Windows® 8.1 Apps
with XAML and C#
# Contents at a Glance

<table>
<thead>
<tr>
<th>Part</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Part I Getting Started</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Hello, <em>Real World!</em></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2 Mastering XAML</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td><strong>Part II Building an App</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Sizing, Positioning, and Transforming Elements</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>4 Layout</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>5 Interactivity</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>6 Handling Input: Touch, Mouse, Pen, and Keyboard</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td><strong>Part III Working with the App Model</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 App Lifecycle</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>8 Threading, Windows, and Pages</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>9 The Many Ways to Earn Money</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td><strong>Part IV Understanding Controls</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 Content Controls</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>11 Items Controls</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>12 Text</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>13 Images</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>14 Audio, Video, and Speech</td>
<td>367</td>
</tr>
<tr>
<td></td>
<td>15 Other Controls</td>
<td>401</td>
</tr>
</tbody>
</table>
Part V  Leveraging the Richness of XAML

16  Vector Graphics  437
17  Animation  469
18  Styles, Templates, and Visual States  513
19  Data Binding  545

Part VI  Exploiting Windows 8.1

20  Working with Data  571
21  Supporting Charms  597
22  Leveraging Contracts  627
23  Reading from Sensors  661
24  Controlling Devices  679
25  Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen  703

Part VII  Advanced Features

26  Integrating DirectX  739
27  Custom Controls and Components  755
28  Layout with Custom Panels  781

Index  799
# Table of Contents

## Part I: Getting Started

1. Hello, *Real* World! 9  
   - Creating, Deploying, and Profiling an App 9  
   - Understanding the App Package 12  
   - Updating XAML and C# Code 22  
   - Making the App World-Ready 30  
   - Making the App Accessible 36  
   - Submitting to the Windows Store 40  
   - Summary 42  

2. Mastering XAML 43  
   - Elements and Attributes 44  
   - Namespaces 45  
   - Property Elements 47  
   - Type Converters 48  
   - Markup Extensions 49  
   - Children of Object Elements 51  
   - Mixing XAML with Procedural Code 56  
   - XAML Keywords 59  
   - Summary 61  

## Part II: Building an App

3. Sizing, Positioning, and Transforming Elements 63  
   - Controlling Size 64  
   - Controlling Position 68  
   - Applying 2D Transforms 71  
   - Applying 3D Transforms 79  
   - Summary 82  

4. Layout 83  
   - Discovering Your Window Size and Location 84  
   - Panels 89  
   - Handling Content Overflow 105  
   - Summary 116  

5. Interactivity 117  
   - Dependency Properties 117  
   - Routed Events 124  
   - Commands 129  
   - Summary 131  

6. Handling Input: Touch, Mouse, Pen, and Keyboard 133  
   - Touch Input 134  
   - Mouse Input 156  
   - Pen Input 159  
   - Keyboard Input 168  
   - Summary 174  

## Part III: Working with the App Model

7. App Lifecycle 175  
   - Killing 177  
   - Suspending 178  
   - Resuming 181  
   - Terminating 181  
   - Launching 182  
   - Activating 185  
   - Managing Session State with SuspensionManager 187  
   - Programmatically Launching Apps 190  
   - Summary 193
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Threading, Windows, and Pages</td>
<td>195</td>
</tr>
<tr>
<td>9</td>
<td>The Many Ways to Earn Money</td>
<td>213</td>
</tr>
<tr>
<td>10</td>
<td>Content Controls</td>
<td>241</td>
</tr>
<tr>
<td>11</td>
<td>Items Controls</td>
<td>271</td>
</tr>
<tr>
<td>12</td>
<td>Text</td>
<td>295</td>
</tr>
<tr>
<td>13</td>
<td>Images</td>
<td>327</td>
</tr>
<tr>
<td>14</td>
<td>Audio, Video, and Speech</td>
<td>367</td>
</tr>
<tr>
<td>15</td>
<td>Other Controls</td>
<td>401</td>
</tr>
</tbody>
</table>

- **Understanding the Threading Model for Windows Store Apps**: 195
- **Displaying Multiple Windows**: 200
- **Navigating Between Pages**: 204
- **Summary**: 211
- **Adding Advertisements to Your App**: 214
- **Supporting a Free Trial**: 220
- **Supporting In-App Purchases**: 224
- **Validating Windows Store Receipts**: 232
- **Testing Windows Store Features**: 235
- **Summary**: 240
- **Button**: 244
- **AppBarButton**: 247
- **HyperlinkButton**: 254
- **RepeatButton**: 256
- **ToggleButton**: 256
- **AppBarToggleButton**: 256
- **CheckBox**: 257
- **RadioButton**: 258
- **ToolTip**: 259
- **App Bars**: 262
- **Summary**: 269
- **ComboBox**: 279
- **ListBox**: 281
- **ListView**: 285
- **GridView**: 287
- **FlipView**: 289
- **SemanticZoom**: 292
- **MenuFlyout**: 293
- **TextBlock**: 295
- **RichTextBlock**: 308
- **TextBox**: 313
- **RichEditBox**: 321
- **PasswordBox**: 324
- **Summary**: 326
- **The Image Element**: 328
- **Multiple Files for Multiple Environments**: 337
- **Decoding Images**: 342
- **Encoding Images**: 351
- **Rendering PDF Content as an Image**: 359
- **Summary**: 365
- **Playback**: 368
- **Capture**: 380
- **Transcoding**: 392
- **Speech Synthesis**: 397
- **Summary**: 399
- **Range Controls**: 401
- **SearchBox**: 404
- **Popup Controls**: 411
- **Hub**: 421
- **Date and Time Controls**: 426
### Part VII Advanced Features

26 Integrating DirectX 739
   - Integrating as an Image Source 740
   - Integrating the Swap Chain 747
   - Summary 752

27 Custom Controls and Components 755
   - Creating a User Control 756
   - Creating a More Complex Control 759
   - Creating a Templated Control 771
   - Creating a Windows Runtime Component 776
   - Summary 779

28 Layout with Custom Panels 781
   - Communication Between Parents and Children 782
   - Creating a SimpleCanvas 785
   - Creating a SimpleStackPanel 789
   - Creating a UniformGrid 792
   - Summary 798

Index 799
About the Author

Adam Nathan is a principal software architect for Microsoft, a best-selling technical author, and arguably the world’s most prolific developer for Windows Phone. He introduced XAML to countless developers through his books on a variety of Microsoft technologies. Currently a part of Microsoft’s Startup Business Group, 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 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.
This page intentionally left blank
Dedication

To Mom and Dad.

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 Ashish Shetty, Tim Heuer, Mark Rideout, Jonathan Russ, Joe Duffy, Chris Brumme, Eric Rudder, 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.

Finally, I thank you for picking up a copy of this book! I don’t think you’ll regret it!
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.

You can email or write us directly 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 stronger.

*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 phone or email address. We will carefully review your comments and share them with the author and editors who worked on the book.

E-mail: 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.
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 new 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, 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 your 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.

Common perceptions of performance differences between the UI frameworks are often wrong, however. It’s important to realize that no matter which of the three UI frameworks you use, about 80% of their core implementation is identical, the Windows APIs are the same, and the graphics are hardware accelerated. Although DirectX offers the most potential for getting the highest performance, you have to do a lot of work to realize that potential! Often, a C#/XAML implementation can outperform a simple C++/DirectX implementation due to the impressive optimizations that the XAML UI Framework does on your behalf. Not only that, but the XAML UI Framework gives you a number of additional features automatically, such as accessibility and localization.

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, and doesn’t have the same multithreading limitations that plague JavaScript. C++ provides the most potential for getting the highest performance. (Does that line sound familiar?) Of course, you have to earn that performance, and you have to be especially careful with how you mix standard C and C++ code with the C++/CX code that is needed to communicate with Windows.

The key to the multiple language support is the Windows Runtime, or WinRT for short. You can think of it like .NET’s Common Language Runtime, except it spans both managed and unmanaged languages. To enable this, WinRT is COM-based. Most of the time, you can’t tell when you interact with WinRT. And most of the time, it doesn’t matter. This is a modern, friendlier version of COM that is more amenable to automatic correct usage from environments such as .NET or JavaScript. (Contrast this to over a decade ago, when I wrote a book about mixing COM with .NET. This topic alone required over 1,600 pages!)
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. General-purpose WinRT APIs follow slightly different conventions and can sometimes look a little odd to developers familiar with .NET. For example, they tend to be exception-heavy for situations that normally don’t warrant an exception (such as the user cancelling an action). Artifacts like this are caused by the projection mechanism mapping HRESULTs (COM error codes) into .NET exceptions.

I wrote this book with the following goals in mind:

- To provide a solid grounding in the underlying concepts, in a practical and approachable fashion
- To answer the questions most people have when learning how to write Windows Store apps and to show how commonly desired tasks are accomplished
- To be an authoritative source, thanks to input from members of the team who designed, implemented, and tested Windows 8.1 and Visual Studio 2013
- To be clear about where the technology falls short rather than blindly singing its praises
- To optimize for concise, easy-to-understand code rather than enforcing architectural patterns that can be impractical or increase the number of concepts to understand
- To be an easily navigated reference that you can constantly come back to
To elaborate on the second-to-last point: You won’t find examples of patterns such as Model-View-ViewModel (MVVM) in this book. I am a fan of applying such patterns to code, but I don’t want to distract from the core lessons in each chapter.

Whether you’re new to XAML or a long-time XAML developer, I hope you find this book to exhibit all these attributes.

**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.

If you are already well versed in XAML, I’m confident that this book still has a lot of helpful information for you. And if you are already familiar with writing Windows Store apps for Windows 8 (perhaps thanks to the first edition of this book), you will still benefit from the significant amount of new content that covers new features in Windows 8.1. It also covers features that were already present in Windows 8 in more depth than ever before. At the very least, this book should be an invaluable reference for your bookshelf.

**Software Requirements**

This book targets Windows 8.1, Windows RT, and the corresponding developer tools. The tools are a free download at the Windows Dev Center: http://dev.windows.com. The download includes the Windows 8.1 SDK, a version of Visual Studio Express specifically for Windows Store apps, and miscellaneous tools. It’s worth noting that although this book almost exclusively refers to Windows 8.1, the content applies to Windows RT as well.

Although it’s not required, I recommend PAINT.NET, a free download at http://getpaint.net, for creating and editing graphics, such as the set of icons needed by apps.

**Code Examples**


**How This Book Is Organized**

This book is arranged into seven parts, representing the progression of feature areas that you typically need to understand. But if you want to jump ahead and learn about a topic such animation or live tiles, the book is set up to allow for nonlinear journeys as well. The following sections provide a summary of each part.
Part I: Getting Started
This part includes the following chapters:

➔ Chapter 1: “Hello, Real World!”
➔ Chapter 2: “Mastering XAML”

Part I provides the foundation for the rest of the book. If you have previously created Windows Phone apps or worked with XAML in the context of other Microsoft technologies, a lot of this should be familiar to you. There are still several unique aspects for Windows 8.1 and the Windows Store, however. Chapter 1 helps you understand all the tools available at your disposal, and even dives into topics such as accessibility and localization, so you can be prepared to get the broadest set of customers possible for your app. This last set of topics is new to this edition of the book.

Part II: Building an App
This part includes the following chapters:

➔ Chapter 3: “Sizing, Positioning, and Transforming Elements”
➔ Chapter 4: “Layout”
➔ Chapter 5: “Interactivity”
➔ Chapter 6: “Handling Input: Touch, Mouse, Pen, and Keyboard”

Part II equips you with the knowledge of how to place things on the screen, how to make them adjust to the wide variety of screen types, and how to interact with the user. Windows 8.1 introduces a new model for how apps should resize, and this is covered in Chapter 4. In Chapter 6, this edition contains new coverage on supporting pens, including rendering strokes and performing handwriting recognition.

Part III: Working with the App Model
This part includes the following chapters:

➔ Chapter 7: “App Lifecycle”
➔ Chapter 8: “Threading, Windows, and Pages“
➔ Chapter 9: “The Many Ways to Earn Money”

The app model for Windows Store apps is significantly different from the app model for desktop applications in a number of ways. It’s important to understand how the app lifecycle works and how you need to interact with it in order to create a well-behaved app. But there are other pieces to what is sometimes called the app model: how one app can launch another, how to work with the Windows Store to enable free trials and in-app purchases, and how to deal with multiple windows and pages. This edition greatly
expands the coverage on trials and in-app purchases, and covers the new Windows 8.1 in-app purchase features. It also contains new coverage on integrating ads into your apps, the threading model for Windows Store apps, and new support for having multiple windows.

**Part IV: Understanding Controls**

This part includes the following chapters:

- Chapter 10: “Content Controls”
- Chapter 11: “Items Controls”
- Chapter 12: “Text”
- Chapter 13: “Images”
- Chapter 14: “Audio, Video, and Speech”
- Chapter 15: “Other Controls”

Part IV provides a tour of the controls built into the XAML UI Framework. There are many controls that you expect to have available, plus several that you might not expect. Windows 8.1 adds many new controls and many features to existing controls. Windows 8.1 also introduces speech synthesis features, which are covered in Chapter 14.

**Part V: Leveraging the Richness of XAML**

This part includes the following chapters:

- Chapter 16: “Vector Graphics”
- Chapter 17: “Animation”
- Chapter 18: “Styles, Templates, and Visual States”
- Chapter 19: “Data Binding”

The features covered in Part V are areas in which XAML really shines. Although previous parts of the book expose some XAML richness (applying transforms to any elements, the composability of controls, and so on), these features push the richness to the next level.

**Part VI: Exploiting Windows 8.1**

This part includes the following chapters:

- Chapter 20: “Working with Data”
- Chapter 21: “Supporting Charms”
- Chapter 22: “Leveraging Contracts”
Chapter 23: “Reading from Sensors”
Chapter 24: “Controlling Devices”
Chapter 25: “Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen”

This part of the book could just as easily appear in a book about JavaScript or C++ Windows Store apps, with the exception of its code snippets. It covers unique and powerful Windows features that are not specific to XAML or C#, but they are things that all Windows Store app developers should know. The most notable new support in Windows 8.1 is covered in Chapter 24: supporting custom devices.

Part VII: Advanced Features

This part includes the following chapters:

- Chapter 26: “Integrating DirectX”
- Chapter 27: “Custom Controls and Components”
- Chapter 28: “Layout with Custom Panels”

The advanced features covered in the last part of the book highlight very different scenarios. Integrating DirectX into your XAML app enables you to do things that aren’t possible otherwise, whereas the last two chapters are about ways to reuse your code. The coverage of all these features is new to this edition. These features all existed in Windows 8, although the DirectX integration support has been improved for Windows 8.1.

Conventions Used in This Book

Various typefaces in this book identify new terms and other special items. These typefaces include the following:

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italic</em></td>
<td>Italic is used for new terms or phrases when they are initially defined and occasionally for emphasis.</td>
</tr>
<tr>
<td>Monospace</td>
<td>Monospace is used for screen messages, code listings, and filenames. In code listings, <em>italic monospace type</em> is used for placeholder text. Code listings are colorized similarly to the way they are colorized in Visual Studio. <em>Blue monospace type</em> is used for XML elements and C# keywords, <em>brown monospace type</em> is used for XML element names and C# strings, <em>green monospace type</em> is used for comments, <em>red monospace type</em> is used for XML attributes, and <em>teal monospace type</em> is used for type names in C#.</td>
</tr>
<tr>
<td><strong>Bold</strong></td>
<td>When appropriate, bold is used for code directly related to the main lesson(s) in a chapter.</td>
</tr>
</tbody>
</table>
Throughout this book, and even in this introduction, you will find a number of sidebar elements:

**What is a FAQ sidebar?**
A Frequently Asked Question (FAQ) sidebar presents a question you might have about the subject matter—and then provides a concise answer.

**Digging Deeper**
A Digging Deeper sidebar presents advanced or more detailed information on a subject than is provided in the surrounding text. Think ofDigging Deeper material as something you can look into if you’re curious but can ignore if you’re not.

**A tip offers information about design guidelines, shortcuts or alternative approaches to produce better results, or something that makes a task easier.**

**This is a warning!**
A warning alerts you to an action or a condition that can lead to an unexpected or unpredictable result—and then tells you how to avoid it.
HELLO, REAL WORLD!

“Oh, no, not another cliché ‘Hello, World’ example,” you might be thinking as you examine this book. However, the length of this chapter alone should tell you that it is not about creating a typical “Hello, World” app.

Sure, we’re going to get started with a simple, contrived app to demonstrate the anatomy of any Windows Store XAML app and the tooling available in Visual Studio. But we’ll also see how to make it really say “hello” to the *entire* world; not just English-speaking people with no disabilities. This means understanding how to localize an app into other languages so you can exploit the vast, global scale of the Windows Store. It also means understanding how to make your app accessible to users who require assistive technologies such as screen readers or high contrast themes. No app deserves to be called “Hello, World” without considering these features.

Creating, Deploying, and Profiling an App

In Visual Studio, let’s create a new Visual C# Blank App (XAML) project called HelloRealWorld. This gives us a project that’s ready to compile and run. Although pressing F5 or clicking the Start Debugging button in Visual Studio launches the app locally, you’ve got three slick options to choose from via the button’s dropdown menu, shown in Figure 1.1 under Visual Studio’s light theme (used throughout this book).
With the **Remote Machine** option, you can deploy and debug to any other Windows 8.x computer reachable on your network (although not over the Internet). This is extremely handy for testing things on a Surface or other tablets. The target device must have the Remote Tools for Visual Studio installed and running, which you can download from the Windows Dev Center.

