    GeneralTransform transform = element.TransformToVisual(null);
    return transform.TransformPoint(new Point(0,0)); // Represents 0,0 */
}
```
You can get the bounding rectangle as follows:

```csharp
static Rect GetBoundsInWindow(FrameworkElement element)
{
    GeneralTransform transform = element.TransformToVisual(null);
    return transform.TransformBounds(new Rect(0, 0, element.ActualWidth, element.ActualHeight));
}
```

### Workshop

### Quiz

1. What event on FrameworkElement is raised its layout has been triggered?

2. When specifying four values for a Thickness value such as Margin, what is the meaning of each value, in order?

3. What property can be set on an element to reverse its notion of left and right?

### Answers

1. LayoutUpdated.

2. Left, Top, Right, and then Bottom.

3. FlowDirection.

### Exercises

1. Place a Button within a Page and choose a Margin for the Button. Then try to produce the same result by replacing the Margin with a TranslateTransform instead. Then try to produce the same result by replacing the TranslateTransform with a MatrixTransform.

2. Visualize a playing card positioned in 3D space. Then take the playing card example from Figure 3.13, and try to update one of the card’s PlaneProjections to match the angle you’re picturing.
Symbols & Numerics

{ (open curly braces),
  escaping, 30
2D transforms
  RenderTransform property, 64
  RotateTransform, 65
  ScaleTransform, 66
  SkewTransform, 67-68
  TranslateTransform, 67-68
3D transforms, 70-73

A

AAMVA (American Association of Motor Vehicle Administrators) cards, 675
absolute sizing, 92

accelerometer, reading from, 647-651
  shaking, 650
  tossing motion, 649-650
accessibility
  handling high contrast themes, 51-53
  improving screen reading, 49-50
  UI Accessibility Checker, 49
accessing known folders, 563
Account Picture Provider contract, 613-615
activating apps, 165, 169-171
activation contracts. See contracts
AddDeleteThemeTransition, 455-456
adding ads to pages, 736-738
advertisements to apps, 734-738
advanced AdControl behaviors, 738-739
defining ad units in pubCenter, 734-739
effects to media playback, 340-341
key/value pairs to dictionaries, 33-34
adjusting camera settings (MediaCapture), 355-357
advanced AdControl behaviors, 738-739
advertisements, adding to apps, 734-739
adding ads to pages, 736-738
advanced AdControl behaviors, 738-739
defining ad units in pubCenter, 734-739
alignment
content alignment, 61-63
FlowDirection property, 63
HorizontalAlignment property (FrameworkElement), 60-61
VerticalAlignment property (FrameworkElement), 60-61
animation
custom animations, 464-475
controlling duration, 467
independent versus dependent animations, 465-466
From property, 467-469
To property, 467-469
property paths, 473-475
Storyboards with multiple animations, 472-473
Timeline properties, 469-472
easing functions, power
easing functions, 481
keyframes, 475-480
discrete keyframes, 478-480
easing, 480
linear keyframes, 475-477
spline keyframes, 477-478
manual animations, 484-486
theme animations, 459-464
tweaking, 463-464
theme transitions, 448-459
AddDeleteThemeTransition, 455-456
app bars, 222-229
app data, 553-559
app files, 556-559
local files, 557
packaged files, 557
roaming files, 558-559
app settings, 554-556
local settings, 554-555
roaming settings, 555-556
app files
local files, 557
packaged files, 557
roaming files, 558-559
app settings, 554-556
local settings, 554-555
roaming settings, 555-556
app themes, 202
### automated Windows Store testing

765

---

**AppBar control**, 224-225
**AppBarButton control**, 207-214
  - icons, 208-214
  - BitmapIcon, 212-214
  - PathIcon, 211-212
  - SymbolIcon, 208-209
**AppBarToggleButton**
**control**, 216
**Application class, methods**, 169-170
**Application tab (package manifest)**, 8
  - applying borders to windows, 195-197
  - theme transitions to elements, 448-449
**Appointment Provider contract**, 634-636
  - Appointment Provider sources, 635
  - Appointment Provider targets, 635-636
**apps**
  - accessibility
    - handling high contrast themes, 51-53
    - improving screen reading, 49-50
    - UI Accessibility Checker, 49
  - activating, 165
  - advertising, 734-739
  - adding ads to pages, 736-738
  - advanced AdControl behaviors, 738-739
  - defining ad units in pubCenter, 734-739
  - capabilities, restricting, 14-15
  - execution states, 160-161
  - free trials
    - feature-differentiated trials, 740-742
    - time-based trials, 740
  - globalization, 42
  - Hour1 project, updating, 39-42
  - in-app purchases, 744-753
  - launching
    - LaunchActivatedEventArgs, 166-167
    - programmatically launching apps, 174-177
  - lifecycle, 159-161
  - activating apps, 169-171
  - killing apps, 161-162
  - launching apps, 165-169
  - resuming apps, 164-165
  - session state, managing, 171-174
  - session state, representing, 171-172
  - suspending apps, 162-164
  - terminating apps, 165
  - listings, 19-20
  - localization, 42-48
    - Pseudo Language, 45-48
  - promotional images, 20
  - running in background, 161
  - splash screen, customizing, 9
  - submitting to Windows Store, 18-20
  - supporting full license purchases, 742-744
  - suspended, 159
  - Visual Studio launching, 5-7
  - troubleshooting, 7
  - windows displaying multiple windows, 186-189
**artifacts (image)**, 302
**asynchronous methods**, 182-184
**attached properties**, 85-86
  - VariableSizeWrapGrid panel, 95-96
**attributes, XAML**, 25
**audio**
  - background audio, 636-640
  - capturing, 358-359
  - custom media formats, 345-347
  - customizing playback, 338-339
  - MediaElement, 341-343
  - MediaPlayer, 343-344
  - playing
    - adding effects, 340-341
    - markers, 339-340
    - media content, 337-338
**automated Windows Store testing**, 759-760

---

*How can we make this index more useful? Email us at indexes@samspublishing.com*
AutoPlay contract, 616-617
AutoPlay Device contract, 616-617
AutoReverse property (Timeline), 470
autosizing, 92
awaiting a method, 183
awaiting an asynchronous operation, 182-184

B

BackEase function, 481
background color, specifying for tile images, 12-13
background tasks, 636-645
  background audio, 636-640
  custom background tasks, 641-645
background threads, 181-182
background transfers, 569-572
badges, 708-710
bank cards, 675
barcode scanners, 670-674
  BarcodeScanner instance, obtaining, 671
  claiming the device, 671-673
  enabling the device, 673
  retrieving data, 673-674
BarcodeScanner instance, obtaining, 671
BeginTime property (Timeline), 469-470
behaviors
  advanced AdControl behaviors, 738-739
  ButtonBase behaviors, 206-207
Bézier curves, 420
Binding markup extension, 525-527
  in C#, 527
bitmap-based brushes, 435-442
  ImageBrush, 435-438
  WebViewBrush, 437-442
BitmapDecoder class, decoding images, 308-317
