## Contents at a glance

*Introduction* xxvii

### PART I ELEMENTALS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Markup and Code</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>XAML Syntax</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>Basic Event Handling</td>
<td>69</td>
</tr>
<tr>
<td>4</td>
<td>Presentation with Panels</td>
<td>97</td>
</tr>
<tr>
<td>5</td>
<td>Control Interaction</td>
<td>139</td>
</tr>
<tr>
<td>6</td>
<td>WinRT and MVVM</td>
<td>193</td>
</tr>
<tr>
<td>7</td>
<td>Asynchronicity</td>
<td>221</td>
</tr>
<tr>
<td>8</td>
<td>App Bars and Popups</td>
<td>261</td>
</tr>
<tr>
<td>9</td>
<td>Animation</td>
<td>329</td>
</tr>
<tr>
<td>10</td>
<td>Transforms</td>
<td>377</td>
</tr>
<tr>
<td>11</td>
<td>The Three Templates</td>
<td>449</td>
</tr>
<tr>
<td>12</td>
<td>Pages and Navigation</td>
<td>539</td>
</tr>
</tbody>
</table>

### PART II SPECIALTIES

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Touch, Etc.</td>
<td>615</td>
</tr>
<tr>
<td>14</td>
<td>Bitmaps</td>
<td>683</td>
</tr>
<tr>
<td>15</td>
<td>Going Native</td>
<td>779</td>
</tr>
<tr>
<td>16</td>
<td>Rich Text</td>
<td>845</td>
</tr>
<tr>
<td>17</td>
<td>Share and Print</td>
<td>893</td>
</tr>
<tr>
<td>18</td>
<td>Sensors and GPS</td>
<td>953</td>
</tr>
<tr>
<td>19</td>
<td>Pen (Also Known as Stylus)</td>
<td>1013</td>
</tr>
</tbody>
</table>

*Index* 1057
This page intentionally left blank
Table of Contents

Introduction xvii

PART I ELEMENTALS

Chapter 1  Markup and Code  3
  The First Project .................................................. 3
  Graphical Greetings .............................................. 9
  Variations in Text ............................................... 13
  Media As Well ..................................................... 22
  The Code Alternatives ......................................... 23
  Images in Code ................................................... 27
  Not Even a Page .................................................. 29

Chapter 2  XAML Syntax  31
  The Gradient Brush in Code ..................................... 31
  Property Element Syntax ....................................... 34
  Content Properties ............................................. 37
  The TextBlock Content Property ............................ 41
  Sharing Brushes (and Other Resources) .................... 43
  Resources Are Shared .......................................... 47
  Exploring Vector Graphics .................................... 48

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can continually improve our books and learning
resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey
Chapter 5  Control Interaction  139
The Control Difference ...................................................... 139
The Slider for Ranges ......................................................... 141
The Grid ................................................................. 146
Orientation and Aspect Ratios ........................................... 152
Slider and the Formatted String Converter ....................... 154
Tooltips and Conversions ................................................. 154
Sketching with Sliders ...................................................... 157
The Varieties of Button Experience .................................... 159
Defining Dependency Properties ....................................... 167
RadioButton Tags ............................................................ 177
Keyboard Input and TextBox ............................................. 184
Touch and Thumb ........................................................... 187

Chapter 6  WinRT and MVVM  193
MVVM (Brief and Simplified) ............................................. 193
Data Binding Notifications ................................................ 194
A View Model for ColorScroll .......................................... 196
Syntactic Shortcuts .......................................................... 201
The DataContext Property .............................................. 204
Bindings and TextBox ..................................................... 206
Buttons and MVVM .......................................................... 212
The DelegateCommand Class .......................................... 213

Chapter 7  Asynchronicity  221
Threads and the User Interface ......................................... 221
Working with MessageDialog ......................................... 222
Callbacks as Lambda Functions ....................................... 228
The Amazing await Operator ........................................... 229
Cancelling an Asynchronous Operation ............................ 231
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approaches to File I/O</td>
<td>233</td>
</tr>
<tr>
<td>Application Local Storage</td>
<td>234</td>
</tr>
<tr>
<td>File Pickers</td>
<td>234</td>
</tr>
<tr>
<td>Bulk Access</td>
<td>235</td>
</tr>
<tr>
<td>File Pickers and File I/O</td>
<td>235</td>
</tr>
<tr>
<td>Handling Exceptions</td>
<td>240</td>
</tr>
<tr>
<td>Consolidating Async Calls</td>
<td>241</td>
</tr>
<tr>
<td>Streamlined File I/O</td>
<td>243</td>
</tr>
<tr>
<td>Application Lifecycle Issues</td>
<td>245</td>
</tr>
<tr>
<td>Your Own Asynchronous Methods</td>
<td>250</td>
</tr>
</tbody>
</table>

**Chapter 8  App Bars and Popups**

Implementing Context Menus                                             261
The *Popup* Dialog                                                     265
Application Bars                                                        268
The Application Bar Button Style                                       271
Inside the Segoe UI Symbol Font                                        276
App Bar *CheckBox* and *RadioButton*                                   283
An App Bar for a Note Pad                                              286
Introducing XamlCruncher                                               293
Application Settings and View Models                                   308
The XamlCruncher Page                                                  311
Parsing the XAML                                                       316
XAML Files In and Out                                                  318
The Settings Dialog                                                    322
Beyond the Windows Runtime                                             327

**Chapter 9  Animation**

The *Windows.UI.Xaml.Media.Animation* Namespace                      329
Animation Basics                                                        330
Animation Variation Appreciation                                       334
Other Double Animations .................................................. 340
Animating Attached Properties ......................................... 347
The Easing Functions ....................................................... 350
All-XAML Animations .................................................... 359
Animating Custom Classes .............................................. 364
Key Frame Animations .................................................... 367
The *Object* Animation .................................................. 371
Predefined Animations and Transitions ............................... 373

**Chapter 10 Transforms** ............................................... 377
A Brief Overview .......................................................... 377
Rotation (Manual and Animated) ......................................... 380
Visual Feedback ............................................................ 386
Translation ................................................................. 388
Transform Groups .......................................................... 391
The Scale Transform ....................................................... 396
Building an Analog Clock ................................................. 400
Skew ................................................................. 406
Making an Entrance ....................................................... 409
Transform Mathematics .................................................. 410
The *CompositeTransform* .............................................. 418
Geometry Transforms ..................................................... 421
Brush Transforms ......................................................... 422
Dude, Where’s My Element? .............................................. 427
Projection Transforms ................................................... 430
Deriving a *Matrix3D* .................................................... 437