The **Simulator** option is the next best thing to having a real tablet, as it provides mechanisms to simulate touch input, device orientations, network conditions, location services, and more. The simulator is shown in Figure 1.2. In fact, it has one huge advantage over testing on a physical device: It enables you to experience your app in a number of different resolutions and virtual screen sizes, including different aspect ratios. Given the wide variety of shapes and sizes of screens out there that run Windows Store apps, testing your app in this fashion is a must.

![FIGURE 1.1](image1.png)

**FIGURE 1.1** The three ways to launch your app in Visual Studio

![FIGURE 1.2](image2.png)

**FIGURE 1.2** Testing your app on the simulator is like testing it on an army of different-sized devices.

---

**The simulator is your actual computer!**

Although the simulator simulates several things, what you see on the virtual device is your real “host” computer running with your actual user account, apps, files, and so on. (Running the simulator is like initiating a special kind of remote desktop connection to yourself.) Changes you make inside the simulator affect your computer just as if you made them outside the simulator.
How do I run my app outside of Visual Studio?

Although compiling your app produces an .exe file in the `bin\Debug` or `bin\Release` subfolder, you can’t simply double-click it from the Windows desktop to run it. If you try, you get an error that explains, “This application can only run in the context of an app container.” (An “app container” refers to the sandbox in which all Windows Store apps run.) Instead, you can launch it from the searchable list of apps underneath the tiles on the Start screen. Visual Studio automatically installs your app the first time you launch it. Like all Windows Store apps in Windows 8.1, however, its tile does not automatically get pinned. Because the Start screen has been enhanced to make it easier to find apps, pinning is now meant to be done selectively by a user, the same as with pinning apps to the desktop taskbar.

When you run the HelloRealWorld project without any changes, you’ll see why the project type was called “Blank App.” The app doesn’t actually do anything other than fill the screen with darkness. (If you launch the app in debug mode, you’ll also see four numbers on the top edge of the screen. These are frame rate counters described in Chapter 17, “Animation.”) It does, however, set up a lot of infrastructure that would be difficult and tedious to create from scratch. The project contains the following items:

➔ The package manifest, a temporary certificate used to sign it, and some images
➔ The main page (`MainPage.xaml` and `MainPage.xaml.cs`)
➔ The application definition: `App.xaml`, `App.xaml.cs`, and `AssemblyInfo.cs`

The next section examines the package manifest and the images used by it. After that, we’ll look at the XAML and C# files and make some code changes.

Visual Studio provides some amazing tools for diagnosing performance problems in your app. You can access them by clicking Performance and Diagnostics on the Debug menu. On this page, select a tool to collect data while your app is launched. You perform the scenario you want to measure, and then stop the data collection. A rich, interactive report is then presented to you. The three tools on the Performance and Diagnostics page are:

➔ XAML UI Responsiveness—Attributes the time spent to activities such as parsing XAML and layout of your elements. Shows you the performance cost of each UI element. You can also investigate times when you’re not achieving the desired 60 frames per second on the UI thread.
➔ CPU Sampling—Traditional profiling, with interactive graphs, diagrams of hot paths complete with annotated code integration, and much more.
➔ Energy Consumption—Estimates how power-hungry your app is, based on its usage of the CPU, display, and network.

In addition to the Visual Studio tools, you can download the Windows Performance Toolkit for additional analysis. This includes a Windows Performance Recorder tool for capturing a trace, and a Windows Performance Analyzer tool for analyzing the trace.
Understanding the App Package

The *package manifest* in the Visual Studio project is a file called `Package.appxmanifest`. ("AppX" is a term sometimes used within Microsoft for Windows Store app packages that stuck around in the filename.) This manifest describes your app to Windows as well as the Windows Store—its name, what it looks like, what it's allowed to do, and more. It's an XML file, although you have to tell Visual Studio to "View Source" in order to see the XML. There's usually no need to view and edit the XML directly, however. The default view is a tabbed set of forms to fill out, which is the easiest way to populate all the information. There are six tabs:

- Application
- Visual Assets
- Capabilities
- Declarations
- Content URIs
- Packaging

For our *HelloRealWorld* app, we don’t need to change anything in the package manifest. But now is a good time to understand what can be done on each of these tabs.

**Application**

On the Application tab, you can set the app’s name and description, default language, its minimum width, and notification settings (if your app supports them). Notifications are covered in Chapter 25, “Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen.” You can even restrict the preferred orientations of your app if you’d rather not have it automatically rotate to all four of them:

- **Landscape** (horizontal)
- **Landscape-flipped** (horizontal but upside down)
- **Portrait** (vertical, with the hardware Start button on the left)
- **Portrait-flipped** (vertical, with the hardware Start button on the right)

Disabling the *flipped* orientations would be an odd thing to do, but disabling some orientations can make sense for certain types of games that wish to be landscape only. Note that this is just a *preference*, not a guarantee, because not all devices support rotation. For example, a portrait-only app launched on a typical desktop PC must accept the one-and-only landscape orientation. However, if a device that *does* support rotation is currently locked to a landscape orientation, a portrait-only app actually runs in the portrait orientation, ignoring the lock setting.

**Visual Assets**

On the Visual Assets tab, you set the characteristics of your app’s tile and splash screen, as well as artwork used in a number of other contexts.
Customizing the Splash Screen

To ensure that every app’s splash screen can be displayed practically instantaneously (before your app even gets loaded), you have little control over it. You specify a 620x300 image (plus two optional larger sizes to support high DPI screens), and a background color for the splash screen. That’s it. Visual Studio gives you an appropriately sized placeholder `SplashScreen.scale-100.png` file in an Assets subfolder, intentionally made ugly to practically guarantee you won’t forget to change it before submitting your app to the Windows Store.

When your splash screen is shown, the image is displayed centered on top of your chosen background color. Figure 1.3 shows an example `SplashScreen.scale-100.png` containing a Pixelwinks logo, and Figure 1.4 shows what this looks like on the simulator. The splash screen is given a yellow background for demonstration purposes. A real app should make the background color match the background of the image or simply make the image’s background transparent.

![Image of a splash screen with a Pixelwinks logo](image)

**FIGURE 1.3** An example `SplashScreen.scale-100.png` with a nontransparent background for demonstration purposes

When your app is launched, the splash screen automatically animates in and automatically fades out once your app has loaded and has made a call to `Window.Current.Activate`. This gives you the flexibility to do arbitrarily complex logic before the splash screen goes away, although you should avoid doing a lot of work here. (Your app is given about fifteen seconds to remain on the splash screen before it gets terminated by Windows.)

Customizing Logo Images

The Tile Images and Logos section on the Visual Assets tab can be confusing and overwhelming. Besides the Store Logo, which supports up to three different sizes, it lists five different logo sizes, although each one actually accepts 4–8 different sizes of image files! All told, you can assign twenty seven different image files representing your logo! Let’s start making some sense out of these images. Figure 1.5 shows what each logo should have been called to make things less confusing, and the following list explains each one using the terminology found in the package manifest:
Square 70x70 Logo—This is used for the small version of your app’s tile on the Start screen. Although assigning an image here is optional, the small tile size is not. If you don’t provide an image, the medium tile image is used (and scaled down) when a user changes your tile size to small.

Square 150x150 Logo—This is used for the medium version of your app’s tile on the Start screen. The medium tile size is the one required size, so at least a 100% scale image is required.

Wide 310x150 Logo—This is used for the wide version of your app’s tile on the Start screen, if you choose to support that tile size. If you assign at least a 100% scale image here, your app automatically supports the wide tile size. Otherwise, it doesn’t.

Large 310x310 Logo—This is used for the large version of your app’s tile on the Start screen, if you choose to support that tile size. If you assign at least a 100% scale image here and for the wide logo, your app automatically supports the large tile size. (Your app can only support a large tile if it also supports a wide tile.) Otherwise, it doesn’t.

Square 30x30 Logo—This is used throughout Windows, including on the desktop. It is used by the apps list, search results, the Share pane, the file picker, an overlay on live tiles, the Alt+Tab user interface, Task Manager, file icons for associated file types, and so on. At least the 100% scale image is required. Although the image is nominally 30x30 pixels, this logo supports four additional sizes to be used for file icons on the desktop (if your app has associated file types): 16x16, 32x32, 48x48, and 256x256.

Store Logo—A 50x50 image (at 100% scale) used by the Windows Store. At least the 100% scale image is required.

Visual Studio provides placeholder image files for the required logo images only: the square 150x150 logo, the square 30x30 logo, and the store logo.
To make your tile look good on all devices (and to increase the chances of Microsoft promoting your app in the Windows Store or in advertisements), you should support all scale sizes for each logo you provide. It’s perfectly okay to omit large tile and wide tile logos, however. Many of Microsoft’s own apps omit them. Furthermore, it’s best not to support a large tile and/or wide tile unless you’re going to make it a live tile (covered in Chapter 25). Otherwise, your pinned app occupies more space without adding any extra value.

**Why does each tile logo support four different image sizes, and how are they used?**

Depending on the pixel density of the screen, Windows automatically scales all non-desktop user interfaces to prevent items from being too small to touch or too hard to read. This applies to all Windows Store apps as well as system UI such as the Start screen, file picker, and so on. To prevent your images from looking unsightly by being scaled upward, you can provide multiple versions of any image: one at its normal size, one at 140% of its normal size, and one at 180% of its normal size. The Start screen additionally supports shrinking its content to an 80% scale.

Windows uses a file naming pattern to manage this, and the package manifest designer in Visual Studio automatically names your assigned image files accordingly. By default, the medium tile icon is assigned to `Assets\Logo.png`. However, at runtime, Windows automatically looks for a file with the following name instead, depending on the current scale being applied:
As with the splash screen, you can specify a background color for your tile. For the best results, this color (as well as the tile images) should match what you use in your splash screen. The desired effect of the splash screen is that your tile springs to life and fills the screen in a larger form. Even if your tile background color is completely covered by opaque tile images, there are still contexts in which the color is seen, such as the zoomed-out Start screen view or the Alt+Tab user interface. Therefore, choose your background color (and determine whether you want your images to use transparency) carefully!

You can choose a “default size,” which is the initial size of your tile if the user decides to pin it to the Start screen. This can only be set to the medium tile or the wide tile (if you support a wide tile). If unset, wide is given precedence over medium.

You can also choose a “short name,” which is the text that gets overlaid on the bottom of your tile. You can even specify which tile sizes should show the text: medium, wide, and/or large. (Small tiles do not support overlaid text.) Many apps turn off the text because their images already include a logo with the name.

Finally, you can decide whether you want the overlaid text to be “light” (which means white) or “dark” (which means a dark gray). Although most apps use white text, you may need to choose the dark option if you want your tile to have a light background color.
To create a logo that fits in with the built-in apps, it should have a transparent background and the drawing inside should:

➔ Be completely white
➔ Be composed of simple geometric shapes
➔ Use an understandable real-world metaphor

The drawing used in all logo images should look the same, just scaled to different sizes and with different margins.

For example, the drawing for the 150x150 image should generally fit in a 66x66 box centered but nudged a little higher to leave more space for any overlaid text. Typically the drawing has a 42-pixel margin on the left and right, a 37-pixel margin on top, and a 47-pixel margin on the bottom. The drawing for the 30x30 image should generally fit in a 24x24 centered box, leaving just 3 pixels of margin so it’s easier to see at the small size. Similarly, the 50x50 store logo drawing should occupy a centered 40x40 square (leaving 5 pixels of margin on each side).

Creating white-on-transparent images requires some practice and patience. You’ll want to use tools such as PAINT.NET, mentioned in this book’s “Introduction” section. A few of the characters from fonts such as Wingdings, Webdings, and Segoe UI Symbol can even be used to help create a decent icon! Resources like thenounproject.com can also be helpful.

Of course, games or apps with their own strong branding usually do not follow these guidelines, as being consistent with their own identity outweighs being consistent with Windows.

Capabilities

On the Capabilities tab, you select each capability required by your app. A capability is a special permission for actions that users might not want certain apps to perform, whether for privacy concerns or concerns about data usage charges. In the Windows Store, prospective users are told what capabilities each app requires before they decide whether to download it. To users, they are described as permissions, sometimes with more descriptive names, as shown in Figure 1.6.

![Fresh Paint](image)

**FIGURE 1.6** The Fresh Paint app uses three capabilities: Pictures Library, Webcam, and Internet (Client).
For the most part, user approval of all requested permissions is an implicit part of downloading an app. However, the use of privacy-related capabilities, such as location services, prompts the user the first time an app invokes a relevant API. Furthermore, some capabilities can be disabled or reenabled at any time by a user. When the Settings charm is invoked while a Windows Store app is running, it contains a “Permissions” link that displays an app’s capabilities and toggle switches for any that can be turned on and off. Figure 1.7 shows what this looks like while running HelloRealWorld, both with the default capability already chosen in our package manifest—Internet (Client)—and after selecting every listed capability in the package manifest.

**FIGURE 1.7** The “Permissions” section of the Settings charm lists the current app’s capabilities, and enables turning some of them on or off at runtime.

The long list of available capabilities can be grouped into four different categories:

- File capabilities
- Device capabilities
- Network capabilities
- Identity capabilities

You want to restrict the set of capabilities requested by your app as much as possible, because it is a competitive advantage. For example, users might decide not to buy your fun piano app if it wants permission to use the Internet!
Most of them can be used freely, although some of them are restricted. Apps that use restricted capabilities must go through extra processes when uploaded to the Windows Store and are only granted to business developer accounts with written justification. Fortunately, the restricted capabilities (called out in the upcoming lists) are for uncommon scenarios.

**File Capabilities**

As you’ll read in Chapter 20, “Working with Data,” apps can read and write their own private files in an isolated spot, and those files can even participate in automatic roaming between a user’s devices. In addition, users can give apps explicit permission to read/write other “normal” files and folders via the Windows file picker. This is all that most apps need, and does not require any capabilities.

Beyond these two features, however, programmatic reading and writing of files requires special capabilities. There is one for each of the four built-in libraries (Documents, Music, Pictures, and Videos) plus another for attached storage devices:

- **Music Library**, **Pictures Library**, and **Videos Library**—Enables enumerating and accessing all music, pictures, and videos, respectively, *without* going through the file picker.

- **Documents Library**—Enables adding, changing, and deleting files in the Documents library on the local computer *without* going through the file picker. However, this capability is restricted to specific file type associations that must also be declared in the package manifest (on the Declarations tab). This is listed separately from the preceding three capabilities because it is a restricted capability that needs special approval from Microsoft in order to publish the app in the Windows Store. And unlike the capabilities for the Music, Pictures, and Videos libraries, this cannot be used to access Documents libraries on other computers in the same HomeGroup.

- **Removable Storage**—Enables adding, changing, and deleting files on devices such as external hard drives or thumb drives connected to the local computer, again *without* going through the file picker. As with the preceding capability, this is restricted to file type associations that must also be declared in the package manifest.

**Device Capabilities**

Apps can access simple sensors such as an accelerometer or devices such as a printer without any capabilities. Accessing other sensors or devices does require specific capabilities, however. The list of device types grows over time (and can be extended by third parties), but the Capabilities tab exposes four choices, listed below. For all of them except proximity, users can disable them at any time, so apps must be prepared to handle this gracefully.

- **Location**—Reveals the computer’s location, either precise coordinates from a GPS sensor (if one exists) or an estimation based on network information.
Microphone—Enables recording audio from a microphone.

Webcam—Enables recording video—or capturing still pictures—from a camera. Note that this doesn’t include sound. If you want to record audio and video, you need both Webcam and Microphone capabilities.

Proximity—Enables communication with nearby devices, either via Wi-Fi Direct or near field communication (NFC).

Chapters 14, “Audio, Video, and Speech,” and 23, “Reading from Sensors,” explain how to write apps that take advantage of these capabilities. Additional device capabilities exist that don’t appear on the Capabilities tab. These must be added manually to the package manifest XML. See Chapter 24, “Controlling Devices,” for more information.

**Network Capabilities**

Without any network capabilities, a Windows Store app cannot do any communication over any kind of network except for the automatic roaming of application data described in Chapter 20, the seamless opening/saving of network files enabled by the file picker, or the peer-to-peer connections enabled by the Proximity capability. Four types of network capabilities exist:

- **Internet (Client)**—This is the only network capability that most apps need. It provides outbound access to the Internet and public networks (going through the firewall).

- **Internet (Client & Server)**—This is just like the preceding capability except it provides both inbound and outbound access, which is vital for peer-to-peer apps. It’s a superset of “Internet (Client)” so if you request this capability in your manifest, then you don’t need to request the other one.

- **Private Networks (Client & Server)**—Provides inbound and outbound access to trusted home and work networks (going through the firewall).

- **Enterprise Authentication**—Enables intranet access using the current Windows domain credentials. This is a restricted capability.

**Identity Capabilities**

This is not really a fourth category of capabilities, but rather a single outlier that doesn’t fit anywhere else. The Shared User Certificates capability enables access to digital certificates that validate a user’s identity. The certificate could be installed on the computer or stored on a smart card. This is mainly for enterprise environments, and it is a restricted capability.