  getting pixel data, 310-313
BitmapEncoder class, 318-325
  writing pixel data, 319-321
BitmapIcon, 212-214
BitmapProperties reading metadata, 315-316
Blend, 24
Bluetooth devices, 676-679
  communicating with, 678-679
  connecting to, 677-678
  declaring right device capability, 676-677
Bluetooth Smart devices
  communicating with, 681-682
  connecting to, 680-681
  declaring the right capability, 679-680
borders, applying to windows, 195-197
BounceEase function, 481

Brushes
  bitmap-based brushes, 435-442
  ImageBrush, 435-438
  WebViewBrush, 437-442
  color brushes, 428-435
  LinearGradientBrush, 429-435
  SolidColorBrush, 428-429
built-in controls, 203
content controls
  AppBar control, 224-225
  AppBarButton control, 207-214
  AppBarToggleButton control, 216
  Button control, 204-207
  CheckBox control, 217
  CommandBar control, 226-229
  HyperlinkButton control, 214
  RadioButton control, 217-219
  RepeatButton control, 215
  ToggleButton control, 215-216
  ToolTip control, 219-222
items controls, 233-234
ComboBox control, 239-241
GridView control, 246-247
ItemsControl control, 234-236
ListBox control, 241-242
ListView control, 242-246
MenuFlyout control, 255
panels, 236-239
text controls, 259
PasswordBox control, 287-289
RichEditBox control, 284-287
RichTextBlock control, 272-277
textBlock control, 259-271
TextBox control, 277-284
built-in panels, 84-97
Canvas, 84-86
Grid, 87-94
StackPanel panel, 86-87
VariableSizedWrapGrid panel, 94-97
bulk transfers, 684-685
Button control, 204-207
ButtonBase behaviors, 206-207
Flyouts, associating, 204-206
Canvas panel, 84-86
attached properties, 85-86
hit testing, 124
capabilities
device capabilities, 15-16
identity capabilities, 17
Location capability, 653-660
geofencing, 657-660
getting current location, 653-656
C#
Binding markup extension, 527
combining with XAML, 35
comparing to XAML, 25
composing JavaScript, 406-407
cached composition, 443-444
caching
controlling, 568-569
Pages, 193-194
camera settings, adjusting, 35-357
CameraCaptureUI, capturing media
photos, 348-350
video, 350-351
Canvas panel, 84-86
attached properties, 85-86
hit testing, 124
capabilities
device capabilities, 15-16
identity capabilities, 17
Location capability, 653-660
geofencing, 657-660
getting current location, 653-656
MouseCapabilities class, 139
network capabilities, 16
restricting, 14-15
Capabilities tab (package manifest), 13-17
file capabilities, 15
CaptureElement, 351
capturing
audio, 358-359
media, 347-359
photos, 348-350
video, 350-351
Points, 120-122
video, 357-358
web content with WebView control, 407-408
certification process, Windows Store, 19-20
charms, 579
Devices charm, 592-603
customizing projection, 603
Play To feature, 601-603
printing, 593-601
Search charm, 580-585
customizing the Search pane, 581-584
showing Search pane programmatically, 585
Windows 8 Search contract, 580-581
Settings charm, 603-608
handling links with SettingsPane, 605-608
showing links, 604
How can we make this index more useful? Email us at indexes@samspublishing.com
Share charm, 585-592
share sources, 585-589
share targets, 589-592

CheckBox control, 217

child elements, layout, 57-58

child object elements
clipping, 98-100
collection items, 32-34
content property, 31-32
stacking, 86-87
type converted values, 34

CircleEase function, 481

classes
Application class, 169-170
Control class, 53
DependencyObject class, 53
Frame class, 191-193
FrameworkElement class, 53
FlowDirection property, 63
Height properties, 58-59
Margin property, 59-60
Padding property, 59-60
VerticalAlignment property, 60-61
Width properties, 58-59
Geolocator class, 653-660
GestureRecognizer class (WinRT), 128-131
InkManager class (WinRT), 142-150
MouseCapabilities class, 139
MouseDevice class, 139

NavigationHelper class, 194-195
.NET, using in XAML, 34
Object class, 53
Page class, 53
Pointer class, 115
PointerDevice class, 114-115
PointerPoint class, 115-116
properties, 116
RenderTargetBitmap class, 302
SuspensionManager class, 172-174
Thickness class, 60
UIElement class, 53
gesture events, 133
WriteableBitmap class, 299-302

clipping, 98-100
code-behind files
combining C# and XAML, 35
Hour1 project, 37-41
collections, 32-34
data binding, 530-533
dictionaries, 33-34
view, customizing, 541-545
 grouping, 542-545
 navigating, 545
color, handling high contrast themes, 51-53
color brushes, 428-435
LinearGradientBrush, 429-435
SolidColorBrush, 428-429
columns
arranging children objects with Grid panel, 87-94
sizing, 92-94
combining
C# and XAML, 35
transforms
CompositeTransform, 68-69
MatrixTransform, 70
TransformGroup, 69-70
ComboBox control, 239-241
CommandBar control, 226-229
communicating with NFC peer device, 662-663
comparing
C# and XAML, 25
gestures and manipulations, 133-134
independent and dependent animations, 465-466
compass, 651-652
composing
HTML with XAML, 405-406
JavaScript with C#, 406-407
CompositeTransform, 68-69
conditions for suspending apps, 163
configuring scan sources, 669
connecting to Bluetooth devices, 677-678
constructors, sizing Grid panel, 93-94
consumable products, in-app purchases, 748-753
AppBarToggleButton control, 216
Button control, 204-207
CheckBox control, 216
CommandBar control, 226-229
RadioButton control, 217-219
RepeatButton control, 215
ToggleButton control, 215-216
toolTip control, 219-222
date and time controls, 397-399
DatePicker control, 397-398
timePicker control, 399
explicit sizing, 59
Hub, 391-397
Hub control, HubSections, 394-397
items controls, 233-234
ComboBox control, 239-241
FlipView control, 248-250
GridView control, 246-247
ItemsControl control, 234-236
ListBox control, 241-242
ListView control, 242-246
MenuFlyout control, 255
panels, 236-239
popup controls, 381-391
CoreWindowDialog control, 381-385
CoreWindowFlyout control, 385-386
MessageDialog control, 386-387
Popup control, 389-391
PopupMenu control, 387-389
ProgressRing control, 400
range controls, 371-374
ProgressBar control, 371-372
Slider control, 372-374
ScrollBar control, 377-384
text controls, 259
PasswordBox control, 287-289
RichEditBox control, 284-287
RichTextBlock control, 272-277
textBlock control, 259-271
textBox control, 277-284
toggleSwitch control, 400-402
WebView control, 402-408
visual states, 512-521
responding to changes, 513-516
visual transitions, 516-521
WebApp control, 402-408
capturing web content, 407-408
composing HTML with XAML, 405-406
composing JavaScript with C#, 406-407
navigation, 403-405
cookies, manipulating, 568
CoreDispatcher mechanism, 185
CreateWindowAsync method, 186-189
CrossSliding event, 128
curly braces ({}), escaping, 30
current location, obtaining, 653-656
CurrentAppSimulator, 756-759
curves, Bézier curves, 420
custom animations, 464-475
controlling duration, 467
independent versus dependent animations, 465-466