**Chapter 11 The Three Templates** ................................. 449
Data in a Button ........................................................... 450
Making Decisions ........................................................... 460
Collection Controls and the *Real Use of DataTemplate* .................. 463
Collections and Interfaces .................................................. 474
Tapping and Selecting ......................................................... 475
Panels and Virtualizing Panels .............................................. 481
Custom Panels ................................................................. 484
The Item Template Bar Chart ................................................ 497
The *FlipView* Control .......................................................... 500
The Basic Control Template .................................................. 502
The Visual State Manager ..................................................... 513
Using generic.xaml .............................................................. 520
Template Parts ................................................................. 521
Custom Controls ............................................................... 530
Templates and Item Containers .............................................. 535

**Chapter 12 Pages and Navigation** .............................. 539
Screen Resolution Issues ..................................................... 539
Scaling Issues ................................................................. 545
Snap Views ................................................................. 549
Orientation Changes .......................................................... 554
Simple Page Navigation ...................................................... 557
The Back Stack ............................................................... 562
Navigation Events and Page Restoration ................................. 564
Saving and Restoring Application State .................................. 568
Navigational Accelerators and Mouse Buttons ......................... 572
Passing and Returning Data ................................................ 575
Visual Studio’s Standard Templates ....................................... 581
View Models and Collections .............................................. 588
Grouping the Items ........................................................... 608
## Chapter 13  Touch, Etc.  615

- A Pointer Roadmap ........................................ 616
- A First Dab at Finger Painting ................................. 619
- Capturing the Pointer ........................................ 622
- Editing with a Popup Menu ..................................... 630
- Pressure Sensitivity ......................................... 633
- Smoothing the Tapers ........................................ 637
- How Do I Save My Drawings? ................................. 646
- Real and Surreal Finger Painting .............................. 647
- A Touch Piano ................................................. 649
- Manipulation, Fingers, and Elements ...................... 655
- Working with Inertia ........................................ 663
- An XYSlider Control ....................................... 667
- Centered Scaling and Rotation ............................... 673
- Single-Finger Rotation ...................................... 676

## Chapter 14  Bitmaps  683

- Pixel Bits ......................................................... 684
- Transparency and Premultiplied Alphas .................... 691
- A Radial Gradient Brush .................................... 696
- Loading and Saving Image Files ............................. 703
- Posterize and Monochromize .................................. 714
- Saving Finger Paint Artwork ................................ 722
- HSL Color Selection ........................................ 747
- Reverse Painting ........................................... 758
- Accessing the Pictures Library .............................. 763
- Capturing Camera Photos ................................... 772
# Chapter 15  Going Native  779
- An Introduction to P/Invoke .......................... 780
- Some Help ............................................. 786
- Time Zone Information ................................. 786
- A Windows Runtime Component Wrapper for DirectX .......................... 808
- DirectWrite and Fonts .................................. 809
- Configurations and Platforms ......................... 821
- Interpreting Font Metrics .............................. 824
- Drawing on a SurfaceImageSource ................... 831

# Chapter 16  Rich Text  845
- Private Fonts ............................................ 847
- A Taste of Glyphs ....................................... 850
- Font Files in Local Storage ............................ 852
- Typographical Enhancements ......................... 856
- RichTextStyleBlock and Paragraphs .................. 858
- RichTextStyleBlock Selection ........................ 862
- RichTextStyleBlock and Overflow ..................... 862
- The Perils of Pagination ............................... 870
- Rich Editing with RichEditBox ....................... 877
- Your Own Text Input .................................... 886

# Chapter 17  Share and Print  893
- Settings and Popups .................................... 894
- Sharing Through the Clipboard ....................... 898
- The Share Charm ....................................... 903
- Basic Printing ......................................... 904
- Printable and Unprintable Margins ................... 911
- The Pagination Process ................................ 915
- Custom Printing Properties ......................... 922
| Printing a Monthly Planner                        | 928 |
| Printing a Range of Pages                        | 937 |
| Where To Do the Big Jobs?                        | 948 |
| Printing FingerPaint Art                         | 948 |

### Chapter 18 Sensors and GPS 953

- Orientation and Orientation                      953
- Acceleration, Force, Gravity, and Vectors       958
- Follow the Rolling Ball                          969
- The Two Norths                                   973
- Inclinometer = Accelerometer + Compass           976
- OrientationSensor = Accelerometer + Compass     980
- Azimuth and Altitude                             986
- Bing Maps and Bing Map Tiles                    999

### Chapter 19 Pen (Also Known as Stylus) 1013

- The InkManager Collections                      1014
- The Ink Drawing Attributes                       1017
- Erasing and Other Enhancements                  1023
- Selecting Strokes                                1029
- The Yellow Pad                                   1038

### Index 1057

---

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

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

microsoft.com/learning/booksurvey
Introduction

This book—the 6th edition of Programming Windows—is a guide to writing applications that run under Microsoft Windows 8.

To use this book, you'll need a computer running Windows 8, on which you can install the Windows 8 development tools and software development kit (SDK), most conveniently in the form of the free download of Microsoft Visual Studio Express 2012 for Windows 8. That download is accessible from the Windows 8 developer portal:

http://msdn.microsoft.com/windows/apps

To install Visual Studio, follow the “Downloads for developers” link on that page and then the “Download the tools and SDK” link on the following page. This page also provides information on obtaining a Windows 8 developer account that lets you upload new applications to the Windows Store.

The Versions of Windows 8

For the most part, Windows 8 is intended to run on the same class of personal computers as Windows 7, which are machines built around the 32-bit or 64-bit Intel x86 microprocessor family. Windows 8 is available in a regular edition called simply Windows 8 and also a Windows 8 Pro edition with additional features that appeal to tech enthusiasts and professionals.

Both Windows 8 and Windows 8 Pro run two types of programs:

- Desktop applications
- New Windows 8 applications, often called Windows Store applications

Desktop applications are traditional Windows programs that currently run under Windows 7 and that interact with the operating system through the Windows application programming interface, known familiarly as the Win32 API. To run these desktop applications, Windows 8 includes a familiar Windows desktop screen.
The new Windows Store applications represent a radical break with traditional Windows. The programs generally run in a full-screen mode—although two programs can share the screen in a “snap” mode—and many of these programs will probably be optimized for touch and tablet use. These applications are purchasable and installable only from the application store run by Microsoft. (As a developer, you can deploy and test applications directly from Visual Studio.)

In addition to the versions of Windows 8 that run on x86 processors, there is also a version of Windows 8 that runs on ARM processors, most commonly found in low-cost tablets and other mobile devices. This version of Windows 8 is called Windows RT, and it only comes preinstalled on these machines. One of the first computers running Windows RT is the initial release of the Microsoft Surface.

Aside from some preinstalled desktop applications, Windows RT runs new Windows Store applications only. You cannot run existing Windows 7 applications under Windows RT. You cannot run Visual Studio under Windows RT, and you cannot develop Windows 8 applications under Windows RT.

The Windows 8 user interface incorporates a new design paradigm that is likely to be reflected in Windows Store applications. Somewhat inspired by signage in urban environments, this design paradigm emphasizes content over program “chrome” and is characterized by the use of unadorned fonts, clean open styling, a tile-based interface, and transitional animations.