**Visual Studio project templates enable the “Internet (Client)” capability by default!**

This is done because the Visual Studio team feared that it would be too confusing for developers if simple network-dependent calls failed in their brand new projects. Therefore, be sure to remove the capability if you don’t need it. Otherwise, your app’s store listing will say that your app “has permission to use your Internet connection.”
Declarations

The Declarations tab is the one with the most options. This is where you declare your app’s support for one or more contracts, if applicable. Contracts enable your app to cooperate with another app, or Windows itself, to complete a well-defined task. Every contract has a source that initiates the task and a target that completes it.

Your app can be the source for a contract without doing anything in the package manifest. (It just makes various API calls.) To be the target, however, your app must be activated in a special manner. This is what requires the declaration in the package manifest. Therefore, you can think of the list of available declarations as the list of available contract targets.

Unlike capabilities, contract target declarations are not listed in the Windows Store as potentially unwanted features. In fact, you should go out of your way to mention your supported contract scenarios, because they can be very useful! There’s nothing about being a contract target that is inherently dangerous for the user. Supporting certain contracts does require relevant capabilities, but many don’t require any. See Chapter 22, “Leveraging Contracts,” for specific examples.

Content URIs

This tab, new to Windows 8.1, only applies if you are hosting HTML content inside your XAML app. It simply houses a list of HTTPS URLs whose JavaScript is allowed (or disallowed) to raise events that can be handled by your app. For more information, see the discussion of the WebView control in Chapter 15, “Other Controls.”

Packaging

The Packaging tab is meant to describe information needed for the app’s listing in the Windows Store. However, for apps in the store, this information is managed by the Windows Dev Center dashboard. You therefore don’t normally need to change these values in your local package manifest:

➔ The package name is a unique identifier. Visual Studio automatically fills it in with a globally-unique identifier known as a GUID. That said, for easier debugging and identification of your app’s local data store, it’s best to replace the GUID with a human-readable name, such as CompanyName.AppName. This name doesn’t impact real users of your app, as the Windows Store assigns this value in the package that users download.

➔ The package display name is the name of your app in the store, but this also gets replaced when you follow the procedure to upload an app, so you can leave this item alone.

➔ The version, set to 1.0.0.0 by default, is a four-part value interpreted as Major.Minor.Build.Revision. You can set this value however you like. There are only two requirements enforced by the Windows Store:
1. Each new published version has a higher version number than previous published versions (for the same target version of Windows).

2. If your app simultaneously has a package for Windows 8 and a package for Windows 8.1, the Windows 8 package version number must never exceed the version number of your first published Windows 8.1 package.

The bottom of this tab contains publisher information based on the certificate used to authenticate the package. Visual Studio configures this to work with the temporary certificate it generates, and the store upload process reconfigures it to work with your developer account.

For testing certain notification or purchase scenarios that depend on an app’s identity in the Windows Store, you can automatically update your local package manifest’s packaging values to match the values maintained by the Windows Store. To do this, you can select Associate App with the Store..., which can be found on the Store menu in Visual Studio Express or on the Project, Store menu in other editions.

Updating XAML and C# Code

With the tour of the package manifest complete, we are ready to fill our blank app with a little bit of content. Let’s look at the remaining files in our project and update them where necessary.

The Main Page User Interface

Every app consists of one or more windows with one or more pages. Our HelloRealWorld project, created from the Blank App template, is given a single window with a single page called MainPage. It defines what the user sees once your app has loaded and the splash screen has gone away. MainPage, like any page that would be used in a XAML app, is implemented across two files: MainPage.xaml contains the user interface, and MainPage.xaml.cs contains the logic, often called the code-behind. Listing 1.1 shows the initial contents of MainPage.xaml.

LISTING 1.1 MainPage.xaml—The Initial Markup for the Main Page

```xml
<Page  
x:Class="HelloRealWorld.MainPage"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:local="using:HelloRealWorld"  
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
mc:Ignorable="d">  
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  
</Grid>  
</Page>
```
At a quick glance, this file tells us:

➔ This is a class called MainPage (in the HelloRealWorld namespace) that derives from a class called Page (the root element in this file).

➔ It contains an empty Grid (an element examined in Chapter 4, “Layout”) whose background is set to a theme-defined color. From running the app, we know this color is a very dark gray (#1D1D1D).

➔ It contains a bunch of XML namespaces to make adding new elements and attributes that aren’t in the default namespace more convenient. These XML namespaces are discussed in the next chapter.

Listing 1.2 updates the blank-screen MainPage.xaml with a few elements to produce the result in Figure 1.8.

**LISTING 1.2  MainPage.xaml—Updated Markup for the HelloRealWorld App**

```xml
<Page x:Class="HelloRealWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HelloRealWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel Name="stackPanel" Margin="100" Background="Blue">
            <TextBlock FontSize="80" TextWrapping="WrapWholeWords" Margin="12,48">
                Hello, English-speaking world!</TextBlock>
            <TextBlock FontSize="28" Margin="12">Please enter your name:</TextBlock>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition Width="Auto"/>
                </Grid.ColumnDefinitions>
                <TextBox Name="nameBox" Margin="12"/>
            </Grid>
            <Button Grid.Column="1" Click="Button_Click">Go</Button>
        </StackPanel>
        <TextBlock Name="result" FontSize="28" Margin="12"/>
    </Grid>
</Page>
```

This listing adds a bunch of new content inside the topmost Grid. The Grid and StackPanel elements help to arrange the user-visible elements: TextBlocks (i.e. labels), a TextBox, and a Button. All of these elements are described in depth in upcoming chapters.
FIGURE 1.8 The HelloRealWorld user interface asks the user to type his or her name.

The idea for this app is to display the user's name in the TextBlock named result once he or she clicks the Go Button. (Granted, this is not a useful app, but it's all we need to demonstrate the concepts throughout the remainder of this chapter.) To act upon the Button being clicked, this XAML specifies that a method called Button_Click should be called when its Click event is raised. This method must be defined in the code-behind file, which we'll look at next.

The Main Page Logic

Listing 1.3 shows the initial contents of MainPage.xaml.cs, the code-behind file for MainPage.xaml. Until we add our own logic, it contains only a required call to InitializeComponent that constructs the page with all the visuals defined in the XAML file. The class is marked with the partial keyword because its definition is shared with a hidden C# file that gets generated when the XAML file is compiled.
LISTING 1.3 MainPage.xaml.cs—The Initial Code-Behind for the Main Page

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at
// http://go.microsoft.com/fwlink/?LinkId=234238

namespace HelloRealWorld
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

We need to add an implementation of the Button_Click method referenced by the XAML. It can look as follows:

```csharp
void Button_Click(object sender, RoutedEventArgs e)
{
    this.result.Text = this.nameBox.Text;
}
```

The named elements in the XAML correspond to fields in this class, so this code updates the result TextBlock with the

---

Never remove the call to InitializeComponent in the constructor of your code-behind class! InitializeComponent is what associates your XAML-defined content with the instance of the class at run-time.
Hello, English-speaking world!

Please enter your name:

Adam

Adam

FIGURE 1.9 The result TextBlock contains the typed text after the user clicks the Button.

The Application Definition

The application definition is contained in App.xaml and its code-behind file, App.xaml.cs. App.xaml is a special XAML file that doesn’t define any visuals, but rather defines an App class that can handle application-level tasks. Usually the only reason to touch this XAML file is to place new application-wide resources, such as custom styles, inside its Application.Resources collection. Chapter 18, “Styles, Templates, and Visual States” contains many examples of this. Listing 1.4 shows the contents of App.xaml in our HelloRealWorld project.

LISTING 1.4 App.xaml—The Markup for the App Class

```xml
<Application
  x:Class="HelloRealWorld.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:HelloRealWorld">
  
  
</Application>
```
Listing 1.5 contains the auto-generated contents of the code-behind file for App.xaml. It contains three vital pieces:

➔ A constructor, which is effectively the app’s main method. The plumbing that makes it the app’s entry point is enabled by an “Entry point” setting in the package manifest (on the Application tab). When you create a project, Visual Studio automatically sets it to the namespace-qualified name of the project’s App class (HelloRealWorld.App in this example).

➔ Logic inside an OnLaunched method that enables the frame rate counter overlay in debug mode, navigates to the app’s first (and in this case only) page, and calls Window.Current.Activate to dismiss the splash screen. If you want to add a new page and make it be the starting point of the app, or if you want to customize the initialization logic, this is where you can do it. See Chapter 7, “App Lifecycle,” for more information.

➔ An OnSuspending method that is attached to the base class’s Suspending event. This gives you an opportunity to save state before your app is suspended, although the generated code does nothing here other than provide a TODO comment. Chapter 7 examines app suspension.

LISTING 1.5  App.xaml.cs—The Code-Behind for the App Class

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace HelloRealWorld
{
    /// <summary>
    /// Provides application-specific behavior to supplement the base class.
    /// </summary>
    /// <summary>
    sealed partial class App : Application
```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}

protected override void OnLaunched(LaunchActivatedEventArgs args)
{
#if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = true;
    }
#endif

    Frame rootFrame = Window.Current.Content as Frame;

    if (rootFrame == null)
    {
        // Create a Frame and navigate to the first page
        var rootFrame = new Frame();

        if (args.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: Load state from previously suspended application
        }

        // Place the frame in the current Window
        Window.Current.Content = rootFrame;
    }
}
if (rootFrame.Content == null)
{
    // When the navigation stack isn't restored, navigate to the first page
    if (!rootFrame.Navigate(typeof(MainPage), args.Arguments))
    {
        throw new Exception("Failed to create initial page");
    }
}

// Ensure the current Window is active
Window.Current.Activate();

/// <summary>
/// Invoked when application execution is being suspended. Application state
/// is saved without knowing whether the application will be terminated or
/// resumed with the contents of memory still intact.
/// </summary>
/// <param name="sender">The source of the suspend request.</param>
/// <param name="e">Details about the suspend request.</param>
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    //TODO: Save application state and stop any background activity
    deferral.Complete();
}

There’s one more file—AssemblyInfo.cs—but it’s not worth showing in this book. It contains a bunch of attributes where you can put a title, description, company name, copyright, and so on that get compiled into your assembly (the EXE or DLL). But setting these is unnecessary because all of the information used by the Windows Store is separately managed. Still, the AssemblyVersion and AssemblyFileVersion attributes, typically set to the same value, can be useful for you to keep track of distinct versions of your application:

If you want to create a richer splash screen, perhaps with an animated progress graphic, the way to do this is by mimicking the splash screen with a custom page. Inside App.OnLaunched, you can navigate to an initial page that looks just like the real (static) splash screen but with extra UI elements and custom logic. The instance of LaunchActivatedEventArgs passed to OnLaunched even has a SplashScreen property that exposes an ImageLocation rectangle that tells you the coordinates of the real splash screen image. This makes it easy to match the splash screen’s appearance no matter what the current screen’s resolution is. Such a user interface is often called an “extended splash screen.”
By using *-syntax, such as "1.0.*", you can even let the version number auto-increment every time you rebuild your app.

Making the App World-Ready

At this point, our HelloRealWorld app still only says "hello" to the English-speaking parts of the world. The Windows Store serves hundreds of markets and over a hundred different languages, so ignoring them greatly reduces the audience for your app. Making your app world-ready involves two things: globalization and localization.

Globalization refers to making your app act appropriately for different markets without any changes or customizations. An example of this is formatting the display of currency correctly for the current region without writing special-case logic. The Windows.Globalization namespace contains a lot of functionality for handling dates and times, geographic regions, number formatting, and more. Plus, built-in XAML controls such as DatePicker and TimePicker, discussed in Chapter 15, are globalization-ready. For many apps, these features might not apply.

Localization, which is relevant for practically every app, refers to explicit activity to adapt an app to each new market. The primary example of this is translating text in your user interface to different languages and then displaying the translations when appropriate. Performing this localization activity is the focus of this section.

To make an app ready for localization, you should remove hardcoded English strings that are user-visible, and instead mark such elements with a special identifier unique within the app. Listing 1.6 updates our XAML from Listing 1.2 to do just that.

**LISTING 1.6  MainPage.xaml—Markup with User-Visible English Text Removed**

```xml
<Page x:Class="HelloRealWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HelloRealWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
        <StackPanel x:Uid="Panel" Name="stackPanel" Margin="100">
            <TextBlock x:Uid="Greeting" FontSize="80" TextWrapping="WrapWholeWords" Margin="12,48"/>
            <TextBlock x:Uid="EnterName" FontSize="28" Margin="12"/>
        </StackPanel>
    </Grid>
    <Grid.ColumnDefinitions>
```

The x:Uid marking is completely independent from an element’s Name. The former is specifically for the localization process, and the latter is for the benefit of code-behind. Note that Listing 1.6 not only removes the three hardcoded strings from the two TextBlocks and the Button, but it also removes the explicit "Blue" color from the StackPanel! This way, we can customize the color for different languages in addition to the text.

With the IDs in place and the text and color for English removed, we need to add them back in a way that identifies them as English-only. To do this, add a new folder to the solution called en. This is the language code for all variations of English. If you want to target the United Kingdom separately, you could add a folder called en-GB. If you want to target Canada separately, you could add a folder called en-CA. And so forth.

Right-click on the en folder and select Add, New Item, then pick Resources file from the General tab. The default name of Resources.resw is fine. This file is a table for all your language-specific strings. Figure 1.10 shows this file populated for English.

Each value must be given a name of the form UniqueId.PropertyName. UniqueId must match the x:Uid value for the relevant element, so the Panel.Background entry in Figure 1.10 sets Background to Blue on the StackPanel marked with x:Uid="Panel" in Listing 1.6. From the listing, it’s not obvious that GoButton’s relevant property is called Content, unlike the TextBlocks’ property called Text, but as you learn about the different elements throughout this book, you’ll understand which properties to set.
After filling out the Resources.resw file, you can run the HelloRealWorld app and the result is identical to what we saw earlier in Figures 1.6 and 1.7. However, the app is now ready to be localized for other languages.

We could add additional folders named after language codes and manually populate translated resources with the help of a knowledgeable friend, a professional translator, or translation software. Depending on the current user’s language settings, the appropriate resources are chosen at runtime, with a fallback to the default language if no such resources exist.

However, a better option exists. To take advantage of it, you must download and install the Multilingual App Toolkit from the Windows Dev Center. Once you do this, you can select Enable Multilingual App Toolkit from Visual Studio’s Tools menu. This automatically adds an .xlf file to a new subfolder added to your project called MultilingualResources for a test-only language called Pseudo Language.

We’ll leverage the Pseudo Language in a moment, but first let’s add support for a second real language: Traditional Chinese. To do this, right-click on your project in Solution Explorer and select Add translation languages…. This produces the dialog shown in Figure 1.11.

In this dialog, Pseudo Language and our default English language is already selected, but we can scroll down and select Chinese (Traditional) [zh-Hant] from the list. After pressing OK, the MultilingualResources folder now has two .xlf files: one for Pseudo Language, and one for Traditional Chinese.

FIGURE 1.11 The Multilingual App Toolkit automates the process for supporting new languages.
Now rebuild the HelloRealWorld app. This populates each .xlf file with a “translation” for each item from the default language .resw file. Initially, each translation is just the duplicated English text. However, for some languages, such as the two we’ve chosen, you can generate machine translations based on the Microsoft Translator service! To do this for the entire file, right-click on each .xlf file and select **Generate machine translations**.

Voilà! Now we’ve got initial translations for all of our resources, which you can see by opening each .xlf file and examining the list inside the multilingual editor. This is shown in Figure 1.12.

Your willingness to trust the results from machine translation is a personal decision, but at least machine translation is a good starting point. (Notice that the generated translations are automatically placed in a “Needs Review” state.) That said, we definitely don’t want the Blue text translated to 藍色! This isn’t a user-visible string, and 藍色 is not a valid value for Background. Instead, let’s “translate” it to red, which will serve as our language-specific background color. Similarly, we don’t want Blue’s Pseudo Language translation of [D05A0][!!_Blüe_!!], so let’s change that to Green.

---

**What is Pseudo Language?**

Pseudo Language is designed to test how well your app handles being localized to various (real) languages. When leveraging machine translation to Pseudo Language, you get an English-looking string whose contents are still recognizable, but designed to catch problems.

Pseudo Language strings are longer than the corresponding English strings, to help you catch cases where text might get truncated or cause issues from wrapping when you translate to a real language whose text tends to be longer than English. Each string also begins with an ID, to help you track a problematic piece of text to its original resource. For example, a Pseudo Language translation of *Hello, English Speaking World!* can look like [07223] [!!_HeiʃO, Ẹŋgliʃ spɛәŋĩŋ wɔɹld !!!!] Because of the unique appearance of Pseudo Language, it also helps you catch user-visible text in your user interface that you forgot to extract to a resource.

**What are .xlf files?**

These files, which are generated by the Multilingual App Toolkit, are XLIFF files, an industry-standard XML format for localizable data. In addition to listing source and target strings (with optional comments), these files enable a workflow in which resources can be marked as New, Needs Review, Translated, Final, or Signed Off.

The benefit of using XLIFF files to store translations is that you can send them directly to a professional translation vendor, as they should already have a workflow involving this format. Or, if you leverage friends to do your translations, you can have them install the Multilingual App Toolkit and use its Multilingual Editor in a standalone fashion. No Visual Studio installation is necessary.

Visual Studio includes functionality for packaging and sending XLIFF files, as well as importing updated files that merge with your local content. These options can be found by right-clicking an .xlf file in Solution Explorer.
We have one more change to make. We don’t want “Hello, English-speaking world!” to be translated to Chinese, but rather “Hello, Chinese-speaking world!” Both Microsoft Translator and a colleague tell me that “!” is a valid translation, so we can paste that into the appropriate spot of the Chinese .xlf file.

After rebuilding the project, we are now ready to test the localized versions of HelloRealWorld. Just as if we had manually added separate .resw files in per-language folders, the translated resources are used automatically based on the current Windows language settings.

To change the default language used by Windows, you can either use the PC Settings app or the desktop Control Panel. In PC Settings, this can be found under Time & language; Region & language. In Control Panel, it’s under Clock, Language, and Region; Language. Add Chinese (Traditional) and make it the default language to test the Traditional Chinese resources.

To add Pseudo Language (and make it the default language), you have to use a hidden trick in Control Panel. After clicking Add languages, type qps-ploc in the search box for

---

**FIGURE 1.12** Each .xlf file contains machine-generated initial translations, courtesy of Microsoft Translator.
the entry called **English (qps-ploc)** to appear. You must type *the whole thing* for this to work! This language is hidden in this way because no normal user should ever enable it.

Figure 1.13 shows the result of running `HelloRealWorld` when Windows is set to use each of the two non-English languages. These changes are handled completely by the resource-loading mechanism. Other than the switch to marking elements with `x:Uid`, no code changes were needed. This figure also highlights Pseudo Language's knack for using really long strings that can highlight potential weaknesses in your app's layout.

![HelloRealWorld in Traditional Chinese and Pseudo Language](image)

**FIGURE 1.13** `HelloRealWorld` now acts appropriately for Traditional Chinese and for the test-only Pseudo Language.

---

You can add additional languages to your apps that have already been published in the Windows Store, thanks to *resource pack* support in Windows 8.1. As long as you don't update any code or your version number, your new resources get downloaded only to users with a matching language preference.

The Microsoft Local Language Portal (http://www.microsoft.com/language) is a fantastic resource for getting translations. You can search for terms and get a translation in every language supported by Windows (over 100). These are not machine translations, but rather translations Microsoft has used in their own products. As such, they tend to be geared towards the kind of user-visible labels that are commonly found in software. The portal even shows you which products have made use of the translated terms. Just be sure you agree with the license and terms of use, which can be found on the website.
Making the App Accessible

XAML apps have a number of accessibility features built in, designed to help users with disabilities. You can test this support by enabling various features in the Ease of Access section in the PC Settings app. You can configure Narrator, a screen reader, and witness it convey information about your app with varying degrees of success. (You can quickly toggle Narrator on and off by pressing Windows+Enter.) You can choose a high contrast theme and watch controls used by your app automatically change to match the theme. You can turn off standard animations.

And so on.

To make your app usable to the broadest set of customers, including people with disabilities, you should take steps to ensure it works even better with these assistive technologies. In this section, we look at improving the screen reading experience for our HelloRealWorld app, and accounting for high contrast themes.

Improving Screen Reading

If you turn on Narrator and launch the HelloRealWorld app (with English as the Windows default language), you hear the following:

"HelloRealWorld window"

"Editing"

The first utterance is triggered by the app’s window getting focus, and the second utterance is triggered by the TextBox getting focus (which happens automatically).

This experience isn’t good enough, because Narrator doesn’t report the purpose of the TextBox. To fix this, we need to leverage the UI Automation framework, which is as simple as setting the following automation property on the TextBox:

```xml
<TextBox AutomationProperties.Name="Please enter your name"
        Name="nameBox" Margin="12"/>
```

If you add this property then rerun HelloRealWorld with Narrator on, you will hear the following:

"HelloRealWorld window"

"Please enter your name"

"Editing"

The Windows SDK includes several tools that help you ensure that your app is accessible. The most important one is **UI Accessibility Checker**, which reports missing accessibility information in your app. Others are **Inspect**, which is a viewer for accessibility data on your elements, and **Accessible Event Watcher**, which focuses on the accessibility events that should be raised.
Note that when you give the Go Button focus, such as by pressing Tab, Narrator says:

“Go button”

This works automatically, thanks to built-in Button behavior that reports its content to the UI Automation framework.

When you click the Button, however, Narrator gives no indication that text has been added to the screen. If a message is worth showing, then it's worth hearing as well. To fix this problem, we can add the following automation property to the result TextBlock that identifies it as a live region:

```xml
<TextBlock AutomationProperties.LiveSetting="Polite" Name="result" FontSize="28" Margin="12" />
```

A live region is an area whose content changes. This AutomationProperties.LiveSetting property can be set to one of the following values:

- **Off**—This is the default value.
- **Polite**—Changes should be communicated, but they should not interrupt the screen reader.
- **Assertive**—Changes should be communicated immediately, even if the screen reader is in the midst of speaking.

Live region changes are not detected automatically, however. You must trigger them in C#. In our example, we just need to add an extra line of code to the existing Button_Click event handler:

```csharp
void Button_Click(object sender, RoutedEventArgs e)
{
    this.result.Text = this.nameBox.Text;
    // Notify a screen reader to report this text
    TextBlockAutomationPeer.FromElement(this.result).RaiseAutomationEvent(AutomationEvents.LiveRegionChanged);
}
```

TextBlock, as with other controls, has a peer class in the Windows.UI.Xaml.Automation.Peers namespace. These classes are named with the pattern `ElementNameAutomationPeer`, and have several members that are designed for accessibility as well as automated testing.
Handling High Contrast Themes

The built-in controls automatically adjust their appearance when the user enables a high contrast theme. They adjust their colors to match the theme’s eight user-customizable colors, and in some cases they change their rendering in other ways. Because of this, your app can automatically look correct under a high contrast theme without you doing extra work. However, when you use images or hardcoded colors, which are quite common, problems arise. Images can be a problem when they convey information but do not use enough contrast. Hardcoded colors are a problem for the same reason, but also because they can make things completely unreadable when intermixed with colors that drastically change under a high contrast theme. In general, mixing hardcoded colors with dynamic colors can be a recipe for disaster.

HelloRealWorld doesn’t use any images, but Chapter 13 explains how you can provide separate versions of your images that can be used for high contrast themes only.

For HelloRealWorld, the hardcoded blue (or red or green) background color could be problematic as the colors of the other elements change. (Although none of the high contrast themes use blue, red, or green as a text color by default, the user could always choose it for the color of text.) We can fix this in code-behind by checking whether the app is running under high contrast and simply removing the StackPanel’s Background in that case:

```csharp
public sealed partial class MainPage : Page
{
    Brush defaultBackground;

    public MainPage()
    {
        InitializeComponent();

        // Save the default background for later
        this.defaultBackground = this.stackPanel.Background;
    }

    // Other code...
}

After the work we did to localize the HelloRealWorld app, it would be unfortunate to give screen readers a hardcoded English string, as shown earlier:

```xml
<TextBox AutomationProperties.Name="Please enter your name"
         Name="nameBox" Margin="12"/>
```

Fortunately, automation properties can be localized just like any other property. To do this, remove the explicit setting and give the element an x:Uid:

```xml
<TextBox x:Uid="NameBox" Name="nameBox" Margin="12"/>
```

In this example, you should then add an entry in the Resources.resw file named NameBox. AutomationProperties.Name, and its value for English should be "Please enter your name".
```csharp
AccessibilitySettings settings = new AccessibilitySettings();

// Update the background whenever the theme changes
settings.HighContrastChanged += OnHighContrastChanged;

// Set the background appropriately on initialization
OnHighContrastChanged(settings, null);

void OnHighContrastChanged(AccessibilitySettings sender, object args)
{
    this.stackPanel.Background = sender.HighContrast ? null : this.defaultBackground;
}

...

Because the user could change the theme while our app is running, we need to handle the HighContrastChanged event to adjust accordingly. The rest of the app's elements already adjust automatically. Figure 1.14 shows the result of adding this code then running the app under two different high contrast themes. Chapter 18 explains how you can define theme-specific colors without needing to write C# code such as this.

FIGURE 1.14  Removing the explicit StackPanel background makes the app look appropriate under any high contrast theme.
By defining and using the defaultBackground member, the code that handles the HighContrastChanged event preserves the language-specific background color that comes from one of the Resources.resw files. It does so without needing to programmatically retrieve the current resource value. However, if you need to do so, you can use code like the following for the Panel1.Background value:

```csharp
string backgroundString = rc.ValueAsString;
```

If you do the following:
➔ check that the Windows SDK accessibility tools have no high-priority complaints about your app
➔ verify that your app acts appropriately when using Narrator
➔ verify that your app acts appropriately when running under high contrast
➔ verify that your app can be used when navigating using only the keyboard
then you should take credit for your work and check the “My app meets accessibility guidelines” checkbox within your app’s listing in your Windows Dev Center dashboard. This fact gets advertised in the Windows Store, and it makes your app shows up for users who search for accessible apps.

### Submitting to the Windows Store

Once your app is finished, you can submit it to the Windows Store via items on the Store menu in Visual Studio Express, or via the Project, Store menu in other editions of Visual Studio. The Visual Studio integration works in concert with pages on the Windows Dev Center website to help you complete your submission. Before doing this, however, you have some tasks to complete:

➔ **Set up your developer account** at http://dev.windows.com, get it verified, and fill out your payout and tax information. This can take a couple of days for an individual account, or a couple of weeks for a business account.

➔ **Reserve your app name** with the Windows Store, as it requires each app’s name to be unique. You can reserve names at any time, and you have up to a year to submit the app before losing each reservation. You can also reserve additional names for other languages.
→ **Download, install, and run the Windows App Certification Kit** (WACK) from the Windows Dev Center. This tests your app for violations that cause it to fail the Windows Store certification process, so running it in advance can save you a lot of time.

The Windows Store certification process consists of three parts:

→ **Technical checks.** This is simply running the Windows App Certification Kit on your app. If you pass its tests before submitting your app, you should have nothing to worry about here.

→ **Security checks.** This ensures that your software isn’t infected with a virus, which again should not be a concern for most developers.

→ **Content checks.** This is the trickiest part of the process and, unlike the other two, is performed manually by human reviewers. Reviewers ensure that the app does what it claims to do and follows all the app certification requirements published in the Windows Dev Center.

The very first certification requirement is that the app “must offer customers unique, creative value or utility,” so **HelloRealWorld** is bound to fail this requirement. This requirement may be obvious, but there are some requirements that often surprise people and cause many apps to fail certification:

→ If your app requires a network capability, you must write a privacy statement that explains what data you collect, how you store or share it, how users can access the collected data, and so on. Requirement 4.1 in the Windows Dev Center helps you figure out how to write one. Furthermore, a link to the statement must be reachable from the Settings pane for your app, and the same link must be included in your listing in the Windows Store. See Chapter 21, “Supporting Charms,” for information about adding content to the Settings pane.

→ You must select an appropriate age rating, using guidelines from the Windows Dev Center. For example, most apps that share personal information must be rated at least 12+. Regardless of your app’s rating, its listing for the Windows Store cannot contain content that is considered too mature for a 12+ rating.

→ You must provide descriptions and screenshots for every language you support. If your app is only partially localized for some languages, you must mention this in your listing.

If you fail certification, you must address the issue(s) and resubmit your app. When you do so, it goes through the entire process again, at the end of the line. Fortunately, at the time of this writing, the average length of certification is only about 2.5 days.

---

**Don’t forget to remove capabilities you don’t need!**

The certification process doesn’t warn you about capabilities you don’t actually use, so it’s up to you to make sure the list is not larger than it needs to be.
Summary

You’ve now seen the basic structure of a Visual Studio project for a XAML-based Windows Store app and gotten a taste for making an app that is ready to sell across the world. If you’ve previously done .NET development, much of this should look familiar. If you’ve previously dabbled in Windows Presentation Foundation (WPF) and/or Silverlight, the role of the XAML files and the C# files should be obvious. And if you’ve previously done development for Windows Phone, then all of these concepts, including things like capabilities, shouldn’t surprise you one bit. If you don’t have any such experience, then you should at least be able to appreciate how easy it is to hit the ground running.

Personally, I’m struck by how easy it has become to localize your app and make it accessible. Software development has come a long way over the years, and you’ll see evidence of this throughout the book, when it comes to handling heterogeneous screen DPI, making money through the Windows Store, communicating with slick peripherals, and much more. The team behind Windows Store apps has taken the best ideas from .NET, XAML, Windows Phone, the Web, C++, and COM in order to create a compelling platform that’s easy for developers to dive into. And now it’s time to dive much deeper into the language of XAML.
INDEX

Symbols & Numerics

{} (curly braces), escaping, 50

2D transforms, applying, 71-72
  RotateTransform, 73
  ScaleTransform, 74-75
  SkewTransform, 76
  TranslateTransform, 77
3D transforms, 79-81
3D video, 369

A

AAMA (American Association of Motor Vehicle Administrators) cards, 688
absolute sizing, 98
absolute URIs, referencing files with, 330
accelerometer, reading from, 661-662
  shake detection, 664
  tossing motion, 663-664
accessibility features of XAML apps, 36-37
accessing known folders, 582-583
Account Picture Provider contract, 629-631
activating
  apps, 182, 185-186
  versus launching, 186
activation contracts, 628
AdControls, 216-217
  ErrorOccurred event, 219
  IsEngagedChanged event, 218
  refresh behavior, 218
  relevant ads, displaying, 219
AddAudioEffect method, 373-374
AddDeleteThemeTransition, 476-478
adding
  ads to a page, 216-218
  search results page to SearchBox control, 405
AddVideoEffect method, 373-374
adjusting
  camera settings, 388-389
  theme animations, 486
agile objects, 196
alarm apps, 725
alarm notifications, 726-727
alignment, 69
  content alignment, 70
  flow direction, 69
  Stretch alignment, 69
Alipay, 214
altform-xxx resource qualifier, 340
animation
  custom animations
    data types, 486-487
    dependent versus independent animations, 488-489
    duration of, 489
    easing functions, 505, 508-509
    From property, 490-492
    property paths, 496-498
    Storyboards, 495-500
    Timeline properties, 493-495
    To property, 490-492
  dependent animations, 470
  DirectX swap chain, integrating, 747-751
duration of, 495
EnableDependentAnimation property, 488-489
event triggers, 483-484
hardware acceleration, 470
independent animations, 470
keyframes, 500
  discrete keyframes, 503-504
  easing keyframes, 504
  linear keyframes, 500-501
  spline keyframes, 501-502
manual animations, 509-511
theme animations
  adjusting, 486
  Storyboards, 481-484
  Timeline class, 486
theme transitions, 470
  AddDeleteThemeTransition, 476, 478
  applying to elements, 470-471
  ContentThemeTransition, 474
  EdgeUIThemeTransition, 475
  EntranceThemeTransition, 472
  PageThemeTransition, 475-476
  PopupThemeTransition, 474
  ReorderThemeTransition, 480
  RepositionThemeTransition, 478-479
visual transitions, 537-542
XAML, 469
APIs
  device protocol APIs, 679
  device scenario APIs, 679
  DirectX, integration with XAML UI Framework, 739-751
  KeyboardCapabilities, 317
  SimpleOrientationSensor API, reading from, 666
  WinRT, 3
app bars. See also app data; apps
  bottom app bar, 262
  CommandBar control, 267-269
  top app bar, 262
app data, 573
  app files, 576
    local files, 577
    roaming files, 577
    temporary files, 578
  app settings, 574
    local settings, 574
    roaming settings, 575-576
AppBar content controls, 264-266
AppBarButton content controls, 247-248
AppBarToggleButton content controls, 256
application definition, HelloRealWorld project, 26-29
Application tab (Visual Studio), 12-16

applying

2D transforms, 71-72
RotateTransform, 73
ScaleTransform, 74-75
SkewTransform, 76
TranslateTransform, 77
background color to Grid panel, 100
theme transitions to animation, 470-471

appointments providers

sources, 649
targets, 650

apps

closing, 177-179
customizing launch, 192-193
files, 576
local files, 577
roaming files, 577
temporary files, 578
launching in Visual Studio, 10-11
lifecycle, 175-177
activating apps, 185-186
killing apps, 177-178
launching apps, 182-185
resuming apps, 181
suspending apps, 178-179
suspending event, handling, 180
terminating apps, 181
package manifest, 12
settings, 574
local settings, 574
roaming settings, 575-576
silodded, 221
states of, 176
submitting to Windows Store, 40-41
suspended, 175
themes, 242
content controls, 243
Flyouts, 247
UI framework pairings, 1
XAML, accessibility features, 36-37

arbitrary objects (content controls), 243
ArcSegment class, 445

arrange step (layout), 784-785
ASHWID (App Specific Hardware ID), 234
associating flyouts to buttons, 245-246
ASTA (App Single-Threaded Apartment) threads, 196
Async suffix, 196-198
asynchronous methods, 196-198
attributes, 60
audio
capturing, 380, 391
custom media formats, 377-380
effects, adding, 373-374, 397
format, changing, 395
markers, 373
playback, 368-370
customizing, 370-372
looping, 371
quality, changing, 392-394
speech synthesis, 397-399
trimming, 396
authentication, fingerprint authentication, 680
Auto length, 65
AutoConfiguration property (ImageScanner), 682
automated testing, 238-240
AutoPlay Content contract, 631-633
AutoPlay Device contract, 631-633
AutoReverse property (Timeline), 493
autosizing, 98
await feature (C#), 196-198
awaitable CreateWindowsAsync method, 200
awaiting a method, 197

B

back button command (CoreWindowDialog control), 413
BackEase function, 508
background audio task, 651-654
background color, applying to Grid panel, 100
background tasks
  background audio, 651-654
  custom background tasks, 655-658
BackgroundDownloader class, 589-591
badges, 718-720
bank cards, 688
barcode scanners, 684
  claiming the device, 685-686
  enabling the device, 686
  retrieving the data, 686-687
BarCodeScannerReport, 686
BaseTextBlockStyle, 518
BasicProperties, reading and writing from a file, 347
BeginTime property (Timeline), 493
behaviors
  of AdControl, 218
  refresh behavior, 218
  relevant ads, displaying, 219
creating
  PlayingCard control, 759-761
  templated controls, 771-772
  user controls, 757-758
Bézier curves, 445
BezierSegment class, 445
BGRA8, 335
binary format (XAML), 59
Binding markup extension, 545
  binding
    to collections, 550-553
    to plain properties, 548
    views, 562-565
C#, 547
controlling rendering, 554
  data templates, 554, 556-557
  template selectors, 558
DataContext property, 549-550
RelativeSource property, 547
rendering, improving performance of, 566-569
source property, 546
target property, 546
UpdateSourceTrigger property, 549
value converters, 558-562
BitmapDecoder class
  decoding images, 342-343
  pixels, retrieving data from, 343-344
  transcoding, 356-359
BitmapEncoder class
  encoding images, 351, 353
  transcoding, 356-359
  writing pixel data, 353-354
BitmapIcon content controls, 252-254
BitmapProperties, reading from decoder, 349-350
BitmapSource class, WriteableBitmap subclass, 334-337
BitmapTransform class, 346
Blocks, 304
Bluetooth Smart devices
  connecting to the device, 690-691
  communicating with the device, 691, 694
  declaring device capability, 689, 692-693
bottom app bar, 262
BounceEase function, 508
Brushes, 452
  color brushes, 452
    LinearGradientBrush, 453-459
    SolidColorBrush, 452-453
tile brushes, 459
  ImageBrush, 460-461
  WebViewBrush, 461-463, 466
bubbling, 125-128
bulk transfers, 697
business models, 213
Button content control, 244
  ButtonBase behaviors, 247
    Flyouts, 245-247
ButtonTypeBase behaviors, 247
buttons
  AppBarToggleButton content control, 256
  back button command (CoreWindowDialog control), 413
  clicking, 244, 283
  custom buttons, adding to MessageDialog control, 416
  HyperlinkButtonClick control, 254-255
  password reveal button, 324
RadioButton control, 258-259
RepeatButton content control, 256
tapping, 244
ToggleButton content control, 256

C

C#, 2
await feature, 197-198
composing JavaScript, 434-435
data binding, 547
updating, 22
cached composition, 467
caching (HTTP), retrieving data over networks, 587
caching pages, 207-208
camera, adjusting settings, 388-389
CameraCaptureUI class, 381, 383
Canvas, SimpleCanvas, 589, 785-789
Canvas panel, 89-92
Capabilities tab (Visual Studio), 17-18
device capabilities, 19-20
file capabilities, 19
identity capabilities, 20
network capabilities, 20
CaptureElement, 384-386
capturing
audio, 380, 391
photos, 381-383, 387-389
pointers, 138-143
video, 380, 383-386, 389-390
web content with WebView control, 435-436
catalog, displaying, 227-228
certification process (Windows Store), 41
change notification, 120
CharacterSpacing property (TextBlock), 296
chars
Devices charm, 611
Play To feature, 618-620
printing, 611-617
projection, customizing, 620
Search charm, 597-598
Search pane, customizing, 599-600, 603
Search pane, showing programatically, 603
Windows 8 Search Contract, 598-599
Settings Charm, 621-622, 624
Share charm, 603-610
CheckBox content controls, 257-258
CheckLicense method, 221
children of object elements
collection items, 52
dictionaries, 54-55
lists, 52-53
content property, 52
processing rules, 55
CircleEase function, 508
classes
ArcSegment, 445
BackgroundDownloader, 589-591
BezierSegment, 445
BitmapDecoder
decoding images, 342-343
retrieving pixel data, 343-344
transcoding, 356-359
BitmapEncoder
encoding images, 351-353
transcoding, 356-359
writing pixel data, 353-354
BitmapTransform, 346
CameraCaptureUI, 381, 383
CompositeTransform class, 77
CurrentAppSimulator, 235-239
EdgeGesture, 150
Geolocator, 666-668
GestureRecognizer, 146-149
InkManager, 161-165
handwriting recognition, 166-167
ItemControl, 271
LicenseInformation, 222
LineSegment, 445
MatrixTransform, 78-79
MediaCapture, 199
MediaStreamSource, 378
MediaTranscoder, transcoding, 392-397
combining transforms
- CompositeTransform class, 77
- MatrixTransform class, 78
- TransformGroup class, 78

combo boxes, DatePicker control, 426-427

CommandBar content controls, 267-269

commands, 129
- CoreWindowFlyout control, 415
- MVVM architecture, 129-130

communicating with peer devices, 675-676

comparing
- dependent and independent animations, 488-489
- DispatcherTimer and ThreadPoolTimer classes, 509-511
- launching and activation, 186

compass, reading from, 665

compatibility, viral compatibility, 141

composing
- HTML with XAML, 433-434
- JavaScript with C#, 434-435

CompositeTransform class, 77

configuring scan sources (image scanners), 682

connection information, obtaining, 594

constructors, 27

color brushes, 452
- LinearGradientBrush, 453-459
- SolidColorBrush, 452-453

clearing local values, 123

clicking, 244, 283

client-side code, 734-736

clipping, handling content overflow, 105-106

ClockIdentifier property (TimePicker control), 428

Closed state (MediaElement), 372

closing apps, 177-179

code-behind, 22, 58

collections
- binding to, 550-553
- dictionaries, 54-55
- lists, 52-53
- views, navigating, 565
- views, customizing, 562-565

columns, sizing in Grid panel, 98, 100

c quasi devices, 156

navigationHelper, 208

NetworkInformation, 594

OrientationSensor, 666

PlaneProjection, 79

PolyBezierSegment, 445

PolyLineSegment, 445

PolyQuadraticBezierSegment, 445

PowerEase, 506

QuadraticBezierSegment, 445

StorageFile, 572

StorageFolder, 572

StorageLibrary, 583

SurfaceImageSource, 740

SuspensionManager, 188-189

TextElement, 303-304

TextPointer, 307

Thickness, 66-68

Timeline, 486

TransformGroup class, 78

UserConsentVerifier, 680

UserInformation, 630

VisualState, 534

contact sources, 645-646

Contact Manager, Contact contract, 644-648

containerContentChanging event, 567-569

containers, item containers, 273

content alignment, 70

content controls, 241-242
- app themes, 243
- AppBar, 264-266
- AppBarButton, 247-248
AppBarToggleButton, 256
BitmapIcon, 252-254
Button, 244
    ButtonBase behaviors, 247
    Flyouts, 245-247
CheckBox, 257-258
CommandBar, 267-269
Content property, 244
FontIcon, 251
HyperlinkButton, 254-255
PathIcon, 252
RadioButton, 258-259
RepeatButton, 256
SymbolIcon, 249-250
ToggleButton, 256
ToolTip, 259-262
content overflow, handling, 105
    with clipping, 105-106
    with scaling, 113-115
    with scrolling, 107-111
content property, 52, 244
Content URIs tab (Visual Studio), 21
ContentThemeTransition, 474
contracts, 21, 627
    Account Picture Provider contract, 629-631
    activation contracts, 628
    AutoPlay Content, 631-633
    AutoPlay Device, 631-633
    Contact Picker contract
        contact picker sources, 642
        contact picker targets, 643-644
    File Associations declarations, 634-635
    File Open Picker contract, 637-638, 640
    File Save Picker contract, 641-642
    New Appointments Provider contract
        appointments provider sources, 649
        appointments provider targets, 650
    New Contact contract, 644
        contact sources, 645
        contact targets, 647-648
    Protocol contract, 636-637
    Share contract, 627
contrast resource qualifier, 339
control parts, 775
control points (Bézier curves), 445
control templates, 524-525, 555
    dependency properties, hijacking, 531-532
    property values, inserting, 526-531
controlling
    rendering in data binding, 554
    data templates, 554-557
    template selectors, 558
size
    Auto length, 65
    height, 64-66
    Margin property, 66-68
    Padding property, 66-68
controls
    behavior, creating, 759-761
    consuming, 764
    content controls, 243
        AppBar, 264-266
        AppBarButton, 247-248
        AppBarToggleButton, 256
        BitmapIcon, 252-254
        Button, 244-247
        CheckBox, 257-258
        CommandBar, 267-269
        FontIcon, 251
        HyperlinkButton, 254-255
        PathIcon, 252
        RadioButton, 258-259
        RepeatButton, 256
        SymbolIcon, 249-250
        ToggleButton, 256
        ToolTip, 259-262
    creating, 759
date and time controls