keyframes, 475-480
discrete keyframes, 478-480
easing, 480
linear keyframes, 475-477
spline keyframes, 477-478
From property, 467-469
To property, 467-469
property paths, 473-475
Storyboards with multiple animations, 472-473
Timeline properties, 469-472
custom background tasks, 641-645
custom Bluetooth devices, 676-679
communicating with, 678-679
connecting to, 677-678
declaring right device capability, 676-677
custom HID devices
communicating with, 687-688
connecting to, 687
declaring the right capability, 686
custom media formats, 345-347
custom USB devices, 682-685
communicating with, 684-685
bulk transfers, 684-685
control transfers, 684
interrupt transfers, 685
connecting to, 683-684
declaring the right capability, 682-683
customizing
app launch, 175-177
collection view, 541-545
grouping, 542-545
navigating, 545
logo images for apps, 10-13
media playback, 338-339
page rendering, 328-331
ScrollViewer, 101-104
Search pane, 581-584
splash screen for apps, 9
text rendering, 261-265
known folders, accessing, 563
libraries, 563-564
data binding, 525
app data, app files, 556-559
Binding markup extension, 525-527
in C#, 527
to collections, 530-533
controlling rendering, 533-541
value converters, 537-541
customizing the data flow, 528-529
to plain properties, 527-528
sharing source with DataContext, 529-530
data templates, 534-537
data virtualization, 246
DataContext, sharing data binding source with, 529-530
date and time controls, 397-399
DatePicker control, 397-398
TimePicker control, 399
DatePicker control, 397-398
Declarations tab (package manifest), 17
declaring
Bluetooth device capability, 676-677
XAML elements, 25
XML namespaces, 26

How can we make this index more useful? Email us at indexes@samspublishing.com
decoding images, 308-317
  getting pixel data, 310-313
  reading metadata, 313-317
    BitmapProperties, 315-316
    ImageProperties, 314-315
    raw metadata, 317
deferrals, handling suspended apps, 164
defining ad units in pubCenter, 734-739
DependencyObject class, 53
detecting
  gestures, 128-131
  keyboard input, 150-156
  manipulations, 133-138
  mouse input, 138-141
  orientation, 652-653
  pen input, 141-150
  shaking with accelerometer, 650
  tossing motion with accelerometer, 649-650
developer tools used in this book, downloading, 3
device capabilities, 15-16
device scenario APIs, 665
  barcode scanners, 670-674
    BarcodeScanner instance, obtaining, 671
    claiming the device, 671-673
    enabling the device, 673
  retrieving data, 673-674
  Bluetooth Smart devices
    communicating with, 681-682
    connecting to, 680-681
    declaring the right capability, 679-680
  custom Bluetooth devices, 676-679
    communicating with, 678-679
    connecting to, 677-678
    declaring right device capability, 676-677
  custom HID devices
    communicating with, 687-688
    connecting to, 687
    declaring the right capability, 686
  custom USB devices, 682-685
    communicating with, 684-685
    connecting to, 683-684
    declaring the right capability, 682-683
  fingerprint readers, 666
  image scanners, 666-670
  configuring scan sources, 669
    ImageScanner instance, obtaining, 667
    performing scans, 667-668
    previews, displaying, 669-670
  magnetic stripe readers, 674-676
  Wi-Fi devices, 688-690
Devices charm, 592-603
  customizing projection, 603
  Play To feature, 601-603
  printing, 593-601
    adding custom options, 600-601
    changing default options, 599
    configuring displayed options, 599-600
dictionaries, 33-34
digitizers, 141
discovering
  window orientation, 82-84
  window size, 80-81
discrete keyframes, 478-480
Dispatcher, 185
displaying multiple windows, 186-189
downloading
  developer tools used in this book, 3
  source code examples in this book, 3
  Visual Studio, 5
drag-and-drop support, enabling for UIElements, 120-122
Dragging events, 140-141
DragItemThemeAnimation, 463
DragOverThemeAnimation, 463
DropTargetItemThemeAnimation, 463
durable products, in-app purchases, 746-748
duration, controlling for custom animations, 467
dynamic color, handling high contrast themes, 51-53
dynamic images
  generating with RenderTargetBitmap class, 302
generating with WriteableBitmap, 299-302

enabling
  barcode scanners, 673
drag-and-drop support for UIElements, 120-122
  incremental item rendering, 547
encoding
  images, 318-325
  writing metadata, 321-322
  writing pixel data, 319-321
  transcoding, 322-325
Energy Consumption tool (Visual Studio), 7
enlarging elements,
  ScaleTransform, 66
EntranceThemeTransition, 450-451
escaping curly braces, 30
event bubbling, 117-119
  halting, 119-120
event handlers
  LayoutUpdated event handler, 80
  OnSuspending event handler, 163-164
  for routed events, 118
events. See also event handlers
  Dragging events, 140-141
gesture events, 128
  UIElement class, 133
  manipulation events, 134-138
  inertia, 138
  media, 339

File Open Picker declaration, 622-626
file picker, 560

F
FadeInThemeAnimation, 462
FadeOutThemeAnimation, 462
feature-differentiated trials, 740-742
fetching data over networks, 564-575
  background transfers, 569-572
  HTTP requests, 565-569
  controlling caching, 568-569
  GET requests, 565-567
  HTTP filters, 567
  HTTP prefetching, 569
  manipulating cookies, 568
  sockets, 572
  syndication, 572-574
file capabilities, 15

How can we make this index more useful? Email us at indexes@samspublishing.com
File Save Picker contract, 626-627
File Type Associations declarations, 619-620
files, 551-553
  app files, 556-559
  local files, 557
  packaged files, 557
  roaming files, 558-559
custom media formats, 345-347
loading
  automatic loading, 303-306
  manual loading, 306-307
programmatically launching apps for, 174
  referencing with URIs, 294-296
trimming, 363-364
FillBehavior property
  (Timeline), 472
FillRule, 423
filters, HTTP, 567
fingerprint readers, 666
FlipView control, 248-250
flow direction, 60
FlowDirection property
  (FrameworkElement), 63
Flyouts, 204-206
focus, 155-156
folder picker, 561-562
folders, 551-553
  known folders, accessing, 563
foreign languages
  localization, 42-48
Microsoft Local Language Portal, 48
Microsoft Translator service, 46
Frame class, 195-197
  navigating between Pages, 191-193
FrameworkElement class, 53
  Height properties, 58-59
  HorizontalAlignment property, 60-61
  Padding property, 59-60
  RequestedTheme property, 202
  VerticalAlignment property, 60-61
  Width properties, 58-59
free trials
  feature-differentiated trials, 740-742
  time-based trials, 740
full license purchases, supporting, 742-744
functions, easing functions, 480-484
  power easing functions, 481
Geolocation class, 653-660
Geolocator class, 653-660
Geometries, 419-427
  Bézier curves, 420
  FillRule, 423
  GeometryGroup, 423-425
  PathFigures, 420-422