Many developers were first introduced to the Windows 8 design paradigm with Windows Phone 7, so it’s interesting to see how Microsoft’s thinking concerning large and small computers has evolved. In years gone by, Microsoft attempted to adapt the design of the traditional Windows desktop to smaller devices such as hand-held computers and phones. Now a user-interface design for the phone is being moved up to tablets and the desktop.

One important characteristic of this new environment is an emphasis on multitouch, which has dramatically changed the relationship between human and computer. In fact, the term “multitouch” is now outmoded because virtually all new touch devices respond to multiple fingers. The simple word “touch” is now sufficient. Part of the new programming interface for Windows 8 applications treats touch, mouse, and pen input in a unified manner so that applications are automatically usable with all three input devices.
The Focus of This Book

This book focuses exclusively on writing Windows Store applications. Plenty of other books already exist for writing Win32 desktop applications, including the 5th edition of *Programming Windows*. I’ll occasionally make reference to Win32 API and desktop applications, but this book is really all about writing new Windows 8 applications.

For writing these applications, a new object-oriented API has been introduced called the Windows Runtime or WinRT (not to be confused with the version of Windows 8 that runs on ARM processors, called Windows RT). Internally, the Windows Runtime is based on COM (Component Object Model) with interfaces exposed through metadata files with the extension .winmd located in the */Windows/System32/WinMetadata* directory. Externally, it is very object-oriented.

From the application programmer’s perspective, the Windows Runtime resembles Silverlight, although internally it is not a managed API. For Silverlight programmers, perhaps the most immediate difference involves namespace names: the Silverlight namespaces beginning with `System.Windows` have been replaced with namespaces beginning with `Windows.UI.Xaml`.

Most Windows 8 applications will be built not only from code but also markup, either the industry-standard HyperText Markup Language (HTML) or Microsoft’s eXtensible Application Markup Language (XAML). One advantage of splitting an application between code and markup is potentially splitting the development of the application between programmers and designers.

Currently there are three main options for writing Windows 8 applications, each of which involves a programming language and a markup language:

- C++ with XAML
- C# or Visual Basic with XAML
- JavaScript with HTML5

The Windows Runtime is common to all these options, but the Windows Runtime is also supplemented by another programming interface appropriate for the particular language. Although you can’t mix languages within a single application, you can create libraries (called Windows Runtime Components) with their own .winmd files that can be accessed from any other Windows 8 language.
The C++ programmer uses a dialect of C++ called C++ with Component Extensions, or C++/CX, that allows the language to make better use of WinRT. The C++ programmer also has direct access to a subset of the Win32 and COM APIs, as well as DirectX. C++ programs are compiled to native machine code.

Programmers who use the managed languages C# or Visual Basic .NET will find WinRT to be very familiar territory. Windows 8 applications written in these languages can’t access Win32, COM, or DirectX APIs with as much ease as the C++ programmer, but it is possible to do so, and some sample programs in Chapter 15, “Going Native,” show how. A stripped-down version of .NET is also available for performing low-level tasks.

For JavaScript, the Windows Runtime is supplemented by a Windows Library for JavaScript, or WinJS, which provides a number of system-level features for Windows 8 apps.

After much consideration (and some anguish), I decided that this book would focus almost exclusively on the C# and XAML option. For at least a decade I have been convinced of the advantages of managed languages for development and debugging, and for me C# is the language that has the closest fit to the Windows Runtime. I hope C++ programmers find C# code easy enough to read to derive some benefit from this book.

I also believe that a book focusing on one language option is more valuable than one that tries for equal coverage among several languages. There will undoubtedly be plenty of other Windows 8 books that show how to write Windows 8 applications using the other options.

With that said, I have greatly enjoyed the renewed debate about the advantages of C++ and native code in crafting high-performance applications. No single tool is best for every problem, and I will be exploring C++ and DirectX development for Windows 8 more in the future, both in my blog and the pages of MSDN Magazine. As a modest start, the companion content for this book includes all the program samples converted to C++.
The Approach

In writing this book, I've made a couple assumptions about you, the reader. I assume that you are comfortable with C#. If not, you might want to supplement this book with a C# tutorial. If you are coming to C# from a C or C++ background, my free online book .NET Book Zero: What the C or C++ Programmer Needs to Know About C# and the .NET Framework might be adequate. This book is available in PDF or XPS format at www.charlespetzold.com/dotnet.

I also assume that you know the rudimentary syntax of XML (eXtensible Markup Language) because XAML is based on XML. But I assume no familiarity with XAML or any XAML-based programming interface.

This is an API book rather than a tools book. The only programming tool I use in this book is Microsoft Visual Studio Express 2012 for Windows 8 (which I'll generally simply refer to as Visual Studio).

Markup languages are generally much more toolable than programming code. Indeed, some programmers even believe that markup such as XAML should be entirely machine-generated. Visual Studio has a built-in interactive XAML designer that involves dragging controls to a page, and many programmers have come to know and love Microsoft Expression Blend for generating complex XAML for their applications. (Expression Blend is included among the free download of the development tools and SDK I mentioned earlier.)

While such design tools are great for experienced programmers, I think that the programmer new to the environment is better served by learning how to write XAML by hand. That's how I'll approach XAML in this book. The XAML Cruncher tool featured in Chapter 8, “App Bars and Popups,” is very much in keeping with this philosophy: it lets you type in XAML and interactively see the objects that are generated, but it does not try to write XAML for you.

On the other hand, some programmers become so skilled at working with XAML that they forget how to create and initialize certain objects in code! I think both skills are important, and consequently I often show how to do similar tasks in both code and markup.
As I began working on this book, I contemplated different approaches to how a tutorial about the Windows Runtime can be structured. One approach is to start with rather low-level graphics and user input, demonstrate how controls can be built, and then describe the controls that have already been built for you.

I have instead chosen to focus initially on those skills I think are most important for most mainstream programmers: assembling the predefined controls in an application and linking them with code and data. This is the focus of the 12 chapters of the book’s Part I, “Elementals.” One of my goals in Part I is to make comprehensible all the code and markup that Visual Studio generates in the various project templates it supports.

Part II, “Specialities,” covers more low-level and esoteric tasks, such as touch, bitmap graphics, rich text, printing, and working with the orientation and GPS sensors.

**Source Code**

Learning a new API is similar to learning how to play basketball or the oboe: You don’t get the full benefit by watching someone else do it. Your own fingers must get involved. The source code in these pages is downloadable via the “Companion Content” link here:

http://shop.oreilly.com/product/0790145369079.do

But you’ll learn better by actually typing in the code yourself.

**My Setup**

For writing this book, I used the special version of the Samsung 700T tablet that was distributed to attendees of the Microsoft Build Conference in September 2011. (For that reason, it’s sometimes called the Build Tablet.) This machine has an Intel Core i5 processor running at 1.6 GHz with 4 GB of RAM and a 64-GB hard drive. The screen (from which most of the screenshots in the book were taken) has 8 touch points and a resolution of 1366 × 768 pixels, which is the lowest resolution for which snap views are supported.