    DatePicker, 426-427
    ProgressRing, 429
    TimePicker, 428
interactivity, handling, 775-776
items controls
    ComboBox, 277-279
    DisplayMemberPath property, 552
    FlipView, 287
items panels, 274-276
ListBox, 272, 279-281
ListView, 281-285
MenuFlyout, 292-293
PasswordBox, 324
PlayingCard control
making "lookless," 765-768
visual states, defining, 769-770
popup controls, 411
CoreWindowDialog control, 411-413
CoreWindowFlyout control, 414-416
Hub control, 421-426
MessageDialog control, 416
Popup control, 418-420
PopupMenu control, 417-418
range controls, 401
ProgressBar, 402
Slider, 402-404
RichEditBox, 321-324
RichTextBlock, 308-309
embedding UIElements, 309-310
text overflow, 310-313
SearchBox, 404-405
history suggestions, providing, 406
local content suggestions, providing, 407-409
query suggestions, providing, 407
result suggestions, providing, 409
search results page, adding, 405
SemanticZoom, 289-292
templated controls, creating, 771-774
TextBlock, 295-297
Inlines property, 301-302
IsColorFontEnabled property, 300-301
OpticalMarginAlignment property, 299-300
properties, 297
Runs, 302-303
Spans, 304-305
text selection, 306-307
TextLineBounds property, 298-299
TextPointer class, 307
TextReadingOrder property, 300
underlining, 302
whitespace, 303
TextBox, 313
input scope, 317-319
MaxLength property, 314
multiline mode, 314
PreventKeyboardDisplayOnProgrammaticFocus property, 313
spell check functionality, 315-316
text prediction, 315-316
text selection, 316
ToggleSwitch, 429-430
UI automation, supporting, 776
user controls
consuming, 759
creating, 756-758
user interface, creating, 761-763
visual states, responding to changes in, 534-537
WebView, 430
HTML, composing with XAML, 433-434
JavaScript, composing with C#, 434-435
navigation, 431-433
web content, capturing, 435-436
XAML, 249
cookies (HTTP), retrieving data over networks, 587
CoreDispatcher property, 199-200
CoreWindowDialog control, 411-412
back button command, 413
custom commands, adding, 412-413
CoreWindowFlyout control, 414
commands, 415
formatting, 416
crashes, handling, 178
CreateAsync method, 351
CreateForTranscoding method, 356
creating
temlates, 771-774
cookies (HTTP), retrieving data over networks, 587
CoreDispatcher property, 199-200
CoreWindowDialog control, 411-412
back button command, 413
custom commands, adding, 412-413
CoreWindowFlyout control, 414
commands, 415
formatting, 416
crashes, handling, 178
CreateAsync method, 351
CreateForTranscoding method, 356
creating
core controls
behavior, 759, 761
templated controls, 771-774
user interface, 761-763
HelloRealWorld project, 9-11
SimpleCanvas, 785-787, 789
SimpleStackPanel, 789-792
UniformGrid, 792, 794-797
user controls, 756
  behavior, 757-758
  user interface, 756-757
WindowsRuntime component, 776-778
cropping images, 358-359, 683
curly braces, escaping, 50
CurrentAppSimulator class, 235-240
custom animations
  data types, 486-487
  dependent versus independent animations, 488-489
  duration of, 489, 495
  easing functions, 505, 508-509
  From property, 490-492
  keyframes, 500
    discrete keyframes, 503-504
    easing keyframes, 504
    linear keyframes, 500-501
    spline keyframes, 501-502
  property paths, 496-498
Storyboards
  multiple animations, 495-496
  Timeline properties, 498-500
Timeline properties
  AutoReverse, 493
  BeginTime, 493
  FillBehavior, 495
  RepeatBehavior, 494
  SpeedRatio, 493
To property, 490-492
custom background tasks, 655-657
  conditions, 658-659
  triggers, 657-658
custom Bluetooth devices, 689. See also custom Bluetooth Smart devices
  communicating with the device, 691
  connecting to the device, 690-691
  declaring device capability, 689
custom Bluetooth Smart devices
  communicating with the device, 694
  declaring device capability, 692-693
custom buttons, adding to MessageDialog control, 416
custom catalogs, managing, 231-232
custom commands, adding to CoreWindowDialog control, 412-413
custom HID devices, 698
  communicating with the device, 699-700
  connecting to the device, 699
  declaring the device capability, 698
custom media formats, 377-380
custom panels, creating, 781
custom USB devices
  bulk transfers, 697
  connecting to the device, 696
  control transfers, 696
  declaring device capability, 695
  interrupt transfers, 697
custom Wi-Fi Direct devices, 701-702
customizing
  app launch, 192-193
  collection views
    groups, 562-565
    navigation, 565
  current value display (Slider control), 404
  data updates flow in data binding, 548-549
  logo images in Visual Studio, 13-16
  media playback, 370-372
  PDF page rendering, 362-364
  projection (Devices charm), 620
  Search pane, 599-600, 603
  splash screen in Visual Studio, 13
CustomResource markup extension, 523

dark app themes, 242
data
  app data, 573-576
  user data, 579
    file picker, 580
    folder picker, 580-581
    known folders, accessing, 582-583
    libraries, managing, 583-584
data binding, 526, 545

- Binding to collections, 550-553, 562-565
- To plain properties, 548
- Controlling rendering, 554
- Data templates, 554-557
- Template selectors, 558
- Data flow, customizing, 548-549
- In C#, 547
- RelativeSource property, 547
- Rendering, improving performance of, 566-569
- Source object, designating as data context, 549-550
- Source property, 546
- Target property, 546
- Value converters, 558-556

data context, designating source object as, 549-550

data packages, 604

data templates, controlling rendering in data binding, 554-557

data virtualization, 284

DataContext property (Binding), 549-550

date and time controls
- DatePicker, 426-427
- ProgressRing, 429
- TimePicker, 428

DatePicker control, 426-427

debugging JavaScript runtime exceptions, 433

Declarations tab (Visual Studio), 21

decoders
- BitmapProperties, reading, 349-350
- Enumerating, 343

decoding images, 342-343

default values, 122

deferrals, 180

defining
- Ad units, 214-216
- Async method with void return type, 198

dependency default values, 122

dependency properties, 117-118
- Attached properties, 123-124
- Change notification, 120

comparing with routed events, 126
example of, 118-120
hijacking in control templates, 531-532
local values, 122-123
multiple provider support, 122
property value inheritance, 120-121
property wrappers, 119

DependencyProperty.Register method, 119

dependent animations, 470, 488-489

designating source object as data context, 549-550

device capabilities, 19-20

device protocol APIs, 679

device scenario APIs, 679

- Barcode scanners, 684
  - Claiming the device, 685-686
  - Enabling the device, 686
  - Retrieving the data, 686-687
- Custom Bluetooth devices, 689
  - Communicating with the device, 691
  - Connecting to the device, 690-691
  - Declaring device capability, 689
- Custom Bluetooth Smart devices
  - Communicating with the device, 694
  - Declaring device capability, 692-693
- Custom USB devices
  - Bulk transfers, 697
  - Connecting to the device, 696
  - Control transfers, 696
  - Declaring device capability, 695
  - Interrupt transfers, 697
- Custom Wi-Fi Direct devices, 701-702

- Fingerprint readers, 680

- HIDs, 698
  - Communicating with the device, 699-700
  - Connecting to the device, 699
  - Declaring the device capability, 698
- Image scanners, 680-681
  - Previews, displaying, 683
- Scan sources, configuring, 682
- Scans, performing, 681-682
- Magnetic stripe readers, 687-688
Devices charm, 611
  Play To feature, 618-620
  printing, 611-614
    changing default options, 616
    custom options, adding, 617
    displayed options, changing, 617
    projections, customizing, 620
dictionaries, 54-55
dimensions of pixels, retrieving, 343-344
direct routing, 125
Direct2D, 741-744
Direct3D, 744-746
DirectX, 2
  integration with XAML UI Framework, 739-740
  as image source, 740
  swap chain, integrating, 747-751
  using Direct2D content, 741-744
  using Direct3D content, 744-746
discovering
  orientation, 88
  window size, 84-85
discrete keyframes, 503-504
Dispatcher property, 199-200
DispatcherTimer class, 509-511
displaying
  catalog, 227-228
  dynamic pixel content, 334-336
  multiple windows, 200-203
  preview of media capture, 384-386
  relevant ads, 219
  toast notifications, 727-728
DisplayMemberPath property
  (item controls), 552
downloading files, BackgroundDownloader, 589-591
drag-and-drop operations
  pointers, capturing, 138
  reordering items, 284
dragging, recognizing, 158
DragItemThemeAnimation, 485
DragOverThemeAnimation, 485
drop-downs, 278
DropTargetItemThemeAnimation, 485
durable products, 225
  catalog, displaying, 227-228
  purchased products, identifying, 226
  purchases, initiating, 226-227
duration of animations, 489, 495
dynamic images, generating
  with RenderTargetBitmap, 337
  with WriteableBitmap, 334-336
easing functions, 505, 508-509
easing keyframes, 504
EasingFunction, 505
EdgeUIThemeTransition, 475
effects, adding to media, 373-374, 397
ElasticEase function, 508
elements
  named elements in templates, 542
  theme transitions, applying, 470-471
  visual trees, 125-126
ElementTheme property, 242
Ellipse, 439
embedding
  Frames, 209-210
  UIElements with RichTextBlock, 309-310
emoji symbols, 300
EnableDependentAnimation property, 488
encoding
  images, 351
    options, selecting, 353
    writing pixel data, 353-354
  transcoding, 356-359
EntranceThemeTransition, 472
enumerating decoders, 343
ErrorOccurred event (AdControl), 219
escaping curly braces, 50
event triggers, 483-484
events
  ErrorOccurred (AdControl), 219
  IsEngagedChanged (AdControl), 218
  keyboard events, 169-171
OrientationChanged, 88
pointer events, 137
  handlers, 137
  keyboard modifiers, 172-174
routed events, 124
  bubbling, 127-128
  comparing with dependency properties, 126
  handlers, 127
  Silverlight, 129
  visual tree, 125-126
SelectionChanged, 272
example of dependency properties, 118-120
execution states of apps, 176
expiring durable in-app purchases, 214
explicit Runs, 303
explicit sizes, 65
ExponentialEase function, 508
extensibility, attached properties, 124

F
FadeInThemeAnimation, 484
FadeOutThemeAnimation, 484
feature reports (HID devices), 700
feature-differentiated trials, 220-222
features of Windows Store, testing, 235
FeederConfiguration property (ImageScanner), 682
File Associations declarations, adding to package manifest, 634-635
file capabilities, 19
File Open Picker contract, 637-640
File Save Picker contract, 641-642
files, 572
  app files, 576
    local files, 577
    roaming files, 577
    temporary files, 578
downloading, 589-591
saving to local file system, 573
user data, 579-580
FillBehavior property (Timeline), 495
FillRule property (PathGeometry), 447-448
filters (HTTP), 586-587
fingerprint readers, 680
FlatbedConfiguration property (ImageScanner), 682
FlipView items control, 287
flow direction, 69
FlowDirection property, 71
flyouts, 245-247
focus, 169
focus rectangle, 173
folder picker, 580-581
folders, 572
  known folders, accessing, 582-583
  libraries, managing, 583-584
FontIcon content controls, 251
FontStretch property (TextBlock), 295
FontWeight property (TextBlock), 295
format (audio/video), changing, 395
formatting CoreWindowFlyout control, 416
Frames, embedding, 209-211
free trials
  feature-differentiated trials, 220-222
  time-based trials, 220
freenium business model, 213
From property (custom animations), 490, 492
full licenses, 222
  listing details, obtaining, 223
  purchase dialog, launching, 223
functions, easing functions, 505, 508

generating dynamic images
  with RenderTargetBitmap, 337
  with WriteableBitmap, 334-336
gerofencing, 670-674
Geolocator class, retrieving current location, 666-668
geometries
GeometryGroup, 448
parameters, syntax, 452
PathSegments, 445-447
representing as strings, 449-451
subclasses, 444
GeometryGroup, 448-449
Geoposition, 668
gesture recognizers, 145-146. See also gestures
dragging, 158
EdgeGesture class, 150
GestureRecognizer class, 146-149
gestures, 145-146
EdgeGesture class, 150
GestureRecognizer class, 146-149
manipulations, 151-156
swipes, recognizing, 150
tap and send, 674
zooming, 289-292
GET requests, 585-586
GetAsyncKeyState method, 170
GetFolderAsync method, 583
GetForCurrentView methods, 204
GetKeyState method, 170
GetPixelDataAsync method, 343
GetScaledImageAsThumbnailAsync method, 573
globalization, 30-32
.xlf files, 33-34
pseudo languages, 33-34
GoBack method, 206
GoForward method, 206
GPU (graphics processing unit), hardware acceleration, 470, 488-489
graphics. See also vector graphics
immediate-mode systems, 466
retained-mode systems, 466
Grid panel, 93-97
background color, applying, 100
columns and rows, sizing, 98-100
comparing to other panels, 100
percentage sizing support, 99
GridView items control, 285
GridView incremental rendering, 566-569
grouping items in collections, 562-565
gyrometer, reading from, 665
H
halting bubbling, 127-128
handlers. See also handling
for pointer events, 137
for routed events, 127
handling
content overflow, 105
with clipping, 105-106
with scaling, 113, 115
with scrolling, 107, 109-111
crashes, 178
suspending events, 180
handwriting
recognizing, 166-167
rendering, 161-165
hardcoded colors, 38
hardware, ASHWID, 234
hardware acceleration, 470, 488-489
Header property (ListView control), 282
height
Auto length, 65
controlling, 64-66
minimum window height, selecting, 86
HelloRealWorld project
application definition, 26-29
creating, 9-12
globalization, 30-34
localization, 30-31, 33-34
main page logic, 24-25
main page user interface, 22-24
Narrator, 37
hero images, 421
HID (Human Interface Device) devices
communicating with device, 699-700
connecting to device, 699
declaring device capability, 698
high contrast themes, 38-39
hijacking, dependency properties in control templates, 531-532
history suggestions, providing to SearchBox users, 406
hit testing, 141-142
hittability, 142
homeregion-xxx resource qualifier, 340
HorizontalAlignment property, 69
HorizontalContentAlignment property, 70
HTML (Hypertext Markup Language), 2, 433-434
HTTP requests, 584
caching, 587
cookies, 587
filters, 586-587
GET requests, 585-586
prefetching, 588
Hub control, 421-422
HubSection property, 423-424
Orientation property, 422
SectionHeaders property, 425-426
sections, 422
HubSection property (Hub control), 423-424
HyperlinkButton content controls, 254-255

IBuffer, 334
IconElement, 248
identifying
hardware, ASHWID, 234
purchased products, 226
identity capabilities, 20
IID_PPV_ARGS, 740
ILDASM (IL Disassembler), 3
Image element (XAML), 327-328
decoding images, 342-343
dynamic images, generating
with RenderTargetBitmap, 337
with WriteableBitmap, 334-336
nine-grid, 331-334
referencing files
in app data, 330
with URLs, 328-330
scaling, 337-338
file variations, loading automatically, 338-339
file variations, loading manually, 340
image scanners, 680
previews, displaying, 683
scan sources, configuring, 682
scans, performing, 681-682
image-based large templates, 715
ImageBrush, 460-461
ImageProperties, reading and writing from a file, 347-349
images
cropping, 358-359
decoders
BitmapProperties, reading, 349-350
enumerating, 343
decoding, 342-343
dynamic images, generating
with RenderTargetBitmap, 337
with WriteableBitmap, 334-336
encoding, 351-353
hero images, 421
metadata
BasicProperties, 347
ImageProperties, 347-349
writing, 355-356
PDF pages, rendering, 359-360
photos
camera settings, adjusting, 388-389
capturing, 381-383, 387
pixels
transforming, 346
writing data, 353-354
scaling, 337-338
file variations, loading automatically, 338-339
file variations, loading manually, 340
stretching with nine-grid, 331-334
transcoding, 356-359
ImageScanner, properties, 682
immediate-mode graphics systems, 466
implicit Runs, 303
implicit styles, 519
improving
   rendering performance, 566
      with incremental item rendering, 566-569
      with scrolling placeholders, 566-569
      screen reading, 36-37
in-app purchases, 214, 225
   consumable products
      custom catalogs, managing, 231-232
      purchases, initiating, 228-230
   durable products, 225
      catalog, displaying, 227-228
      purchased products, identifying, 226
      purchases, initiating, 226-227
in-memory data, sources of, 571
inclinometer, reading from, 665
incoming call notifications, 726-727
incremental item rendering, 566-569
independent animations, 470
   versus dependent animations, 488-489
independent input, 751-752
indeterminate ProgressBar control, 429
index markers, 373
indirection, 515
inertia, 156
inheritance
   style inheritance, 518
   TextElement class, 304
initiating purchases, 226-230
InkManager class, 161-167
Inlines property (TextBlock control), 301-302
input
   gestures, 145-146
      EdgeGesture class, 150
      GestureRecognizer class, 146-149
   independent input, 751
   keyboard input, 168-169, 171
   manipulations, 151-156
   mouse input, 156
   palm rejection, 159
   pen input, 159
      handwriting, recognizing, 166-167
      handwriting, rendering, 161-165
      properties, 159-160
   touch input, 134
      multiple pointers, tracking, 143-145
      pointer events, 137
      pointers, 134-136
      pointers, capturing, 138-143
input reports (HID devices), 699
input scope, 317-319
insert.js script, 730
   client-side code, 734-736
   server-side, 732-733
inserting properties in control templates, 526-531
integration of DirectX with XAML UI Framework
   as image source, 740
   swap chain, integrating, 747-751
   using Direct2D content, 741-744
   using Direct3D content, 744-746
interactive zooming, 116
interactivity (controls), handling, 775-776
interrupt transfers, 697
invoking software keyboard from custom controls, 321
IsAvailable property (StorageFile), 572
IsColorFontEnabled property (TextBlock control), 298-301
IsEngagedChanged event (AdControl), 218
IsFullScreen property (MediaElement), 371
IStorageltem interface, 571-572
item containers, 273
ItemControl class, 271
items controls
   ComboBox, 277-279
   DisplayMemberPath property, 552
   FlipView, 287
   Gridview, 285
   items panels, 274-276
   ListBox, 272, 279-281
ListView
  data virtualization, 284
  Header property, 282
  incremental data reordering, 285
  reordering items, 284
  SelectionMode property, 283-284
  properties, 281-282
  ScrollIntoView, 282
items panels, 274-276

J-K

JavaScript
  composing with C#, 434-435
  runtime exceptions, debugging, 433
jumping to HubSections, 425-426

keyboard
  display, changing, 317-319
  input, 168-171, 317-319
  invoking from custom controls, 321
  responding to showing/hiding, 321
keyboard modifiers in pointer events, 172-174
KeyboardCapabilities API, 317
keyframes, 500
  discrete keyframes, 503-504
  easing keyframes, 504
  linear keyframes, 500-501
  spline keyframes, 501-502
keywords (XAML), 59
killing apps, 177-178
known folders, accessing, 582-583

L

language-xxx resource qualifier, 340
large tile templates
  image-based large templates, 715
  text-only large templates, 713