  PathSegments, 420-422
  representing as strings, 425-427
GeometryGroup, 423-425
GestureRecognizer class
  (WinRT), 128-131
  interacting with, 129-131
  mouse dragging events, recognizing, 140
gestures, 127-133
  EdgeGesture, 132-133
  events, 128
  manipulations, 133-138
  inertia, 138
  UIElement gesture events, 133
GET requests, 565-567
GetAsyncKeyState method, 152-154
GetIntermediatePoints method, 118
GetKeyState method, 152-154
GetPosition method, 118
globalization, 42
Grid panel, 87-94
GridView control, 246-247
  high-performance rendering, 545-549
gyrometer, 651
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halting, event bubbling</td>
<td>119-120</td>
</tr>
<tr>
<td>Handling</td>
<td></td>
</tr>
<tr>
<td>Content overflow, clipping</td>
<td>97-109</td>
</tr>
<tr>
<td>Interactive zooming, scaling</td>
<td>105-109</td>
</tr>
<tr>
<td>Scrolling, snapping points</td>
<td>99-105</td>
</tr>
<tr>
<td>Snap points</td>
<td>104-105</td>
</tr>
<tr>
<td>Viewbox</td>
<td>106-108</td>
</tr>
<tr>
<td>Crashes</td>
<td>162</td>
</tr>
<tr>
<td>High contrast themes, keyboard input, focus, key states, retrieving</td>
<td>152-154</td>
</tr>
<tr>
<td>Keyboard input, keyboard modifiers in pointer events</td>
<td>154</td>
</tr>
<tr>
<td>Links with SettingsPane</td>
<td>605-608</td>
</tr>
<tr>
<td>Mouse input</td>
<td>138-141</td>
</tr>
<tr>
<td>Mouse-specific APIs</td>
<td>139</td>
</tr>
<tr>
<td>Pen input</td>
<td>141-150</td>
</tr>
<tr>
<td>Digitizers</td>
<td>141</td>
</tr>
<tr>
<td>Palm rejection</td>
<td>141</td>
</tr>
<tr>
<td>Suspending event</td>
<td>163-164</td>
</tr>
<tr>
<td>Touch input</td>
<td>114-138</td>
</tr>
<tr>
<td>Gestures</td>
<td>127-133</td>
</tr>
<tr>
<td>Manipulations</td>
<td>133-138</td>
</tr>
<tr>
<td>Pointers</td>
<td>114-127</td>
</tr>
<tr>
<td>Handwriting</td>
<td></td>
</tr>
<tr>
<td>Recognizing</td>
<td>147-150</td>
</tr>
<tr>
<td>Rendering</td>
<td>142-148</td>
</tr>
<tr>
<td>Hardcoded color, handling high contrast themes</td>
<td>51-53</td>
</tr>
<tr>
<td>Height properties</td>
<td>58-59</td>
</tr>
<tr>
<td>HID devices</td>
<td></td>
</tr>
<tr>
<td>Communicating with</td>
<td>687-688</td>
</tr>
<tr>
<td>Connecting to, declaring the right capability</td>
<td>686</td>
</tr>
<tr>
<td>High contrast themes, handling</td>
<td>51-53</td>
</tr>
<tr>
<td>High-performance rendering</td>
<td>545-549</td>
</tr>
<tr>
<td>Hijacking dependency properties</td>
<td>511</td>
</tr>
<tr>
<td>History suggestions (SearchBox)</td>
<td>375-376</td>
</tr>
<tr>
<td>Hit testing</td>
<td>123-124</td>
</tr>
<tr>
<td>Holding event</td>
<td>128</td>
</tr>
<tr>
<td>HorizontalAlignment property (FrameworkElement)</td>
<td>60-61</td>
</tr>
<tr>
<td>HorizontalContentAlignment property (Control class)</td>
<td>61-62</td>
</tr>
<tr>
<td>Hour1 project</td>
<td></td>
</tr>
<tr>
<td>Application definition, updating</td>
<td>39-42</td>
</tr>
<tr>
<td>Main page, code-behind, main page logic, updating</td>
<td>37-38</td>
</tr>
<tr>
<td>HTML, composing with XAML</td>
<td>405-406</td>
</tr>
<tr>
<td>HTTP requests</td>
<td></td>
</tr>
<tr>
<td>Controlling caching</td>
<td>568-569</td>
</tr>
<tr>
<td>GET requests</td>
<td>565-567</td>
</tr>
<tr>
<td>HTTP filters</td>
<td>567</td>
</tr>
<tr>
<td>HTTP prefetching</td>
<td>569</td>
</tr>
<tr>
<td>Manipulating cookies</td>
<td>568</td>
</tr>
<tr>
<td>Hub control</td>
<td>391-397</td>
</tr>
<tr>
<td>HubSections</td>
<td>394-397</td>
</tr>
<tr>
<td>HyperlinkButton control</td>
<td>214</td>
</tr>
<tr>
<td>IconElement</td>
<td>208</td>
</tr>
<tr>
<td>Icons (AppBarButton control)</td>
<td>208-214</td>
</tr>
<tr>
<td>BitmapIcon</td>
<td>212-214</td>
</tr>
<tr>
<td>PathIcon</td>
<td>211-212</td>
</tr>
<tr>
<td>SymbolIcon</td>
<td>208-209</td>
</tr>
<tr>
<td>Identity capabilities</td>
<td>17</td>
</tr>
<tr>
<td>Image element (XAML)</td>
<td>294-302</td>
</tr>
<tr>
<td>Artifacts</td>
<td>302</td>
</tr>
<tr>
<td>Leveraging resource packages</td>
<td>307</td>
</tr>
<tr>
<td>Loading file variations</td>
<td></td>
</tr>
<tr>
<td>Automatic loading</td>
<td>303-306</td>
</tr>
<tr>
<td>Manual loading</td>
<td>306-307</td>
</tr>
<tr>
<td>Nine-grid, custom stretching</td>
<td>296-299</td>
</tr>
<tr>
<td>Referencing files with URIs</td>
<td>294-296</td>
</tr>
<tr>
<td>RenderTargetBitmap class</td>
<td></td>
</tr>
<tr>
<td>Generating dynamic images</td>
<td>302</td>
</tr>
<tr>
<td>Scaling</td>
<td>302-303</td>
</tr>
<tr>
<td>WriteableBitmap class</td>
<td></td>
</tr>
<tr>
<td>Generating dynamic images</td>
<td>299-302</td>
</tr>
</tbody>
</table>

How can we make this index more useful? Email us at indexes@samspublishing.com
image scanners, 666-670
   configuring scan sources, 669
   ImageScanner instance, obtaining, 667
   performing scans, 667-668
   previews, displaying, 669-670
image-based large tile templates, 705-706
ImageBrush, 435-438
ImageProperties, reading metadata, 314-315
images
   decoding, 308-317
      getting pixel data, 310-313
      reading metadata, 313-317
   encoding, 318-325
      writing metadata, 321-322
      writing pixel data, 319-321
   logos, customizing for apps, 10-13
   manipulating, 135-138
   PDF pages, rendering, 326-328
   promotional app images, 20
   tile images, specifying background color, 12-13
   transcoding, 322-325
ImageScanner instance, obtaining, 667
implicit styles, 499-500
improving screen reading, 49-50
in-app purchases, 744-753
   consumable products, 748-753
   durable products, 746-748
inclinometer, 651
incremental item rendering, 246, 546
independent versus dependent animations, 465-466
inertia, 138
inheritance, styles, 497-499
InkManager class (WinRT)
   recognizing handwriting, 147-150
   rendering handwriting, 142-148
input, 113
   keyboard input, handling, 150-156
   mouse input, handling, 138-141
   pen input, handling, 141-150
   touch input, handling, 114-138
inserting property values in control templates, 505-507
intended audience for this book, 2
interacting with GestureRecognizer, 129-131
interrupt transfers, 685
IsColorFontEnabled property (TextBlock), 264-265
items controls, 233-234
   ComboBox control, 239-241
   FlipView control, 248-250
GridView control, 246-247
ItemsControl control, 234-236
ListBox control, 241-242
ListView control, 242-246
   data virtualization, 246
   incremental item rendering, 246
   reordering items, 245
   ScrollIntoView method, 243
   selection methods, 244-245
MenuFlyout control, 255
panels, 236-239
SemanticZoom, 251-254
ItemsControl control, 234-236
JavaScript, 2
   composing with C#, 406-407
jumping to HubSections, 396-397
key states, retrieving, 152-154
keyboard input, handling, 150-156
   focus, 155-156
   key states, retrieving, 152-154
   keyboard modifiers in pointer events, 154
   TextBox control, 277-284
keyframes, 475-480
  discrete keyframes, 478-480