Although the Build Tablets were originally distributed with the Windows 8 Developer Preview installed, I progressively replaced that with the Consumer Preview (build 8250) in March 2012 and the Release Preview (build 8400) in June 2012, and eventually the official release of Windows 8 Pro. Except when testing orientation sensors, I generally used the tablet in the docking port with an external 1920×1080 HDMI monitor, and an external keyboard and mouse.
When the Microsoft Surface first became available, I purchased one for testing my applications. For deploying and debugging applications on the Surface, I used the technique discussed by Tim Heuer in his blog entry:


This technique is more formally described in the documentation topic “Running Windows Store apps on a remote machine”:


The Surface became particularly vital for testing programs that access the orientation sensors.

For the most part, however, I’m still using the Build Tablet in the docking station. The external keyboard, mouse, and monitor lets me run Visual Studio and Microsoft Word as I’m accustomed to, while my Windows 8 programs run on the tablet’s touch screen. This is a fine development environment, particularly compared with the setup I used to write the first edition of Programming Windows.

But that was 25 years ago.

The Programming Windows Heritage

This is the 6th edition of Programming Windows, a book that was first conceived by Microsoft Press in the fall of 1986. The project came to involve me because at the time I was writing articles about Windows programming for Microsoft Systems Journal (the predecessor to MSDN Magazine).

I still get a thrill when I look at my very first book contract:
Perhaps the most amusing part of this contract occurs further down the first page:

The reference to “typescript” means that the pages must at least resemble something that came out of a typewriter. A double-spaced manuscript page with a fixed-pitch font has about 250 words, as the description indicates. A book page is more in the region of 400 words, so Microsoft Press obviously wasn’t expecting a very long book.

For writing the book I used an IBM PC/AT with an 80286 microprocessor running at 8 MHz with 512 KB of memory and two 30 MB hard drives. The display was an IBM Enhanced Graphics Adapter, with a maximum resolution of 640×350 with 16 simultaneous colors. I wrote some of the early chapters using Windows 1 (introduced over a year earlier in November 1985), but beta versions of Windows 2 soon became available.

In those years, editing and compiling a Windows program occurred outside of Windows in MS-DOS. For editing source code, I used WordStar 3.3, the same word processor I used for writing the chapters. From the MS-DOS command line, you would run the Microsoft C compiler and then launch Windows with your program to test it out. It was necessary to exit Windows and return to MS-DOS for the next edit-compile-run cycle.

As I got deeper into writing the book over the course of 1987, much of the rest of my life faded away. I stayed up later and later into the night. I didn’t have a television at the time, but the local public radio station, WNYC-FM, was on almost constantly with classical music and programming from National Public Radio. For a while, I managed to shift my day to such a degree that I went to bed after Morning Edition but awoke in time for All Things Considered.

As the contract stipulated, I sent chapters to Microsoft Press on diskette and paper. (We all had email, of course, but email didn’t support attachments at the time.) The edited chapters came back to me by mail decorated with proofreading marks and numerous sticky notes. I remember a page on which someone had drawn
a thermometer indicating the increasing number of pages I was turning in with the caption “Temperature’s Rising!”

Along the way, the focus of the book changed. Writing a book for “Programmers and Other Advanced Users” proved to be a flawed concept. I don’t know who came up with the title *Programming Windows*.

The contract had a completion date of April, but I didn’t finish until August and the book wasn’t published until early 1988. The final page total was about 850. If these were normal book pages (that is, without program listings or diagrams) the word count would be about 400,000 rather than the 100,000 indicated in the contract.

The cover of the first edition of *Programming Windows* described it as “The Microsoft Guide to Programming for the MS-DOS Presentation Manager: Windows 2.0 and Windows/386.” The reference to Presentation Manager reminds us of the days when Windows and the OS/2 Presentation Manager were supposed to peacefully coexist as similar environments for two different operating systems.

The first edition of *Programming Windows* went pretty much unnoticed by the programming community. When MS-DOS programmers gradually realized they needed to learn about the brave new environment of Windows, it was mostly the 2nd edition (published in 1990 and focusing on Windows 3) and the 3rd edition (1992, Windows 3.1) that helped out.


From the 1st edition to the 5th, I used the C programming language. Sometime between the 3rd and 4th editions, my good friend Jeff Prosise said that he wanted to write *Programming Windows with MFC*, and that was fine by me. I didn’t much care for the Microsoft Foundation Classes, which seemed to me a fairly light wrapper on the Windows API, and I wasn’t that thrilled with C++ either.

As the years went by, *Programming Windows* acquired the reputation of being the book for programmers who needed to get close to the metal without any extraneous obstacles between their program code and the operating system.

But to me, the early editions of *Programming Windows* were nothing of the sort. In those days, getting close to the metal involved coding in assembly language, writing character output directly into video display memory, and resorting to MS-DOS only for file I/O. In contrast, programming for Windows involved a high-level language,
completely unaccelerated graphics, and accessing hardware only through a heavy layer of APIs and device drivers.

This switch from MS-DOS to Windows represented a deliberate forfeit of speed and efficiency in return for other advantages. But what advantages? Many veteran programmers just couldn’t see the point. Graphics? Pictures? Color? Fancy fonts? A mouse? That’s not what computers are all about! The skeptics called it the WIMP (window-icon-menu-pointer) interface, which was not exactly a subtle implication about the people who chose to use such an environment or code for it.

If you wait long enough, a high-level language becomes a low-level language, and multiple layers of interface seemingly shrink down (at least in lingo) to a native API. Some C and C++ programmers of today reject a managed language like C# on grounds of efficiency, and Windows has even sparked some energetic controversy once again. Windows 8 is easily the most revolutionary updating to Windows since its very first release in 1985, but many old-time Windows users are wondering about the wisdom of bringing a touch-based interface tailored for smartphones and tablets to the mainstream desktop, and they grumble when they can’t find familiar features.

I suppose that Programming Windows could only be persuaded to emerge from semi-retirement with an exciting and controversial new user interface on Windows, and an API and programming language suited to its modern aspirations.

More in the Future

I suspect that Windows 8 will dominate my programming life for a while, which means that I’m likely to be posting blog entries about various aspects of Windows 8 programming. You can access my blog and subscribe to the RSS feed at www.charlespetzold.com.

I always enjoy solving a thorny programming problem and posting a blog entry about it, so if you have a Windows 8 programming issue that you’d like me to take a look at and possibly figure out, write me at cp@charlespetzold.com.

Beginning with the January 2013 issue of MSDN Magazine, I will be writing a monthly column called “DirectX Factor,” focusing specifically on using DirectX from Windows 8 and Windows Phone 8 applications. MSDN Magazine is available for free perusal at http://msdn.microsoft.com/magazine.
Behind the Scenes

This book exists only because Ben Ryan and Devon Musgrave at Microsoft Press developed an interesting way to release early content to the developer community and get advance sales of the final book simultaneously.