launching
  apps, 182-185
  launching programatically, 190-191
    in Visual Studio, 10-12
  comparing with activation, 186
layout, 63
  arrange step, 784-785
  Auto length, 65
  content alignment, 70
  explicit sizes, 65
  FlowDirection property, 71
  height, controlling, 64-66
  Margin property, 66-68
  measure step, 782-783
  orientation, discovering, 88
  Padding property, 66-68
  panels, 84, 89
    Canvas, 89-92
    Grid, 93-100
    StackPanel, 92
    VariableSizedWrapGrid, 101-104
  parent elements, 63
  positioning, 68-69
  snap points, 111-112
  view states (Windows 8), 87
  window size
    discovering, 84-85
    minimum height, selecting, 86
    minimum width, selecting, 85
layoutdir-xxx resource qualifier, 340
libraries
  known folders, accessing, 582-583
  managing, 583-584
  Windows Runtime component, creating, 776-778
LicenseInformation class, 222
licensing, full licenses, 222-223
lifecycle of apps, 175-177
  activating apps, 185-186
  killing apps, 177-178
  launching apps, 182-185
  resuming apps, 181
  suspending apps, 178-180
  terminating apps, 181
lifting pointers, 247
light app themes, 242
light sensor, reading from, 665
linear interpolation, 487
linear keyframes, 500-501
LinearGradientBrush, 453-459
LineHeight property (TextBlock), 296
Lines, 440
LineSegment class, 445
ListBox control, 272
listing details, obtaining, 223
ListBox items control, 279
properties, 280
when to use, 281
lists, 52-53
ListView incremental rendering, 566-569
ListView items control
data virtualization, 284
Header property, 282
incremental data reordering, 285
properties, 281-282
reordering items, 284
ScrollIntoView, 282
SelectionMode property, 283-284
live regions, 37
live tiles, 703
tile templates, 704-705
large tile templates, 713, 715
medium tile templates, 706
static medium templates, 706-709
wide tile templates, 710
loading
scale variations for images, 338-340
XAML at runtime, 56
LoadListingInformation method, 224
local content suggestions, providing to
SearchBox users, 407-409
local files, 577
local settings, 574
local updates, 717
local values, 122-123
localization, 30-34
.xlf files, 33-34
pseudo languages, 33-34
location information
geofencing, 670-674
proximity, 674-675
retrieving, 666-668
lock screen, 736, 738
log images (Visual Studio), customizing, 13-16
logical products, 232
looping media playback, 371

M
magnetic stripe readers, 687-688
main page logic, HelloRealWorld project, 24-25
main page user interface, HelloRealWorld
project, 22-24
managed-to-unmanaged code transitions, 335
managing
custom catalogs, 231-232
libraries, 583-584
session state with SuspensionManager,
187-188
manipulations, 151-156
manual animations, 509-511
Margin property, 66-68
markers, 373
markup compatibility XML namespace, 47
markup extensions
curly braces, escaping, 50
procedural code, 51
syntax, 49-50
Matrix3DProjection, 82
MatrixTransform class, 78-79
MaxLength property (TextBox), 314
measure step (layout), 782-783
media. See also media players
audio
capturing, 391
speech synthesis, 397-398
SSML, 398-399
capturing, 380
custom formats, 377-380
markers, 373
Media Foundation components, 367-368

Media players
- MediaElement class, 374-375
- MediaPlayer, 376-377

MediaCapture class, 199
- audio, capturing, 391
- video, capturing, 389-390

MediaElement class
- Closed state, 372
- IsFullScreen property, 371
- Paused state, 372
- playback, 368-370
  - customizing, 370-372
  - looping, 371
- using as media player, 374-375
- video, capturing, 389-390

MediaPlayer, 376-377

MediaStreamSource class, 378

MediaPlayer, 376-377

Media Transcoder class, transcoding, 392
- adding effects, 397
- format, changing, 395
- quality, changing, 392-394
- trimming, 396

MenuFlyout, 292-293

MessageDialog control, 416

Metadata control
- BitmapProperties, reading from decoder, 349-350
- ImageProperties, reading and writing from a file, 347-349
- reading with WIC metadata query language, 350-351
- writing, 355-356

Methods
- AddAudioEffect, 373-374
- AddVideoEffect, 373-374
- asynchronous, 196-197
  - defining with void return types, 198

awaitable CreateWindowAsync, 200

awaiting, 197

CheckLicense, 221

CreateAsync, 351

CreateForTranscoding, 356

DependencyProperty.Register, 119

GetFolderAsync, 583

GetForCurrentView, 204

GetProcAddress, 170

GetPixelDataAsync, 343

GetScaledImageAsThumbnailAsync, 573

GoBack, 206

GoForward, 206

LoadListingInformation, 224

Navigate, 206

OnNavigatedFrom, 206

OnNavigatingFrom, 206

RenderToStreamAsync, 362-364

ScrollToSection, 422

SetPixelData, 353

SetSource, 369

ShowUserSelectedFile, 359-362

TryGetItemAsync, 572

UseDecoder, 351

Microsoft Advertising pubCenter
- ad units, defining, 214-216
- ads, adding to a page, 216-218

Microsoft MediaPlayer, 376-377

middle mouse button event handling, 157

minimum height, selecting, 86

minimum width, selecting, 85

MinuteIncrement property (TimePicker control), 428

modal dialog boxes, CoreWindowDialog control, 411-412
- back button command, 413
- custom commands, adding, 412-413

mouse input, 156

MouseCapabilities class, 156

MouseDevice class, 156

multicolor font support (TextBlock control), 300

multiline mode (TextBox control), 314

multiple windows, displaying, 200-203
multithreading, 195
  transitioning between threads, 199-200
multitouch input, 134
MVVM (Model-View-ViewModel) architecture, 129-130

N

named elements in templates, 542
named styles, 520
namespaces, 45-47
naming object elements, 57
Narrator, 37
Navigate method, 206
navigating
  views, 565
  between pages, 204-207
    NavigationHelper class, 208
  page caching, 207-208
  WebView control, 431-433
NavigationHelper class, 208
.NET classes, using with XAML, 55
network capabilities, 20
NetworkInformation class, 594
networking
  connection information, obtaining, 594
  data, retrieving
    BackgroundDownloader, 589-591
    HTTP prefetching, 588
    HTTP requests, 584-587
    sockets, 591
    syndication, 591-593
  peer devices, communicating with, 675-676
New Appointments Provider contract
  appointments provider sources, 649
  appointments provider targets, 650
New Contact contract, 644
  contact sources, 645-646
  contact targets, 647-648
NFC (Near Field Communication) tags, sending and receiving information, 674-675
nine-grid feature (Image element), 331-334
notifications
  push notifications, 728-729
  toast notifications, 722
    alarm notifications, 726-727
    displaying, 727-728
    responding to, 725
    templates, 723, 725

O

object elements, 44-45
  agile objects, 196
  children, processing rules, 55
  collection items, 52
    dictionaries, 54-55
    lists, 52-53
    content property, 52
    naming, 57
    visual trees, 125-126
OfferIds, 232
offers, 232
OneTime binding, 549
OneWay binding, 549
OnNavigatedFrom method, 206
OnNavigatedTo method, 206
OnNavigatingFrom method, 206
OnSuspending handler, 180
OpticalMarginAlignment property (TextBlock control), 298-300
orientation, discovering, 88
Orientation property (Hub control), 422
OrientationChanged event, 88
OrientationSensor class, 666
output reports (HID devices), 700

P

package manifest, 12
Packaging tab (Visual Studio), 21-22
Padding property, 66-68
pages, navigating between, 204-207
  NavigationHelper class, 208
  page caching, 207-208
PageThemeTransition, 475-476
PAINT.NET, 4
palm rejection, 159
panels, 63, 84, 89
  Canvas, 89-92
  content alignment, 70
  creating, 781
  FlowDirection property, 71
  Grid, 93-96
    background color, applying, 100
    comparing to other panels, 100
    percentage sizing support, 99
    rows and columns, sizing, 98-100
  items panels, 274-276
layout
  arrange step, 784-785
  measure step, 782-783
positioning, 68-69
  SimpleStackPanel, creating, 789-792
  StackPanel, 92
  VariableSizedWrapGrid, 101-104
parameters for geometries, syntax, 452
parent elements
  layout, 63
  panels, 63
  templated parents, 526
parsing XAML at runtime, 56
password reveal button, 324
PasswordBox control, 324-325
PathFigures, 445-447
PathGeometry, FillRule property, 447-448
PathIcon content controls, 252
Paths, 442
PathSegments, 445-447
Paused state (MediaElement), 372
PDF pages
  customizing rendering, 362-364
  rendering, 359-362
peek templates, 708
peek wide templates, 710
peer devices, communicating with, 675-676
pen input, 159
  handwriting
    recognizing, 166-167
    rendering, 161-165
    properties, 159-160
percentage sizing support, Grid panel, 99
performance
  cached composition, 467
  managed-to-unmanaged code transitions, 335
  of DirectX integration with XAML
    UI Framework, 739-740
  of rendering, improving, 566
    with incremental item rendering, 566-569
    with scrolling placeholders, 566-569
Performance and Diagnostics page (Visual Studio), 11
perimeters, geofencing, 670-674
permissions, 17
perspective transforms, 79-81
photos, capturing, 381-383, 387
  adjusting camera settings, 388-389
  preview, displaying, 384-386
pixels, 67
  BGRA8, 335
  dynamic pixel content, displaying, 334-336
  retrieving data from, dimensions, 343-344
  transforming, 346
  writing data, 353-354
plain properties, binding to, 548
PlaneProjection class, 79
Play To feature (Devices charm), 618-620
playback, custom media formats, 377-380
PlayingCard control
  behavior, creating, 759, 761
  consuming, 764
  creating, 759
  interactivity, handling, 775-776
  making “lookless,” 765-768
  UI automation, supporting, 776
  user interface, creating, 761-763
  visual states, defining, 769-770
point of sale devices, 684
point of service devices, 684
Pointer class, 135
pointer events, keyboard modifiers, 172-174
PointerDevice class, 135
PointerDownThemeAnimation, 484
PointerPoint class, 136
PointerPointProperties, 157
pointers, 134
capturing, 138-143
events, 137
gesture recognizers, 145-146
EdgeGesture class, 150
GestureRecognizer class, 146-149
independent input, 751
lifting, 247
multiple pointers, tracking, 143-145
Pointer class, 135
PointerDevice class, 135
PointerPoint class, 136
releasing, 138
PointerUpThemeAnimation, 484
PolyBezierSegment class, 445
Polygons, 441
PolyLines, 440
PolyLineSegment class, 445
PolyQuadraticBezierSegment class, 445
PopInThemeAnimation, 484
PopOutThemeAnimation, 484
Popup control, 418-420
popup controls, 411
CoreWindowDialog, 411
back button command, 413
custom commands, adding, 412-413
CoreWindowFlyout, 414
commands, 415
formatting, 416
Hub, 421-422
HubSection property, 423-424
Orientation property, 422
SectionHeaders property, 425-426
sections, 422
MessageDialog, 416
Popup, 418-420
PopupMenu, 417-418
PopupMenu control, 417-418
PopupThemeTransition, 474
positioning, 68-69
content alignment, 70
power easing functions, 505
PowerEase class, 506
prefetching (HTTP), retrieving data over networks, 588
PreventKeyboardDisplayOnProgrammaticFocus property (TextBox), 313
PreviousExecutionState property, 183-185
printing, 611-614
custom options, 617
default options, 616
displayed options, changing, 617
procedural code, mixing with XAML, 56
processing rules for object element children, 55
programmatically launching apps, 190-191
programming language, selecting, 2
ProgressBar control, 402
ProgressRing control, 429
projections
customizing (Devices charm), 620
Matrix3DProjection, 81
PlaneProjection, 79
projects
C#, updating, 22
HelloRealWorld
application definition, 26-29
creating, 9-11
globalization, 30-34
localization, 30-34
main page logic, 24-25
main page user interface, 22-24
package manifest, 12
Split App, 204
XAML, updating, 22
proga snippet (Visual Studio), 123
properties
ClockIdentifier (TimePicker control), 428
Content property (content controls), 52, 244
CoreDispatcher, 199
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>data binding</td>
<td>545</td>
</tr>
<tr>
<td>binding to collections</td>
<td>550-553</td>
</tr>
<tr>
<td>data flow, customizing</td>
<td>548-549</td>
</tr>
<tr>
<td>in C#</td>
<td>547</td>
</tr>
<tr>
<td>RelativeSource</td>
<td>547</td>
</tr>
<tr>
<td>source object, designating as data object</td>
<td>549-550</td>
</tr>
<tr>
<td>source property</td>
<td>546</td>
</tr>
<tr>
<td>target property</td>
<td>546</td>
</tr>
<tr>
<td>to plain properties</td>
<td>548</td>
</tr>
<tr>
<td>value converters</td>
<td>558-562</td>
</tr>
<tr>
<td>views</td>
<td>562-565</td>
</tr>
<tr>
<td>dependency default values</td>
<td>122</td>
</tr>
<tr>
<td>dependency properties</td>
<td>117-118</td>
</tr>
<tr>
<td>attached properties</td>
<td>123-124</td>
</tr>
<tr>
<td>change notification</td>
<td>120</td>
</tr>
<tr>
<td>comparing with routed events</td>
<td>126</td>
</tr>
<tr>
<td>example of</td>
<td>118-120</td>
</tr>
<tr>
<td>local values</td>
<td>122-123</td>
</tr>
<tr>
<td>multiple provider support</td>
<td>122</td>
</tr>
<tr>
<td>property value inheritance</td>
<td>120-121</td>
</tr>
<tr>
<td>property wrappers</td>
<td>119</td>
</tr>
<tr>
<td>ElementTheme</td>
<td>242</td>
</tr>
<tr>
<td>FillRule</td>
<td>447-448</td>
</tr>
<tr>
<td>FlowDirection</td>
<td>71</td>
</tr>
<tr>
<td>for Slider control</td>
<td>403</td>
</tr>
<tr>
<td>HorizontalAlignment</td>
<td>69</td>
</tr>
<tr>
<td>HorizontalContentAlignment</td>
<td>70</td>
</tr>
<tr>
<td>HubSection</td>
<td>423-424</td>
</tr>
<tr>
<td>IsFullWindow</td>
<td>371</td>
</tr>
<tr>
<td>Lines</td>
<td>440</td>
</tr>
<tr>
<td>ListBox items control</td>
<td>280</td>
</tr>
<tr>
<td>ListView items control</td>
<td>281-282</td>
</tr>
<tr>
<td>Header</td>
<td>282</td>
</tr>
<tr>
<td>SelectionMode</td>
<td>283-284</td>
</tr>
<tr>
<td>Margin</td>
<td>66-68</td>
</tr>
<tr>
<td>MaxLength</td>
<td>314</td>
</tr>
<tr>
<td>MinutIncrement</td>
<td>428</td>
</tr>
<tr>
<td>NineGrid</td>
<td>333-334</td>
</tr>
<tr>
<td>Orientation</td>
<td>422</td>
</tr>
<tr>
<td>Padding</td>
<td>66-68</td>
</tr>
<tr>
<td>for pen input</td>
<td>159-160</td>
</tr>
<tr>
<td>PointerPoint class</td>
<td>136</td>
</tr>
<tr>
<td>PreventKeyboardDisplayOnProgrammaticFocus</td>
<td>(TextBox)</td>
</tr>
<tr>
<td>PreviousExecutionState, reacting to</td>
<td>183-185</td>
</tr>
<tr>
<td>ProgressBar control</td>
<td>402</td>
</tr>
<tr>
<td>RequestedTheme</td>
<td>242</td>
</tr>
<tr>
<td>RoutedEvent</td>
<td>484</td>
</tr>
<tr>
<td>SectionHeaders</td>
<td>425-426</td>
</tr>
<tr>
<td>Selector subclass</td>
<td>271</td>
</tr>
<tr>
<td>SpreadMethod</td>
<td>455</td>
</tr>
<tr>
<td>Style, setters</td>
<td>515-516</td>
</tr>
<tr>
<td>TargetName</td>
<td>482</td>
</tr>
<tr>
<td>TextBlock</td>
<td>295</td>
</tr>
<tr>
<td>CharacterSpacing</td>
<td>296</td>
</tr>
<tr>
<td>Inlines</td>
<td>301-302</td>
</tr>
<tr>
<td>IsColorFontEnabled</td>
<td>298, 300-301</td>
</tr>
<tr>
<td>LineHeight</td>
<td>296</td>
</tr>
<tr>
<td>OpticalMarginAlignment</td>
<td>298-300</td>
</tr>
<tr>
<td>TextAlignment</td>
<td>296</td>
</tr>
<tr>
<td>TextLineBounds</td>
<td>298-299</td>
</tr>
<tr>
<td>TextReadingOrder</td>
<td>298-300</td>
</tr>
<tr>
<td>TextTrimming</td>
<td>296</td>
</tr>
<tr>
<td>TextWrapping</td>
<td>296</td>
</tr>
<tr>
<td>Timeline</td>
<td>498-500</td>
</tr>
<tr>
<td>VerticalAlignment</td>
<td>69</td>
</tr>
<tr>
<td>VerticalContentAlignment</td>
<td>70</td>
</tr>
<tr>
<td>property elements</td>
<td>47-48</td>
</tr>
<tr>
<td>property paths</td>
<td>271, 496-498</td>
</tr>
<tr>
<td>property value inheritance</td>
<td>120-121</td>
</tr>
<tr>
<td>property wrappers</td>
<td>119</td>
</tr>
<tr>
<td>proportional sizing</td>
<td>98</td>
</tr>
<tr>
<td>Protocol contract</td>
<td>636-637</td>
</tr>
<tr>
<td>providers</td>
<td></td>
</tr>
<tr>
<td>dependency default values</td>
<td>122</td>
</tr>
<tr>
<td>dependency properties</td>
<td></td>
</tr>
<tr>
<td>attached properties</td>
<td>123-124</td>
</tr>
<tr>
<td>change notification</td>
<td>120</td>
</tr>
<tr>
<td>example of</td>
<td>118-120</td>
</tr>
<tr>
<td>local values</td>
<td>122-123</td>
</tr>
<tr>
<td>multiple provider support</td>
<td>122</td>
</tr>
<tr>
<td>property value inheritance</td>
<td>120-121</td>
</tr>
<tr>
<td>property wrappers</td>
<td>119</td>
</tr>
</tbody>
</table>
providing suggestions to SearchBox users, 405
  history suggestions, 406
  local content suggestions, 407-409
  query suggestions, 407
  result suggestions, 409
proximity information, retrieving, 674-675
pseudo languages, 33-34
pubCenter, 214
  ad units, defining, 214-216
  ads, adding to a page, 216-218
pull updates, 717-718
purchase dialog, launching, 223
purchases
  consumable products
    custom catalogs, managing, 231-232
    initiating, 228-230
    identifying, 226
  in-app purchases
    consumable products, 228-232
    durable products, 225-228
    initiating, 226-227
Push Notification wizard, 729
push notifications, 718, 728-729

Q
quadric Bézier curves, 445
QuadraticBezierSegment class, 445
quality (audio/video), changing, 392-394
query suggestions, providing to SearchBox users, 407

R
RadioButton content controls, 258-259
range controls, 401
  ProgressBar, 402
  Slider, 402
    current value display, customizing, 404
    properties, 403
reacting to PreviousExecutionState property, 183-185
reading
  BitmapProperties from decoder, 349-350
  from accelerometer, 661-662
    shake detection, 664
    tossing motion, 663-664
  from compass, 665
  from gyrometer, 665
  from inclinometer, 665
  from light sensor, 665
  from SimpleOrientationSensor API, 666
    metadata
      BasicProperties, 347
      ImageProperties, 347-349
      with WIC metadata query language, 350-351
reading order of text, 300
receipts, validating, 232-234
receiving NFC tag information, 674-675
recognizing
  gestures, swipes, 150. See also gesture recognizers
  handwriting, 166-167
Rectangles, 438-439
referencing files
  in app data, 330
  with URLs, 328-330
refresh behavior (AdControl), 218
regions (lock screen), 736
RelativeSource property (Binding), 547
releasing pointers, 138
relevant ads, displaying, 219
rendering
  controlling in data binding, 554
    data template, 554-557
    template selectors, 558
  handwriting, 161-165
  improving performance of, 566
    with incremental item rendering, 566-569
    with scrolling placeholders, 566-569
  PDF pages, 359-364
RenderTargetBitmap, generating dynamic images, 337
RenderToStreamAsync method, 362-364
RenderTransformOrigin property, 71-72
reordering items, 284
ReorderThemeTransition, 480
RepeatBehavior property (Timeline), 494
RepeatButton content controls, 256
reports, HID devices, 700
RepositionThemeAnimation, 484
RepositionThemeTransition, 478-479
representing geometries as strings, 449-451
RequestedTheme, 242
resource packages, 341
resource qualifiers, 338
  contrast, 339
  language-xxx, 340
responding
  to changes in visual states, 534-537
  to clicked toast notifications, 725
result suggestions, providing to SearchBox users, 409
resuming apps, 181
retained-mode graphics systems, 466
retrieving
  accelerometer data, 662
  shake detection, 664
  tossing motion, 663-664
  compass data, 665
data over networks
  BackgroundDownloader, 589-591
  HTTP prefetching, 588