easing, 480
linear keyframes, 475-477
spline keyframes, 477-478
key/value pairs, adding to dictionaries, 33-34
killing apps, 161-162
  versus terminating, 165
known folders, accessing, 563

landscape screen orientation, discovering, 82-84
languages
  localization, 42-48
  Microsoft Local Language Portal, 48
  Microsoft Translator service, 46
  Multilingual App Toolkit, 45
  Pseudo Language, 45-48
large tile templates, 703
LaunchActivatedEventArgs, 166-167
launching
  apps, 165-169
    customizing launch, 175-177
    LaunchActivatedEventArgs, 166-167
programmatically
  launching apps, 174-177
  reacting to
    PreviousExecutionState, 167-169
Visual Studio apps, 5-7
  layout, 57-58, 79-80
  clipping, 98-100
  panels, 84-97
    Canvas, 84-86
    Grid, 87-94
    StackPanel, 86-87
  sizing
    absolute sizing, 92
    autosizing, 92
    proportional sizing, 92
  window orientation,
    discovering, 82-84
  window size, discovering, 80-81
LayoutUpdated event
  handler, 80
leveraging resource packages, 307
libraries, 563-564
lifecycle of apps, 159-161
  activating apps, 169-171
  killing apps, 161-162
  launching apps, 165-169
    customizing launch, 175-177
    LaunchActivatedEventArgs, 166-167
programmatically
  launching apps, 174-177
  reacting to
    PreviousExecutionState, 167-169
resuming apps, 164-165
session state
  managing, 171-174
  representing, 171-172
suspending apps
  deferrals, 164
  OnSuspending event handler, 163-164
suspending apps, 162-164
terminating apps, 165
light sensor, 652
linear keyframes, 475-477
LinearGradientBrush, 429-435
LineHeight property
  (TextBlock), 261
links
  handling with SettingsPane, 605-608
  showing, 604
ListBox control, 241-242
listings, 19-20
ListView control, 242-246
  data virtualization, 246
  high-performance rendering, 545-549
  incremental item rendering, 246
  reordering items, 245
  ScrollIntoView method, 243
  selection methods, 244-245
live regions, 50
live tiles
badges, 708-710
secondary tiles, 710-713
tile templates, 693-706
image-based large templates, 705-706
large templates, 703
medium tile templates, 696
peek medium templates, 697-700
peek wide templates, 700
static image-based wide templates, 700
static medium templates, 696
static text-only wide templates, 700
text-only large templates, 703-705
wide templates, 700
updating, 706-708
local updates, 707
pull updates, 707-708
push notifications, 708
scheduled updates, 707
loading file variations
automatic loading, 303-306
manual loading, 306-307
local content suggestions (SearchBox), 379
local files, 557
local settings, 554-555
localization, 42-48
Microsoft Translator service, 46
Pseudo Language, 45-48
Location capability, 653-660
geofencing, 657-660
getting current location, 653-656
lock screen, 726-728
logic for Hour1 project, 37-38
logo images, customizing for apps, 10-13
magnetic stripe readers, 674-676
managing session state, 171-174
manifest, 8-18
Application tab, 8
Capabilities tab, 13-17
device capabilities, 15-16
file capabilities, 15
identity capabilities, 17
network capabilities, 16
Content URIs tab, 17
Declarations tab, 17
File Open Picker declaration, 622-626
File Type Associations declarations, 619-620
Packaging tab, 17-18
Visual Assets tab, 9-13
logo images, customizing, 10-13
manipulation events, inertia, 138
ManipulationCompleted event, 134
ManipulationDelta event, 134
ManipulationInertiaStarting event, 134
manipulations, 133-138
ManipulationStarted event, 134
manipulationStarting event, 134
manual animations, 484-486
Margin property, 59-60
markers (media), 339-340
markup extensions, 29-31
MatrixTransform, 70
media, 335
audio
background audio, 636-640
capturing, 358-359
capturing, 347-359
photos, capturing, 348-350, 354-355
playing, 336-347
markers, 339-340
media content, 337-338
states, 339
playing media, custom formats, 345-347
speech synthesis, 364-366
SSML, 365-366
transcoding, 359-364
   adding effects, 364
changing format, 362-363
changing quality, 359-362
trimming files, 363-364
video, capturing, 350-351, 357-358
media extensions, 366-367
Media Foundation components, 366-367
media players
   MediaElement as, 341-343
   MediaPlayer, 343-344
MediaCapture, adjusting camera settings, 355-357
MediaElement, 341-343
   states, 339
MediaPlayer, 343-344
medium tile templates, 696
MenuFlyout control, 255
MessageDialog control, 386-387
metadata
   reading, 313-317
      BitmapProperties, 315-316
      ImageProperties, 314-315
      raw metadata, 317
   writing, 321-322
methods
   for Application class, 169-170
   asynchronous methods, 182-184
awaiting an asynchronous operation, 182-184
CreateWindowAsync method, 182-184
GetAsyncKeyState method, 152-154
GetIntermediatePoints method, 118
GetKeyState method, 152-154
GetPosition method, 118
OnWindowCreated method, 170
   Pointer method, 118
   ScrollIntoView method, 243
      for ScrollViewer, 102-104
Microsoft Local Language Portal, 48
Microsoft Translator service, 46
   improving screen reading, 49-50
navigating
   between Pages, 190-197
   NavigationHelper class, 194-195
   WebView control, 403-405
NavigationHelper class, 194-195
.NET classes, using in XAML, 34
network capabilities, 16
networks, fetching data over, 564-575
   background transfers, 569-572
   HTTP requests, 565-569
      controlling caching, 568-569
      GET requests, 565-567
      HTTP filters, 567
      HTTP prefetching, 569
         manipulating cookies, 568
      sockets, 572
      syndication, 572-574
NFC (Near Field Communication) chip, 660-663
   communicating with peer device, 662-663
      sending and receiving messages, 661-662

How can we make this index more useful? Email us at indexes@samspublishing.com
nine-grid, custom stretching, 296-299
not running state, transitioning to running state, 165

Object class, 53
object elements, child object elements
  collection items, 32-34
  content property, 31-32
OnSuspending event handler, 163-164
OnWindowCreated method, 170
open curly braces ({}), escaping, 30
OpticalMarginAlignment property (TextBlock), 263
orientation
  OrientationSensor, 653
  SimpleOrientationSensor, 652-653
  of windows, discovering, 82-84
OrientationSensor, 653

P
package manifest, 8-18
  Application tab, 8
  Capabilities tab, 13-17
    device capabilities, 15-16
    file capabilities, 15
  identity capabilities, 17
  network capabilities, 16
  Content URIs tab, 17
  Declarations tab, 17
  Packaging tab, 17-18
  Visual Assets tab, 9-13
    logo images, customizing, 10-13
    splash screen, customizing, 9
  packaged files, 557
  Packaging tab (package manifest), 17-18
  Padding property (FrameworkElement), 59-60
Page class, 53
  pages, adding ads, 736-738
Pages
  caching, 193-194
  navigating between, 190-197
    NavigationHelper class, 194-195
  PaneThemeTransition, 453-454
  parent elements, 57.