Part of the job duties of Devon and my technical reviewer Marc Young is to protect me from embarrassment by identifying blunders in my prose and code, and I thank them both for finding quite a few.

Thanks also to Andrew Whitechapel for giving me feedback on the C++ sample code; Brent Rector for an email with a crucial solution for an issue involving touch, as well as some background into IBuffer; Robert Levy for reflections about touch; Jeff Prosise for always seeming to have a dead-on answer when I’m puzzled; Larry Smith for finding numerous flaws in my prose; and Admiral for prodding me to make the book as useful as possible to C++ programmers.

The errors that remain in these chapters are my own fault, of course. Later in this Introduction is an email address for reporting errors to the publisher, but I’ll also try to identify the most egregious issues on my website at www.charlespetzold.com/pw6.

Finally, I want to thank my wife Deirdre Sinnott for love and support and making the necessary adjustments to our lives that writing a book inevitably entails.

Charles Petzold
Roscoe, NY and New York City
December 31, 2012

Errata & Book Support

We’ve made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at oreilly.com. Search for the book at http://microsoftpress.oreilly.com, and then click the “View/Submit Errata” link. If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software is not offered through the addresses above.
We Want to Hear from You

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

http://aka.ms/tellpress

The feedback form is very short, and we read every one of your comments and ideas. Thanks in advance for your input.

Stay in Touch

Let's keep the conversation going! We're on Twitter: http://twitter.com/MicrosoftPress
CHAPTER 3
Basic Event Handling

The previous chapters have demonstrated how you can instantiate and initialize elements and other objects in either XAML or code. The most common procedure is to use XAML to define the initial layout and appearance of elements on a page but then to change properties of these elements from code as the program is running.

As you've seen, assigning a Name or x:Name to an element in XAML causes a field to be defined in the page class that gives the code-behind file easy access to that element. This is one of the two major ways that code and XAML interact. The second is through events. An event is a general-purpose mechanism that allows one object to communicate something of interest to other objects. The event is said to be “fired” or “triggered” or “raised” by the first object and “handled” by the other. In the Windows Runtime, one important application of events is to signal the presence of user input from touch, the mouse, a pen, or the keyboard.

Following initialization, a Windows Runtime program generally sits dormant in memory waiting for something interesting to happen. Almost everything the program does thereafter is in response to an event, so the job of event handling is one that will occupy much of the rest of this book.

The Tapped Event

The UIElement class defines all the basic user-input events. These include

- eight events beginning with the word Pointer that consolidate input from touch, the mouse, and the pen;
- five events beginning with the word Manipulation that combine input from multiple fingers;
- two Key events for keyboard input; and
- higher level events named Tapped, DoubleTapped, RightTapped, and Holding.

No, the RightTapped event is not generated by a finger on your right hand; it’s mostly used to register right-button clicks on the mouse, but you can simulate a right tap with touch by holding your finger down for a moment and then lifting, a gesture that also generates Holding events. It’s the application’s responsibility to determine how it wants to handle these.
An extensive exploration of touch, mouse, and pen events awaits us in Chapter 13, “Touch, Etc.”

The only other events that UIElement defines are also related to user input:

- **GotFocus** and **LostFocus** signal when an element is the target of keyboard input; and
- **DragEnter**, **DragOver**, **DragLeave**, and **Drop** relate to drag-and-drop.

For now, let’s focus on **Tapped** as a simple representative event. An element that derives from UIElement fires a **Tapped** event to indicate that the user has briefly touched the element with a finger, or clicked it with the mouse, or dinged it with the pen. To qualify as a **Tapped** event, the finger (or mouse or pen) cannot move very much and must be released in a short period of time.

All the user-input events have a similar pattern. Expressed in C# syntax, UIElement defines the **Tapped** event like so:

```csharp
public event TappedEventHandler Tapped;
```

The **TappedEventHandler** is defined in the Windows.UI.Xaml.Input namespace. It’s a delegate type that defines the signature of the event handler:

```csharp
public delegate void TappedEventHandler(object sender, TappedRoutedEventArgs e);
```

In the event handler, the first argument indicates the source of the event (which is always an instance of a class that derives from UIElement) and the second argument provides properties and methods specific to the **Tapped** event.

The XAML file for the TapTextBlock program defines a **TextBlock** with a **Name** attribute as well as a handler for the **Tapped** event:

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"

    <TextBlock Name="txtblk"
        Text="Tap Text!"
        FontSize="96"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Tapped="txtblk_Tapped_1" />

</Grid>
```

As you type **TextBlock** attributes in XAML, IntelliSense suggests events as well as properties. These are distinguished with little icons: a wrench for properties and a lightning bolt for events. (You’ll also see a few with pairs of curly braces. These are attached properties that I’ll describe in Chapter 4, “Presentation with Panels.”) If you allow it, IntelliSense also suggests a name for the event handler, and I let it choose this one. Based solely on the XAML syntax, you really can’t tell which attributes are properties and which are events.
The actual event handler is implemented in the code-behind file. If you allow Visual Studio to select a handler name for you, you'll discover that Visual Studio also creates a skeleton event handler in the MainPage.xaml.cs file:

```csharp
private void txtblk_Tapped_1(object sender, TappedRoutedEventArgs e)
{
}
```

This is the method that is called when the user taps the TextBlock. In future projects, I'll change the names of event handlers to make them more to my liking. I'll remove the private keyword (because that's the default), I'll change the name to eliminate underscores and preface it with the word On (for example OnTextBlockTapped), and I'll change the argument named e to args. You can rename the method in the code file and then click a little global-rename icon to rename the method in the XAML file as well.

For this sample program, I decided I want to respond to the tap by setting the TextBlock to a random color. In preparation for that job, I defined fields for a Random object and a byte array for the red, green, and blue bytes:

```csharp
private void txtblk_Tapped_1(object sender, TappedRoutedEventArgs e)
{
    rand.NextBytes(rgb);
    Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
    txtblk.Foreground = new SolidColorBrush(clr);
}
```

I've removed the OnNavigatedTo method because it's not being used here. In the Tapped event handler, the NextBytes method of the Random object obtains three random bytes, and these are used to construct a Color value with the static Color.FromArgb method. The handler finishes by setting the Foreground property of the TextBlock to a SolidColorBrush based on that Color value.

When you run this program, you can tap the TextBlock with a finger, mouse, or pen and it will change to a random color. If you tap on an area of the screen outside the TextBlock, nothing happens. If you're using a mouse or pen, you might notice that you don't need to tap the actual strokes that comprise the letters. You can tap between and inside those strokes, and the TextBlock will still respond. It's as if the TextBlock has an invisible background that encompasses the full height of the font including diacritical marks and descenders, and that's precisely the case.
If you look inside the MainPage.g.cs file generated by Visual Studio, you'll see a Connect method containing the code that attaches the event handler to the Tapped event of the TextBlock. You can do this yourself in code. Try eliminating the Tapped handler assigned in the MainPage.xaml file and instead attach an event handler in the constructor of the code-behind file:

```csharp
public MainPage()
{
    this.InitializeComponent();
    txtblk.Tapped += txtblk_Tapped_1;
}
```

No real difference.

Several properties of TextBlock need to be set properly for the Tapped event to work. The IsHitTestVisible and IsTapEnabled properties must both be set to their default values of true. The Visibility property must be set to its default value of Visibility.Visible. If set to Visibility.Collapsed, the TextBlock will not be visible at all and will not respond to user input.

The first argument to the txtblk_Tapped_1 event handler is the element that sent the event, in this case the TextBlock. The second argument provides information about this particular event, including the coordinate point at which the tap occurred, and whether the tap came from a finger, mouse, or pen. This information will be explored in more detail in Chapter 13.

## Routed Event Handling

Because the first argument to the Tapped event handler is the element that generates the event, you don't need to give the TextBlock a name to access it from within the event handler. You can simply cast the sender argument to an object of type TextBlock. This technique is particularly useful for sharing an event handler among multiple elements, and I've done precisely that in the RoutedEvents0 project.

RoutedEvents0 is the first of several projects that demonstrate the concept of routed event handling, which is an important feature of the Windows Runtime. But this particular program doesn’t show any features particular to routed events. Hence the suffix of zero. For this project I created the Tapped handler first with the proper signature and my preferred name:

---

**Project: RoutedEvents0 | File: MainPage.xaml.cs (excerpt)**

```csharp
public sealed partial class MainPage : Page
{
    Random rand = new Random();
    byte[] rgb = new byte[3];

    public MainPage()
    {
        this.InitializeComponent();
    }

    void OnTextBlockTapped(object sender, TappedRoutedEventArgs args)
    {
        TextBlock txtblk = sender as TextBlock;
```
rand.NextBytes(rgb);
Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
txtblk.Foreground = new SolidColorBrush(clr);
}
}

Notice that the first line of the event handler casts the *sender* argument to *TextBlock*.

Because this event handler already exists in the code-behind file, Visual Studio suggests that name when you type the name of the event in the XAML file. This was handy because I added nine *TextBlock* elements to the *Grid*:

**Project: RoutedEvents0 | File: MainPage.xaml (excerpt)**

```xml
<Page x:Class="RoutedEvents0.MainPage"
... FontSize="48">
  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <TextBlock Text="Left / Top"
     HorizontalAlignment="Left"
     VerticalAlignment="Top"
      Tapped="OnTextBlockTapped" />
    ...
    <TextBlock Text="Right / Bottom"
      HorizontalAlignment="Right"
      VerticalAlignment="Bottom"
      Tapped="OnTextBlockTapped" />
  </Grid>
</Page>
```

I'm sure you don't need to see them all to get the general idea. Notice that *FontSize* is set for the *Page* so that it is inherited by all the *TextBlock* elements. When you run the program, you can tap the individual elements and each one changes its color independently of the others:

<table>
<thead>
<tr>
<th>Left / Top</th>
<th>Center / Top</th>
<th>Right / Top</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Left / Center</th>
<th>Center / Center</th>
<th>Right / Center</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Left / Bottom</th>
<th>Center / Bottom</th>
<th>Right / Bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If you tap anywhere between the elements, nothing happens.

You might consider it a nuisance to set the same event handler on nine different elements in the XAML file. If so, you’ll probably appreciate the following variation to the program. The RoutedEvents1 program uses routed input handling, a term used to describe how input events such as Tapped are fired by the element on which the event occurs but the events are then routed up the visual tree. Rather than set a Tapped handler for the individual TextBlock elements, you can instead set it on the parent of one of these elements (for example, the Grid). Here’s an excerpt from the XAML file for the RoutedEvents1 program:

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}
    Tapped="OnGridTapped">
    <TextBlock Text="Left / Top"
        HorizontalAlignment="Left"
        VerticalAlignment="Top" />

    ...

    <TextBlock Text="Right / Bottom"
        HorizontalAlignment="Right"
        VerticalAlignment="Bottom" />
</Grid>
```

In the process of moving the Tapped handler from the individual TextBlock elements to the Grid, I’ve also renamed it to more accurately describe the source of the event.

The event handler must also be modified. The previous Tapped handler cast the sender argument to a TextBlock. It could perform this cast with confidence because the event handler was set only on elements of type TextBlock. However, when the event handler is set on the Grid as it is here, the sender argument to the event handler will be the Grid. How can we determine which TextBlock was tapped?

Easy: The TappedRoutedEventArgs class—an instance of which appears as the second argument to the event handler—has a property named OriginalSource, and that indicates the source of the event. In this example, OriginalSource can be either a TextBlock (if you tap the text) or the Grid (if you tap between the text), so the new event handler must perform a check before casting:

```csharp
void OnGridTapped(object sender, TappedRoutedEventArgs args)
{
    if (args.OriginalSource is TextBlock)
    {
        TextBlock txtblk = args.OriginalSource as TextBlock;
        rand.NextBytes(rgb);
        Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
        txtblk.Foreground = new SolidColorBrush(clr);
    }
}
```
Slightly more efficient is performing the cast first and then checking if the result is non-null.

`TappedRoutedEventArgs` derives from `RoutedEventArgs`, which defines `OriginalSource` and no other properties. Obviously, the `OriginalSource` property is a central concept of routed event handling. The property allows elements to process events that originate with their children and other descendents in the visual tree and to know the source of these events. Routed event handling lets a parent know what its children are up to, and `OriginalSource` identifies the particular child involved.

Alternatively, you can set the `Tapped` handler on `MainPage` rather than the `Grid`. But with `MainPage` there's an easier way. I mentioned earlier that `UIElement` defines all the user-input events. These events are inherited by all derived classes, but the `Control` class adds its own event interface consisting of a whole collection of virtual methods corresponding to these events. For example, for the `Tapped` event defined by `UIElement`, the `Control` class defines a virtual method named `OnTapped`. These virtual methods always begin with the word `On` followed by the name of the event, so they are sometimes referred to as "On methods." `Page` derives from `Control` through `UserControl`, so these methods are inherited by the `Page` and `MainPage` classes.