  HTTP requests, 584-587
  sockets, 591
  syndication, 591-593
  gyrometer data, 665
  inclinometer data, 665
  light sensor data, 665
data location information, 666-668
  pixel data, dimensions, 343-344
  proximity information, 674-675
  SensorOrientationSensor API data, 666
  session state information, 188
RichTextBox control, 321-324
RichTextBlock control, 308
  text overflow, 310-313
  UIElements, embedding, 309-310
RichTextBlockOverflow element, 310-313
right-to-left environments, and
  VariableSizedWrapGrid panel, 104
roaming files, 577
roaming settings, 575-576
RotateTransform, 73
routed events, 124
  bubbling, 127-128
  comparing with dependency properties, 126
  handlers, 127
  Silverlight, 129
  visual tree, 125-126
RoutedEvent property (Storyboards), 484
rows, sizing in Grid panel, 98-100
RTF files, handling with RichTextBox control, 323
Runs, 301-303
saving files to local file system, 573
scale resource qualifier, 338
ScaleTransform, 74-75
scaling
  handling content overflow, 113-115
  Image element, 337-338
  file variations, loading automatically, 338-339
  file variations, loading manually, 340
scheduled updates, 717
screen reading, improving, 36-37
scrolling
  GridView, 566-569
  handling content overflow, 107-111
  ListView, 566-569
  snap points, 111-112
scrolling placeholders, 566-569
ScrollToDevice method, 422
ScrollView, 111, 116
Search charm, 597-598
Search pane
  customizing, 599-600
  showing programmatically, 603
  Windows 8 Search Contract, 598-599
search results page, adding to SearchBox control, 405
SearchBox control, 404
  history suggestions, providing, 406
  local content suggestions, providing, 407-409
  query suggestions, providing, 407
  result suggestions, providing, 409
  search results page, adding, 405
secondary tiles, 703, 720-722
SectionHeaders property (Hub control), 425-426
sections (Hub control), 422
  HubSection property, 423-424
  jumping to, 425-426
Segoe UI Emoji, 300
selecting
  image encoding options, 353
  programming language, 2
  text
    with TextBlock control, 306-307
    with TextBox control, 316
window size
  minimum height, 86
  minimum width, 85
selection boxes, 277
SelectionChanged event, 272
SelectionMode property (ListView control), 283-284
Selector class, 271
selectors
  ComboBox, 277-279
  FlipView, 287
  GridView, 285
  ListBox, 279-280
    properties, 280
    when to use, 281
ListView
  data virtualization, 284
  Header property, 282
  incremental data reordering, 285
  properties, 281-282
  reordering items, 284
  ScrollIntoView, 282
  SelectionMode property, 283-284
style selectors, 516-517
template selectors, controlling rendering in data binding, 558
SemanticZoom control, 289-292
sending NFC tag information, 674-675
sensor APIs, SimpleOrientationSensor, 666
sensors
  accelerometer
    reading from, 661-662
    shake detection, 664
tossing motion, reading, 663-664
compass, reading from, 665
gyrometer, reading from, 665
inclinometer, reading from, 665
light sensor, reading from, 665
separators for query suggestions, 410
server-side code, 730-733
session state
  managing with SuspensionManager, 187-188
  retrieving values, 188
SetPixelData method, 353-354
SetSource method, 369
Setters, 515-516
Settings Charm, 621-624
shaking detection (accelerometer), 664
Shapes, 438. See also geometries
  Ellipse, 439
  Lines, 440
  Paths, 442
  Polygons, 441
  PolyLines, 440
  Rectangle, 438-439
  Stroke property, 442-444
Share charm, 603-610
Share contract, 627
share sources, 603
share targets, 607-610
SharpDX, 740
ShowScrollingPlaceholders, 566-569
ShowUserSelectedFile method, 359-362
siedloaed apps, 221
Silverlight, routed events, 129
SimpleCanvas, creating, 785-789
SimpleOrientationSensor API, 666
SimpleStackPanel, creating, 789-792
simulator, 10
   CurrentAppSimulator class, 236-238
automated testing, 239
SineEase function, 508
sizing
   absolute sizing, 98
autosizing, 98
Grid panel, percentage sizing support, 99
height
   Auto length, 65
controlling, 64-66
Margin property, 66-68
Padding property, 66-68
proportional sizing, 98
SkewTransform, 76
SkyDrive, 572
Slider control, 402
   current value display, customizing, 404
properties, 403
snap points, 111-112
snap-point enabled ScrollViewers, 109
snippets, propa (Visual Studio), 123
sockets, retrieving data over networks, 591
software keyboard, 317
   input scope, 317-319
invoking from custom controls, 321
responing to showing/hiding, 321
SolidColorBrush, 452-453
source property (data binding), 546
Spans, 304-305
speech synthesis, 397-399
SpeechSynthesisStream, 397-398
SpeechSynthesizer, 398-399
speeding up/down playback, 371
SpeedRatio property (Timeline), 493
spell check functionality (TextBox control), 315-316
splash screen (Visual Studio), customizing, 13
SplashScreen property, 183
spline keyframes, 501-502
Split App project, 204
SplitCloseThemeAnimation, 485
SplitOpenThemeAnimation, 485
SpreadMethod property
   (LinearGradientBrush), 455
SSML (Speech Synthesis Markup Language), 398-399
StackPanel panel, 92
star syntax, 98
states of apps, 176
static image-based wide templates, 710
static medium templates, 706-709
StaticResource markup extension, 521-522
stereoscopic 3D video, 369
sticky buttons, 256
StorageFile class, 572
StorageFolder class, 572
StorageLibrary class, 583
Storyboards, 481-482
   RoutedEvent property, 484
   TargetName property, 482
   TargetProperty, 496-498
   Timeline properties, 498-500
   VisualStates, 483
   with multiple animations, 495-496
strecthing images, nine-grid feature (Image
element), 331-334
Stretch alignment, 69
strings, representing geometries as, 449-451
StrokeDashArray property, 442
StrokeDashCap property, 442
StrokeEndLineCap property, 442
StrokeLineJoin property, 442
strokes, 442-444
StrokeStartLineCap property, 442
Style property, setters, 515-516
styles, 514
  base TargetType, 517
  BaseTextBlockStyle, 518
  implicit styles, 519
  indirect, 515
  inheritance, 518
  named styles, 520
  selectors, 516-517
  templates, setting inside, 532-533
  theme resources, 520-523
  theme style, 771

subclasses
  of Geometries, 444
  Selector, 271
  WriteableBitmap, generating dynamic images, 334-336

submitting apps to Windows Store, 40-41

suggestions, providing for SearchBox users, 405
  history suggestions, 406
  local content suggestions, 407-409
  query suggestions, 407
  result suggestions, 409

SurfaceImageSource class, 740

suspected apps, 175

suspending event, handling, 180

SuspensionManager, managing session state, 187-188

SuspensionManager class, 188-189

swap chains, integrating, 747-751

SwapChainBackgroundPanel element, 751

SwapChainPanel element, 749

SwipeBackThemeAnimation, 485

SwipeHintThemeAnimation, 485

swipes, recognizing, 150

SymbolIcon content controls, 249-250

syndication, retrieving data over networks, 591-593

syntax
  geometry parameters, 452
  markup extensions, 49-50
  MatrixTransform class, 79
  property elements, 48
  star syntax, 98

tabs (Visual Studio)
  Application tab, 12-16
  Capabilities tab, 19
    device capabilities, 19-20
    file capabilities, 19
    identity capabilities, 20
    network capabilities, 20
  Content URIs tab, 21
  Declarations tab, 21
  Packaging tab, 21-22

tap and send, 674

tapping versus clicking, 244

target property (data binding), 546

TargetName property (Storyboards), 482

TargetProperty, 498

targetsize-xxx resource qualifier, 340

tasks
  background audio, 651-654
  custom background tasks, 655-657
  conditions, 658-659
  triggers, 657-658

template selectors, controlling rendering in data binding, 558

templated controls, 755
  behaviors, creating, 771-772
  creating, 771

templated controls user interface, creating, 772-774

templated parents, 526

templates, 524
  control templates, 524-525, 555
    dependency properties, hijacking, 531-532
    property values, inserting, 526-531
  data templates, controlling rendering in data binding, 554-557
  named elements, 542
  setting inside Styles, 532-533

tile templates, 704-705
  large tile templates, 713, 715
  medium tile templates, 706
  static medium templates, 706-709
  wide tile templates, 710

toast templates, 723-725
temporary files, 578
terminating apps, 181
testing Windows Store features, 235
text
  reading order, 300
  Runs, 301
  selecting
    with TextBlock, 306-307
    with TextBox control, 316
  Spans, 304-305
  speech synthesis, 397-399
  underlining, 302
text prediction functionality (TextBox control), 315-316
text-only large templates templates, 713
TextAlignment property (TextBlock), 296
TextBox control, 295-296
  properties, 297
    Inlines, 301-302
    IsColorFontEnabled, 300-301
    new properties in Windows 8.1, 297
    OpticalMarginAlignment, 299-300
    TextLineBounds, 298-299
    TextReadingOrder, 300
    Runs, 302-303
    Spans, 304-305
text selection, 306-307
  TextPointer class, 307
  underlining, 302
  whitespace, 303
TextBox control, 313
  input scope, 317-319
  MaxLength property, 314
  multiline mode, 314
  PreventKeyboardDisplayOnProgrammaticFocus property, 313
  spell check functionality, 315-316
text prediction, 315-316
text selection, 316
TextElement class, 303-304
TextLineBounds property (TextBox control), 298-299
TextPointer class, 307
TextReadingOrder property (TextBlock control), 298-300
TextTrimming property (TextBlock control), 296
TextWrapping property (TextBlock control), 296
theme animations, 481, 484. See also theme transitions
  adjusting, 486
  Storyboards, 481-482
    RoutedEvent property, 484
    TargetName property, 482
    VisualStates, 483
  Timeline class, 486
theme resources, 520-523
theme style, 771
theme transitions
  AddDeleteThemeTransition, 476-478
  applying to elements, 470-471
  ContentThemeTransition, 474
  EdgeUIThemeTransition, 475
  EntranceThemeTransition, 472
  PageThemeTransition, 475-476
  PopupThemeTransition, 474
  ReorderThemeTransition, 480
  RepositionThemeTransition, 478-479
ThemeResource markup extension, 521-522
themes, 242
  app themes, flyouts, 247
  high contrast, 38-39
Thickness class, 66-68
third-party payment systems, Windows Store, 213
ThreadPoolTimer class, 509-511
threads, 195
  ASTA threads, 196
  transitioning between, 199-200
ticks (Slider control), 402
tile brushes, 459
  ImageBrush, 460-461
  WebViewBrush, 461-463, 466
tiles
  badges, 718-720
  live tiles, 703
  secondary tiles, 703, 720-721
templates, 704-705
  large tile templates, 713, 715
medium tile templates, 706
static templates, 706-709
wide tile templates, 710
updating, 716
  local updates, 717
pull updates, 717-718
push notifications, 718
scheduled updates, 717
time, duration of custom animations, 489.
See also date and time controls
time-based trials, 220
Timeline class, 486
Timeline properties (custom animations), 498-500
  AutoReverse, 493
BeginTime, 493
FillBehavior, 495
RepeatBehavior, 494
SpeedRatio, 493
TimePicker control, 428
To property (custom animations), 490-492
toast notifications, 722
  alarm notifications, 726-727
displaying, 727-728
responding to, 725
templates, 723-725
ToggleButton content controls, 256
ToggleSwitch control, 429-430
tombstoning, 193
toolTip content controls, 259-262
top app bar, 262
tossing motion, reading from accelerometer, 663-664
touch input, pointers, 134
  capturing, 138-143
events, 137
multiple pointers, tracking, 143-145
Pointer class, 135
PointerDevice class, 135
PointerPoint class, 136
tracking multiple pointers, 143-145
transcoding, 356-359, 392
effects, adding, 397
format, changing, 395-396
quality, changing, 392-394
TransformGroup class, 78
transforming pixels, 346
transforms
  3D transforms, 79-81
  combining, 77
    CompositeTransform class, 77
    MatrixTransform class, 78
    TransformGroup class, 78
  RotateTransform, 73
  ScaleTransform, 74-75
  SkewTransform, 76
  TranslateTransform, 77
TransitionCollections, applying to elements, 471
transitioning between threads, 199-200
transitions, 176, 537-542
TranslateTransform, 77
trimming media files, 396
TryGetItemAsync method, 572
tunneling, 125
tweaking theme animations, 486
TwoWay binding, 549
type converters, 48-49
typed styles, 519

U

UI automation, custom control support for, 776
UI framework pairings, 1
UI threads, 195-196
UI virtualization, 274
UIElements
  embedding with RichTextBlock, 309-310
  hit testing, 140, 142
  RenderTransform property, 71
  RenderTransformOrigin property, 71-72
underlining with TextBlock, 302
UniformGrid, creating, 792-797
UpdateSourceTrigger property (Binding), 549
updating
C#, 22
tiles, 716
  local updates, 717
  pull updates, 717-718
  push notifications, 718
  scheduled updates, 717
XAML, 22
URIs
  apps, launching, 190-191
  referencing files with, 328-330
USB devices
  bulk transfers, 697
  connecting to the device, 696
  control transfers, 696
  declaring device capability, 695
  interrupt transfers, 697
UseDecoder method, 351
user controls, 755
  behavior, creating, 757-758
  consuming, 759
  creating, 756
  user interface, creating, 756-757
user data, 579
  file picker, 580
  folder picker, 580-581
  known folders, accessing, 582-583
  libraries, managing, 583-584
user interfaces, creating
  for PlayingCard control, 761-764
  for templated controls, 772-774
user themes, 242
UserConsentVerifier class, 680
UserInformation class, 630
V
validating Windows Store receipts, 232-234
value converters, 558-562
VariableSizedWrapGrid panel, 101-104
vector graphics
  Brushes, 452
    color brushes, 452-459
    tile brushes, 459-463, 466
  geometries, 444
    GeometryGroup, 448-449
    PathFigures, 445-447
    PathSegments, 445-447
    representing as strings, 449-451
    syntax, 452
Shapes, 438
  Ellipse, 439
  Lines, 440
  Paths, 442
  Polygons, 441
  PolyLines, 440
  Rectangle, 438-439
  Stroke property, 442-444
text
video
  capturing, 380, 383-386, 389-390
  custom media formats, 377-380
  effects, adding, 373-374, 397
  format, changing, 395
  index markers, 373
  markers, 373
  playback, 368-370
    customizing, 370-372
    looping, 371
  quality, changing, 392-394
  stereoscopic 3D video, 369
  trimming, 396
video stabilization, 374
view model, 556-557
view states, 87
Viewbox, 115
views, 200
   GridView, 566-569
   ListView, 566-569
   navigating, 565
   of collections, customizing, 562-565
viral compatibility, 141
virtualization
   data virtualization, 284
   UI virtualization, 274
Visual State Manager, 533-542
   visual states, responding to, 534-537
   visual transitions, 537-542
visual states, 533-534
   of PlayingCard control, defining, 769-770
   responding to changes in, 534-537
   visual transitions, 537-542
Visual Studio
   ad units, defining, 214-216
   Application tab, 12-16
   apps
      launching, 10-11
      package manifest, 12
   Capabilities tab, 19
      device capabilities, 19-20
      file capabilities, 19
      identity capabilities, 20
      network capabilities, 20
   Content URIs tab, 21
   Declarations tab, 21
   logo images, customizing, 13-16
   Packaging tab, 21-22
   Performance and Diagnostics page, 11
   propa snippet, 123
   search results page, adding to SearchBox control, 405
   simulator, 10
   splash screen, customizing, 13
   support for XAML, 57-59
visual transitions, 537-542

visual tree, 125-126
VisualState class, 534
VisualStateManager, 533-542
   visual states, responding to, 534-537
   visual transitions, 537-542
VisualStates, 483
voices
   speech synthesis, 398
   SSML, 398-399
void return type, defining async methods with, 198
VSM, 533-542
   visual states, responding to, 534-537
   visual transitions, 537-542
W
W3C (World Wide Web Consortium), 60
web content, capturing with WebView control, 435-436
WebView control, 430
   HTML, composing with XAML, 433-434
   JavaScript, composing with C#, 434-435
   navigation, 431-433
   web content, capturing, 435-436
WebViewBrush, 461-463, 466
when to use ListBox items control, 281
Wi-Fi Direct devices, 700-702
WIC (Windows Imaging Component) metadata language, reading metadata, 350-351
wide tile templates, 710
width
   Auto length, 65
   controlling, 64-66
   minimum window width, selecting, 85
window size
   discovering, 84-85
   minimum height, selecting, 86
   minimum width, selecting, 85
windows
  Frames, embedding, 209-210
  multiple windows, displaying, 200-203
  orientation, discovering, 88
  pages
    navigating between, 204-207
    NavigationHelper class, 208
    page caching, 207-208
  panels
    Canvas panel, 89-92
    Grid panel, 93-100
    StackPanel panel, 92
    VariableSizedWrapGrid, 101-104
  snap points, 111-112
Windows 8 Search Contract, 598-599
Windows 8.1
  ApplicationModel.Search namespace, 600
  resource packages, 341
  TextBlock Control properties, 297
  view states, 87
Windows Media, 367-368
Windows Runtime
  components, creating, 776-779
  IStorageItem interface, 571-572
Windows Store, 1
  apps
    compatibility with XBox controllers, 134
    submitting, 40-41
    business models, 213
    certification process, 41
    features, testing, 235
  free trials
    feature-differentiated trials, 220-222
    time-based trials, 220
  full licenses, 222
    listing details, obtaining, 223
    purchase dialog, launching, 223
    receipts, validating, 232-234
    third-party payment systems, 213
Windows.Devices.Scanners namespace, 680
Windows.Networking.Sockets namespace, 591
Windows.Storage.FileIO class, 572
Windows.Web.Syndication namespace, 591-593
WinRT (Windows Runtime), 2
WPF (Windows Presentation Foundation), 43
wrappers, property wrappers, 119
WriteableBitmap subclass, generating dynamic images, 334-336
writing metadata, 355-356
  BasicProperties, 347
  ImageProperties, 347-349
X
x:name syntax, 57
XAML, 2, 43-44
  accessibility features, 36-37
  animation, 469
  composing HTML, 433-434
  content controls, 241
  controls, 249
  Image element, 327-328
    dynamic images, generating, 334-337
    images, encoding, 351-353
    nine-grid, 331-334
    referencing files in app data, 330
    referencing files with URLs, 328-330
    writing pixel data, 353-354
  keywords, 59
  layout, 63
  loading and parsing at runtime, 56
  markup extensions, 49-51
    and procedural code, 51
    curly braces, escaping, 50
    syntax, 49-50
  MediaElement
    customizing playback, 370-372
    playback, 368-370
    mixing with procedural code, 56
  .NET classes, 55
  object elements, 44-45
    naming, 57
    processing rules for children of object elements, 55
property elements, 47-48
routed events, 124
  comparing with dependency properties, 126
  visual tree, 125-126
type converters, 48-49
updating, 22
Visual Studio’s support for, 57-59
XML namespaces, 45-47
XAML binary format, 59
XAML language namespace, 46
XBox controllers, compatibility with Windows Store apps, 134
XLIFF files, 33-34
.xlf files, 33-34
XML namespaces, 45-47
xml:lang attribute, 60
xml:space attribute, 60

Y-Z

zigzag motion, creating with keyframe animation, 501

zooming
  interactive zooming with ScrollViewer, 116
  SemanticZoom, 290-292