  See also panels
  PasswordBox control, 287-289
  PathFigures, 420-422
  PathIcon, 211-212
  Paths, 416
  PathSegments, 420-422
  PDF pages, rendering, 326-328
  peek medium tile templates, 697-700
  peek wide tile templates, 700
  pen input, handling, 141-150
    digitizers, 141
    handwriting
      recognizing, 147-150
      rendering, 142-148
    palm rejection, 141
  PointerPointProperties object, pen-specific properties, 141-142
performance
  cached composition, 443-444
  Windows Performance Toolkit, 7
  Performance and Diagnostics page (Visual Studio), 7
  performing scans, 667-668
permissions. See capabilities
perspective transforms, 71-73
photos
  camera settings, adjusting, 355-357
  capturing, 348-350, 354-355
  previewing, 352-354
pixels
  retrieving data, 310-313
  scale factor, 84
  writing data, 319-321
plane projections, 71-73
Play To feature (Devices charm), 601-603
playing media, 336-347
  adding effects, 340-341
  custom formats, 345-347
  customizing playback, 338-339
  events, 339
  markers, 339-340
  media content, 337-338
  MediaElement, 341-343
  MediaPlayer, 343-344
  states, 339
Pointer class, 115
pointer events, 116-119
  keyboard modifiers, 154
Pointer method, 118
PointerCanceled event, 117
PointerCaptureLost event, 117
PointerDevice class, 114-115
PointerDownThemeAnimation, 462
PointerEntered event, 117
PointerExited event, 117
PointerPoint class, 115-116
  properties, 116
PointerPointProperties object, 139-140
  pen-specific properties, 141-142
PointerPressed event, 117
  PreviousExecutionState, reacting to, 167-169
primary XML namespaces, 27
printing with Devices charm, 593-601
  adding custom options, 600-601
  changing default options, 599
  configuring displayed options, 599-600
programmatically launching apps, 174-177
  for files, 174
  for URIs, 175
programming languages,
  Windows Store app options, 1-2
ProgressBar control, 371-372
ProgressRing control, 400
projects
  Hour1 project
    main page logic, 37-38
    updating, 35-42
  Visual Studio, 7
    package manifest, 8-18
promotional app images, 20
properties
  AppBarButton control, 207
  attached properties, 85-86
  data binding, 527-528
  dependency properties, 511
  hijacking, 511
  FlowDirection property
    (FrameworkElement), 63
  hit testing, 123-124
  multiple pointers, tracking, 124-127
  PointerPointProperties object, 139-140
  PointerUpThemeAnimation, 462
  PointerWheelChanged event, 139-140
  Polygons, 415-416
  Polylines, 414-415
  PopInThemeAnimation, 462
  PopOutThemeAnimation, 462
  popup controls, 381-391
    CoreWindowDialog control, 381-385
    CoreWindowFlyout control, 385-386
    MessageDialog control, 386-387
    Popup control, 389-391
    PopupMenu control, 387-389
    PopupMenu control, 387-389
    PopupThemeTransition, 452
    portrait screen orientation, discovering, 82-84
    positioning elements,
      HorizontalAlignment property
        (FrameworkElement), 60-61
    power easing functions, 481
    preventing content overflow with clipping, 98-100
    previewing
      photos and video, 352-354
      scanned images, 669-670
  PreviousExecutionState, reacting to, 167-169
  primary XML namespaces, 27
  printing with Devices charm, 593-601
  adding custom options, 600-601
  changing default options, 599
  configuring displayed options, 599-600
  programmatically launching apps, 174-177
  for files, 174
  for URIs, 175
  programming languages,
    Windows Store app options, 1-2
  ProgressBar control, 371-372
  ProgressRing control, 400
  projects
    Hour1 project
      main page logic, 37-38
      updating, 35-42
    Visual Studio, 7
      package manifest, 8-18
  promotional app images, 20
  properties
    AppBarButton control, 207
    attached properties, 85-86
    data binding, 527-528
    dependency properties, 511
    hijacking, 511
    FlowDirection property
      (FrameworkElement), 63
How can we make this index more useful? Email us at indexes@samspublishing.com
Height property (FrameworkElement), 58-59
HorizontalAlignment property (FrameworkElement), 60-61
of LaunchActivatedEventArgs, 166-167
Margin property (FrameworkElement), 59-60
Padding property (FrameworkElement), 59-60
PointerPoint class, 116
PointerPointProperties object, 139-140
pen-specific properties, 141-142
property element syntax, 28
RenderTransform property, 64
RequestedTheme property (FrameworkElement), 202
ScrollViewer property, 101-104
Stroke property (Shapes), 417-419
TextBlock control, 260-261
VariableSizedWrapGrid panel, 96-97
VerticalAlignment property (FrameworkElement), 60-61
Width property (FrameworkElement), 58-59
Window.Current.Bounds property, 80
property elements (XAML), 27-28
property paths, 473-475
proportional sizing, 92
Protocol contract, 621-622
proximity, detecting with NFC chip, 660-663
communicating with NFC peer device, 662-663
sending and receiving messages, 661-662
Pseudo Language, 45-48
pubCenter, defining ad units, 734-739
pull updates (live tiles), 707-708
Push Notification Wizard, 719-720
push notifications, 708
configuring, 718-726
client-side code, 724-726
Push Notification Wizard, 719-720
server-side code, 720-724
range controls, 371-374
ProgressBar control, 371-372
Slider control, 372-374
raw metadata, reading, 317
reacting to PreviousExecutionState, 167-169
reading
metadata, 313-317
BitmapProperties, 315-316
ImageProperties, 314-315
raw metadata, 317
from sensors
accelerometer, 647-651
compass, 651-652
gyrometer, 651
inclinometer, 651
light sensor, 652
OrientationSensor, 653
SimpleOrientationSensor, 652-653
receipts, validating, 753-755
recognizing
gestures, 128-131
handwriting, 147-150
keyboard input, 150-156
manipulations, 133-138
mouse input, 138-141
pen input, 141-150
Rectangles, 412-413
registering this book, 3
Remote Machine options, launching Visual Studio apps with, 6
quality of media, changing, 359-362
query suggestions (SearchBox), 377
RadioButton control, 217-219
Q-R
Rectangles, 412-413
Registering this book, 3
Remote Machine options, launching Visual Studio apps with, 6
rendering
controlling with data binding, 533-541
data templates, 534-537
value converters, 537-541
handwriting, 142-148
high-performance rendering, 545-549
nine-grid, 296-299
page rendering, customizing, 328-331
PDF pages, 326-328
text rendering, customizing, 261-265
RenderTargetBitmap class, generating dynamic images, 302
RenderTransform property, 64
RenderTransformOrigin property, 64
reordering items with ListView control, 245
ReorderThemeTransition, 458-459
RepeatBehavior property (Timeline), 470-472
Repeater control, 215
RepositionThemeAnimation, 462
RepositionThemeTransition, 457-459
representing
Geometries as strings, 425-427
session state, 171-172
RequestedTheme property (FrameworkElement), 202
requirements
for keyboard input focus, 155-156
for Windows Store certification, 19-20
resolution, scale factor, 84
resource lookup, 502-503
resource packages, leveraging, 307
responding to visual state changes, 513-516
restoring session state with SuspensionManager, 172-174
restricting capabilities, 14-15
result suggestions (SearchBox), 380