Here's an excerpt from the XAML file for RoutedEvents2 demonstrating that the XAML file defines no event handlers:

```
<Project: RoutedEvents2 | File: MainPage.xaml (excerpt)
<Page
    x:Class="RoutedEvents2.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:RoutedEvents2"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    FontSize="48">
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <TextBlock Text="Left / Top"
            HorizontalAlignment="Left"
            VerticalAlignment="Top" />
        
        ...
        
        <TextBlock Text="Right / Bottom"
            HorizontalAlignment="Right"
            VerticalAlignment="Bottom" />
    </Grid>
</Page>
```
Instead, the code-behind file has an override of the `OnTapped` method:

```
protected override void OnTapped(TappedRoutedEventArgs args)
{
    if (args.OriginalSource is TextBlock)
    {
        TextBlock txtblk = args.OriginalSource as TextBlock;
        rand.NextBytes(rgb);
        Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
        txtblk.Foreground = new SolidColorBrush(clr);
    }
    base.OnTapped(args);
}
```

When you’re typing in Visual Studio and you want to override a virtual method like `OnTapped`, simply type the keyword `override` and press the space bar, and Visual Studio will provide a list of all the virtual methods defined for that class. When you select one, Visual Studio creates a skeleton method with a call to the base method. A call to the base method isn’t really required here, but including it is a good habit to develop when overriding virtual methods. Depending on the method you’re overriding, you might want to call the base method first, last, in the middle, or not at all.

The `On` methods are basically the same as the event handlers, but they have no `sender` argument because it would be redundant: `sender` would be the same as `this`, the instance of the `Page` that is processing the event.

The next project is RoutedEvents3. I decided to give the `Grid` a random background color if that’s the element being tapped. The XAML file looks the same, but the revised `OnTapped` method looks like this:

```
protected override void OnTapped(TappedRoutedEventArgs args)
{
    rand.NextBytes(rgb);
    Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
    SolidColorBrush brush = new SolidColorBrush(clr);
    if (args.OriginalSource is TextBlock)
    {
        args.OriginalSource as TextBlock).Foreground = brush;
    }
    else if (args.OriginalSource is Grid)
    {
        args.OriginalSource as Grid).Background = brush;
    }
    base.OnTapped(args);
}
```

Now when you tap a `TextBlock` element, it changes color, but when you tap anywhere else on the screen, the `Grid` changes color.

Now suppose for one reason or another, you decide you want to go back to the original scheme of explicitly defining an event handler separately for each `TextBlock` element to change the text colors, but you also want to retain the `OnTapped` override for changing the `Grid` background color. In the
RoutedEvents4 project, the XAML file has the *Tapped* events restored for *TextBlock* elements and the *Grid* has been given a name:

**Project: RoutedEvents4 | File: MainPage.xaml (excerpt)**

```xml
<Grid Name="contentGrid"
      Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
>
  <TextBlock Text="Left / Top"
             HorizontalAlignment="Left"
             VerticalAlignment="Top"
             Tapped="OnTextBlockTapped" />

  ...

  <TextBlock Text="Right / Bottom"
             HorizontalAlignment="Right"
             VerticalAlignment="Bottom"
             Tapped="OnTextBlockTapped" />
</Grid>
```

One advantage is that the methods to set the *TextBlock* and *Grid* colors are now separate and distinct, so there's no need for *if-else* blocks. The *Tapped* handler for the *TextBlock* elements can cast the *sender* argument with impunity, and the *OnTapped* override can simply access the *Grid* by name:

**Project: RoutedEvents4 | File: MainPage.xaml.cs (excerpt)**

```csharp
public sealed partial class MainPage : Page
{
    Random rand = new Random();
    byte[] rgb = new byte[3];

    public MainPage()
    {
        this.InitializeComponent();
    }

    void OnTextBlockTapped(object sender, TappedRoutedEventArgs args)
    {
        TextBlock txtblk = sender as TextBlock;
        txtblk.Foreground = GetRandomBrush();
    }

    protected override void OnTapped(TappedRoutedEventArgs args)
    {
        contentGrid.Background = GetRandomBrush();
        base.OnTapped(args);
    }

    Brush GetRandomBrush()
    {
        rand.NextBytes(rgb);
        Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
        return new SolidColorBrush(clr);
    }
}
```
However, the code might not do exactly what you want. When you tap a TextBlock, not only does the TextBlock change color, but the event continues to go up the visual tree where it’s processed by the OnTapped override, and the Grid changes color as well! If that’s what you want, you’re in luck. If not, then I’m sure you’ll be interested to know that the TappedRoutedEventArgs has a property specifically to prevent this. If the OnTextBlockTapped handler sets the Handled property of the event arguments to true, the event is effectively inhibited from further processing higher in the visual tree.

This is demonstrated in the RoutedEvents5 project, which is the same as RoutedEvents4 except for a single statement in the OnTextBlockTapped method:

```csharp
void OnTextBlockTapped(object sender, TappedRoutedEventArgs args)
{
    TextBlock txtblk = sender as TextBlock;
    txtblk.Foreground = GetRandomBrush();
    args.Handled = true;
}
```

### Overriding the Handled Setting

You’ve just seen that when an element handles an event such as Tapped and concludes its event processing by setting the Handled property of the event arguments to true, the routing of the event effectively stops. The event isn’t visible to elements higher in the visual tree.

In some cases, this behavior might be undesirable. Suppose you’re working with an element that sets the Handled property to true in its event handler, but you still want to see that event higher in the visual tree. One solution is to simply change the code, but that option might not be available. The element might be implemented in a dynamic-link library, and you might not have access to the source code.

In RoutedEvents6, the XAML file is the same as in RoutedEvents5: Each TextBlock has a handler set for its Tapped event. The Tapped handler sets the Handled property to true. The class also defines a separate OnPageTapped handler that sets the background color of the Grid:

```csharp
public MainPage()
{
    this.InitializeComponent();

    this.AddHandler(UIElement.TappedEvent,
                    new TappedEventHandler(OnPageTapped),
                    true);
}
```
void OnTextBlockTapped(object sender, TappedRoutedEventArgs args)
{
    TextBlock txtblk = sender as TextBlock;
    txtblk.Foreground = GetRandomBrush();
    args.Handled = true;
}

void OnPageTapped(object sender, TappedRoutedEventArgs args)
{
    contentGrid.Background = GetRandomBrush();
}

Brush GetRandomBrush()
{
    rand.NextBytes(rgb);
    Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
    return new SolidColorBrush(clr);
}

But look at the interesting way that the constructor sets a Tapped handler for the Page. Normally, it would attach the event handler like so:

this.Tapped += OnPageTapped;

In that case the OnPageTapped handler would not get a Tapped event originating with the TextBlock because the TextBlock handler sets Handled to true. Instead, it attaches the handler with a method named AddHandler:

this.AddHandler(UIElement.TappedEvent,
    new TappedEventHandler(OnPageTapped),
    true);

AddHandler is defined by UIElement, which also defines the static UIElement.TappedEvent property. This property is of type RoutedEvent.

Just as a property like FontSize is backed by a static property named FontSizeProperty of type DependencyProperty, a routed event such as Tapped is backed by a static property named TappedEvent of type RoutedEvent. RoutedEvent defines nothing public on its own; it mainly exists to allow an event to be referenced in code without requiring an instance of an element.