resuming apps, 164-165
retrieving
key states, 152-154
network connection info, 575
pixel data, 310-313
RichTextBox control, 284-287
RichTextBlock control, 272-277
text overflow, 274-277
UIElements, embedding, 273-274
RightTapped event, 128
roaming files, 558-559
roaming settings, 555-556
RotateTransform, 65
routed events, 116-119
handlers, 118
for keyboard input, 150-151
mouse-specific events, 140-141
rows
arranging children objects with Grid panel, 87-94
sizing, 92-94
running apps in background, 161
S
saving session state with SuspensionManager, 172-174
scale factor, 84
scale sizes of logos, 10-11
ScaleTransform, 66
scaling
handling content overflow, 105-109
interactive zooming, 109
Viewbox, 106-108
scanners, 666-670
scheduled live tile updates, 707
screen orientation, discovering, 82-84
screen reading, improving, 49-50
screenshots, promotional app images, 20
scrolling, 99-105
snap points, 104-105
ScrollIntoView method, 243
ScrollViewer
methods, 102-104
properties, 101-104

How can we make this index more useful? Email us at indexes@samspublishing.com
Search charm, 580-585
  customizing the Search pane, 581-584
  showing Search pane programmatically, 585
  Windows 8 Search contract, 580-581
Search pane
  customizing, 581-584
  showing programmatically, 585
SearchBox control, 374-380
  history suggestions, 375-376
  local content suggestions, 379
  query suggestions, 377
  result suggestions, 380
secondary tiles, 710-713
selecting
  minimum window width, 81-82
  text, 270-271, 280-284
Selectors, 233-234
  ComboBox control, 239-241
  FlipView control, 248-250
  GridView control, 246-247
  ListBox control, 241-242
  ListView control, 242-246
SemanticZoom, 251-254
sensors, 647
  accelerometer, reading from, 647-651
  shaking, 650
  tossing motion, 649-650
  compass, 651-652
gyrometer, 651
inclinometer, 651
light sensor, 652
NFC chip, 660-663
  communicating with peer device, 662-663
  sending and receiving messages, 661-662
OrientationSensor, 653
session state
  managing, 171-174
  representing, 171-172
  restoring with SuspensionManager, 172-174
setting up push notifications, 718-726
  client-side code, 724-726
  Push Notification Wizard, 719-720
  server-side code, 720-724
Settings charm, 603-608
  handling links with SettingsPane, 605-608
  showing links, 604
Shapes
  Ellipses, 413
  Paths, 416
  Polygons, 415-416
  Polylines, 414-415
shapes, Rectangles, 412-413
Shapes, Stroke property, 417-419
Share charm, 585-592
  share sources, 585-589
  share targets, 589-592
share sources (Share charm), 585-589
sharing data binding source with DataContext, 529-530
showing toast notifications, 716-717
shrinking elements,
  ScaleTransform, 66
signing up for Windows Store developer account, 5
SimpleOrientationSensor, 652-653
simulator (Visual Studio), 7
Simulator option, launching Visual Studio apps with, 6
SineEase function, 483
sizes of logos, 10-11
sizing
  absolute sizing, 92
  autosizing, 92
  columns, 92-94
  discovering window size, 80-81
  Height properties, 58-59
  Margin property, 59-60
  proportional sizing, 92
  rows, 92-94
  Width properties, 58-59
SkewTransform, 67-68
slanting elements,
  SkewTransform, 67-68
Slider control, 372-374
snap points, 104-105
  zoom snap points, 106-108
sockets, 572
software keyboard input, handling, 150-156
TextBox control, 277-284
SolidColorBrush, 428-429
source code examples in this book, downloading, 3
speech synthesis, 364-366
  SSML, 365-366
SpeedRatio property (Timeline), 470
spelling prediction, TextBox control, 279-280
splash screen, customizing, extended splash screens, 42
splash screen, customizing for apps, 9
spline keyframes, 477-478
SplitCloseThemeAnimation, 463
SplitOpenThemeAnimation, 463
SSML (Speech Synthesis Markup Language), 365-366
StackPanel panel, 86-87
Start screen, semantic zooming, 251-254
states of media, 339
static image-based wide tile templates, 700
static medium tile templates, 696
static text-only wide tile templates, 700
stopping event bubbling, 119-120
storage objects
  files, 551-553
  folders, 552
    known folders, accessing, 563
Storyboards, 460-461
  with multiple animations, 472-473
stretching images with nine-grid, 296-299
strings
  Geometries represented as, 425-427
  Pseudo Language strings, 46
  type converted strings, 34
  type converters, 28-29
Stroke property, 417-419
styles, 494-503
  base TargetType, 497
  implicit styles, 499-500
  inheritance, 497-499
  resource lookup, 502-503
  theme resources, 500-502
styles. See pen input, handling
submitting apps to Windows Store, 18-20
suggestions, providing to SearchBox users
  history suggestions, 375-376
  local content suggestions, 379
  query suggestions, 377
  result suggestions, 380
supporting
  full license purchases, 742-744
  scale sizes of logos, 11-12
suspended apps, 159
  conditions for, 163
  deferrals, 164
SuspensionManager class, 172-174
SwipeBackThemeAnimation, 462
SwipeHintThemeAnimation, 462
SymbolIcon, 208-209
syndication, 572-574
syntax
  markup extensions, 29-31
  property elements, 28

T
tap and send, 660
Tapped event, 128
templates, 503-512
  control templates, 503-505
  dependency properties, hijacking, 511
  inserting property values, 505-507
  data templates, 534-537
  tile templates, 693-706
  image-based large templates, 705-706
  large templates, 703
  medium tile templates, 696
  peek medium templates, 697-700
  peek wide templates, 700
  static image-based wide templates, 700
  static medium templates, 696
templates

static text-only wide templates, 700
text-only large templates, 703-705
wide templates, 700
toast templates, 713-716
terminating apps, 165
testing Windows Store features, 755-760
automated testing, 759-760
CurrentAppSimulator, 760

text controls, 259
PasswordBox control, 287-289
RichEditBox control, 284-287
RichTextBlock control, 272-277
text overflow, 274-277
UIElements, embedding, 273-274
TextBlock control, 259-271
customizing text rendering, 261-265
properties, 260-261
text content, 265-267
text elements, 267-270
text selection, 270-271
Textbox control, 277-284
spelling prediction, 279-280
text prediction, 279-280
text selection, 280-284
TextLineBounds property (TextBlock), 262-263
text-only large tile templates, 703-705
TextReadingOrder property (TextBlock), 264
text-to-speech. See speech synthesis
theme animations, 459-464
Storyboards, 460-461
tweaking, 463-464
theme resources, 500-502
theme transitions, 448-459
AddDeleteThemeTransition, 455-456
applying to elements, 448-449
ContentThemeTransition, 452-453
EdgeUIThemeTransition, 453
EntranceThemeTransition, 450-451
PaneThemeTransition, 453-454
PopupThemeTransition, 452
ReorderThemeTransition, 458-459
RepositionThemeTransition, 457-459
ThemeResource, 501
themes
app themes, 202
dark themes, 201
high contrast, handling, 51-53
user themes, 202
Thickness class, 60
threads
background threads, 181-182
transitioning between, 185
UI threads, 181-182
awaiting an asynchronous operation, 182-184
tile images
background color, specifying, 12-13
customizing for apps, 10-13
tile templates, 693-706
image-based large templates, 705-706
large templates, 703
medium templates, 696
peek medium templates, 697-700
peek wide templates, 700
static image-based wide templates, 700
static medium templates, 696
static text-only wide templates, 700
text-only large templates, 703-705
wide templates, 700
time-based trials, 740
Timeline properties, 469-472
TimePicker control, 399
toast notifications, 713-717
    options for showing, 716-717
    responding to, 716
toast templates, 713-716
toggle templates, 713-716
toggleButton control, 215-216
toggleSwitch control, 400-402	
toolTip control, 219-222
tossing motion, detecting, 649-650
touch input, handling, 114-138
gestures, 127-133
    EdgeGesture, 132-133
    GestureRecognizer class (WinRT), 128-131
    UIElement gesture events, 133
manipulations, 133-138
    inertia, 138
pointers, 114-127
    hit testing, 123-124
    multiple pointers, 118-119
    multiple pointers, tracking, 124-127
    Pointer class, 115
    pointer events, 116-119
    PointerDevice class, 114-115
    PointerPoint class, 115-116
    PointerPointProperties object, 139-140
    Pointers, capturing, 120-122
    tracking multiple pointers, 124-127
transcoding
    adding effects, 364
    media, 359-364
    changing format, 362-363
    changing quality, 359-362
    images, 322-325
    trimming, 363-364
TransformGroup, 69-70
transforms
    2D transforms
        RotateTransform, 65
        ScaleTransform, 66
        SkewTransform, 67-68
        TranslateTransform, 67-68
    3D transforms, 70-73
    combining
        CompositeTransform, 68-69
        MatrixTransform, 70
        TransformGroup, 69-70
    perspective transforms, 71-73
transitioning
    from not running to running state, 165
    between threads, 185
TranslateTransform, 67-68
translating Microsoft Local Language Portal, 48
trimming, 98
trimming media files, 363-364
troubleshooting Visual Studio apps, 7
tweaking theme animations, 463-464
type converters, 28-29
    type converted strings, 34
UI
UI Accessibility Checker, 49
UI elements, layout, 57-58
UI frameworks, Windows Store app options, 1-2
UI threads, 181-182
    awaiting an asynchronous operation, 182-184
    transitioning between, 185
    views, displaying multiple views, 186-189
UIElement class, 53. See also FrameworkElement class
    embedding elements, 273-274
gesture events, 133
    RenderTransform property, 64

How can we make this index more useful? Email us at indexes@samspublishing.com
updating
Hour1 project
  application definition, 39-42
  main page logic, 35-42
live tiles, 706-708
local updates, 707
pull updates, 707-708
push notifications, 708
scheduled updates, 707
URIs
  programmatically launching apps for, 175
  referencing files with, 294-296
USB devices, 682-685
  communicating with, 684-685
  bulk transfers, 684-685
  control transfers, 684
  interrupt transfers, 685
  connecting to, 683-684
  declaring the right capability, 682-683
user data, 559-564
  file picker, 560
  folder picker, 561-562
  known folders, accessing, 563
  libraries, 563-564
user themes, 202
validating receipts, 753-755
value converters, 537-541
VariableSizedWrapGrid panel, 94-97
vector graphics, 411
  Brushes, 427-442
  bitmap-based brushes, 435-442
  color brushes, 428-435
  Geometries, 419-427
  Bézier curves, 420
  FillRule, 423
  GeometryGroup, 423-425
  PathFigures, 420-422
  PathSegments, 420-422
  representing as strings, 425-427
  Shapes
    Ellipses, 413
    Paths, 416
    Polygons, 415-416
    Polylines, 414-415
    Rectangle, 412-413
    Stroke property, 417-419
  VerticalAlignment property (FrameworkElement), 60-61
  VerticalContentAlignment property (Control class), 61-62
video
  capturing, 350-351, 357-358
  custom media formats, 345-347
  customizing playback, 338-339
  MediaElement, 341-343
  MediaPlayer, 343-344
  playing
    adding effects, 340-341
    markers, 339-340
    media content, 337-338
    previewing, 352-354
  video stabilization, 340-341
Viewbox, 106-108
views
  of collections, customizing, 541-545
  grouping, 542-545
  navigating, 545
  GridView, 545-549
  ListView, 545-549
  multiple, displaying, 186-189
Visual Assets tab (package manifest), 9-13
  logo images, customizing, 10-13
  splash screen, customizing, 9
Visual State Manager, 512-521
  responding to changes, 513-516
  visual transitions, 516-521
visual states, 512-521
  responding to changes, 513-516
  visual transitions, 516-521
Visual Studio
apps
launching, 5-7
submitting to Windows Store, 18-20
troubleshooting, 7
Performance and Diagnostics page, 7
projects, 7
package manifest, 8-18
simulator, 7
website, 5, 20
visual transitions, 516-521
VisualStateManager, 512-521
responding to changes, 513-516
visual transitions, 516-521
VSM, 512-521
responding to changes, 513-516
visual transitions, 516-521

W
web content, capturing with WebView control, 407-408
websites
PAINT.NET, 3
Visual Studio, 5, 20
Windows Dev Center, 3
WebView control, 402-408
capturing web content, 407-408
composing HTML with XAML, 405-406
composing JavaScript with C#, 406-407
navigation, 403-405
WebViewBrush, 437-442
wide tile templates, 700
width, selecting minimum
window width, 81-82
Width properties, 58-59
Wi-Fi devices, 688-690
Window.Current.Bounds property, 80
windows
borders, applying, 195-197
Frames, 195-197
minimum width, selecting, 81-82
multiple windows,
displaying, 186-189
orientation, discovering, 82-84
Pages, navigating between, 190-197
scale factor, 84
size, discovering, 80-81
zooming in/out, 106-108
Windows 8 Search contract, 580-581
Windows Dev Center website, 3
Windows Media, 366-367
Windows Performance Toolkit, 7
Windows Store
apps
programming language/
UI framework pairing
options, 1-2
submitting, 18-20
certification process, 19-20
developer account, signing up for, 5
features, testing, 755-760
automated testing, 759-760
CurrentAppSimulator, 756-759
in-app purchases, 744-753
consumable products, 748-753
durable products, 746-748
receipts, validating, 753-755
DisplayProperties.
CurrentOrientation
property, 83
Windows.System.Launcher
class, 174-177
WinRT
APIs, 2
asynchronous methods,
182-184
displaying multiple views,
186-189
GestureRecognizer class, 128-131
InkManager class, 142-148
wrapping elements,
VariableSizedWrapGrid panel,
94-97
WriteableBitmap class,
generating dynamic images,
299-302
writing
metadata, 321-322
pixel data, 319-321

How can we make this index more useful? Email us at indexes@samspublishing.com
XAML, 23-24
attributes, 25
base TargetType, 497
child object elements
  collection items, 32-34
  content property, 31-32
  type converted values, 34
combining with C#, 35
comparing with C#, 25
composing HTML, 405-406
elements, 25
Hour1 project
  application definition, 39-42
  updating, 35-42
markup extensions, 29-31
motivation for, 23-24
.NET classes, 34
parent elements, 57
property elements, 27-28
styles, 494-503
  implicit styles, 499-500
  inheritance, 497-499
  resource lookup, 502-503
  theme resources, 500-502
type converters, 28-29
XML namespaces, 25-27

XAML UI Responsiveness tool
(Visual Studio), 7
XML namespaces, 25-27

Y-Z
Z order, 86
zooming in/out windows, 106-108
  semantic zooming, 251-254