The AddHandler method attaches a handler to that event. The second argument of AddHandler is defined as just an object, so creating a delegate object is required to reference the event handler. And here’s the magic: Set the last argument to true if you want this handler to also receive routed events that have been flagged as Handled.

The AddHandler method isn’t used often, but when you need it, it is essential.
Input, Alignment, and Backgrounds

I have just one more, very short program in the RoutedEvents series to make a couple important points about input events.

The XAML file for RoutedEvents7 has just one TextBlock and no event handlers defined:

```xml
<Project: RoutedEvents7 | File: MainPage.xaml (excerpt)
<Page ...
    FontSize="48">
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
        <TextBlock Text="Hello, Windows 8!"
            Foreground="Red" />
    </Grid>
</Page>
```

The absence of HorizontalAlignment and VerticalAlignment settings on the TextBlock cause it to appear in the upper-left corner of the Grid.

Like RoutedEvents3, the code-behind file contains separate processing for an event originating from the TextBlock and an event coming from the Grid:

```csharp
<Project: RoutedEvents7 | File: MainPage.xaml.cs (excerpt)
public sealed partial class MainPage : Page
{
    Random rand = new Random();
    byte[] rgb = new byte[3];

    public MainPage()
    {
        this.InitializeComponent();
    }

    protected override void OnTapped(TappedRoutedEventArgs args)
    {
        rand.NextBytes(rgb);
        Color clr = Color.FromArgb(255, rgb[0], rgb[1], rgb[2]);
        SolidColorBrush brush = new SolidColorBrush(clr);
        if (args.OriginalSource is TextBlock)
            (args.OriginalSource as TextBlock).Foreground = brush;
        else if (args.OriginalSource is Grid)
            (args.OriginalSource as Grid).Background = brush;

        base.OnTapped(args);
    }
}
```
As you tap the `TextBlock`, it changes to a random color like normal, but when you tap outside the `TextBlock`, the `Grid` doesn't change color like it did earlier. Instead, the `TextBlock` changes color! It's as if...yes, it's as if the `TextBlock` is now occupying the entire page and snagging all the `Tapped` events for itself.

And that's precisely the case. This `TextBlock` has default values of `HorizontalAlignment` and `VerticalAlignment`, but those default values are not `Left` and `Top` like the visuals might suggest. The default values are named `Stretch`, and that means that the `TextBlock` is stretched to the size of its parent, the `Grid`. It's hard to tell because the text still has a 48-pixel font, but the `TextBlock` has a transparent background that now fills the entire page.

In fact, throughout the Windows Runtime, all elements have default `HorizontalAlignment` and `VerticalAlignment` values of `Stretch`, and it's an important part of the Windows Runtime layout system. More details are coming in Chapter 4.

Let's put `HorizontalAlignment` and `VerticalAlignment` values in this `TextBlock`:

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock Text="Hello, Windows 8!"
             HorizontalAlignment="Left"
             VerticalAlignment="Top"
             Foreground="Red" />
</Grid>
```

Now the `TextBlock` is only occupying a small area in the upper-left corner of the page, and when you tap outside the `TextBlock`, the `Grid` changes color.
Now change HorizontalAlignment to TextAlignment:

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock Text="Hello, Windows 8!"
             TextAlignment="Left"
             VerticalAlignment="Top"
             Foreground="Red" />
</Grid>
```

The program looks the same. The text is still positioned at the upper-left corner. But now when you tap to the right of the TextBlock, the TextBlock changes color rather than the Grid. The TextBlock has its default HorizontalAlignment property of Stretch, so it is now occupying the entire width of the screen, but within the total width that the TextBlock occupies, the text is aligned to the left.

The lesson: HorizontalAlignment and TextAlignment are not equivalent, although they might seem to be if you judge solely from the visuals.

Now try another experiment by restoring the HorizontalAlignment setting and removing the Background property of the Grid:

```xml
<Grid>
  <TextBlock Text="Hello, Windows 8!"
             HorizontalAlignment="Left"
             VerticalAlignment="Top"
             Foreground="Red" />
</Grid>
```

With a light theme, the Grid has an off-white background. When the Background property is removed, the background of the page changes to black. But you'll also experience a change in the behavior of the program: The TextBlock still changes color when you tap it, but when you tap outside the TextBlock, the Grid doesn't change color at all.

The default value of the Background property defined by Panel (and inherited by Grid) is null, and with a null background, the Grid doesn't trap touch events. They just fall right through.

One way to fix this without altering the visual appearance is to give the Grid a Background property of Transparent:

```xml
<Grid Background="Transparent">
  <TextBlock Text="Hello, Windows 8!"
             HorizontalAlignment="Left"
             VerticalAlignment="Top"
             Foreground="Red" />
</Grid>
```

It looks the same as null, but now you'll get Tapped events with an OriginalSource of Grid.

The lessons here are important: Looks can be deceiving. An element with default settings of HorizontalAlignment and VerticalAlignment might look the same as one with settings of Left and Top, but it is actually occupying the entire area of its container and might block events from reaching underlying elements. A Panel derivative with a default Background property of null might look the same as one with a setting of Transparent, but it does not respond to touch events.
I can almost guarantee that sometime in the future, one of these two issues will cause a bug in one of your programs that will drive you crazy for the good part of a day, and that this will happen even after many years of working with the XAML layout system.

I speak from experience.

**Size and Orientation Changes**

Many, many years ago when Windows was very young, information about Windows programming was hard to find. It wasn’t until the December 1986 issue of *Microsoft Systems Journal* (the predecessor to *MSDN Magazine*) that the very first magazine article about Windows programming appeared. The article described a program called WHATSIZE (all capital letters, of course), which did little more than display the current size of the program’s window. But as the size of the window changed, the displayed values reflected that change.

Obviously, the original WHATSIZE program was written for the Windows APIs of that era, so it redrew the display in response to a WM_PAINT message. In the original Windows API, this message occurred whenever the contents of part of a program’s window became “invalid” and needed redrawing. A program could define its window so that the entire window was invalidated whenever its size changed.

The Windows Runtime has no equivalent of the WM_PAINT message, and indeed, the entire graphics paradigm is quite different. Previous versions of Windows implemented a “direct mode” graphics system in which applications drew to the actual video memory. Of course, this occurred through a software layer (the Graphics Device Interface) and a device driver, but at some point in the actual drawing functions, code was writing into video display memory.

The Windows Runtime is quite different. In its public programming interface, it doesn’t even have a concept of drawing or painting. Instead, a Windows 8 application creates elements—that is, objects instantiated from classes that derive from *FrameworkElement*—and adds them to the application’s visual tree. These elements are responsible for rendering themselves. When a Windows 8 application wants to display text, it doesn’t draw text but instead creates a *TextBlock*. When the application wants to display a bitmap, it creates an *Image* element. Instead of drawing lines and Bézier splines and ellipses, the program creates *Polyline* and *Path* elements.

The Windows Runtime implements a “retained mode” graphics system. Between your application and the video display is a composition layer on which all the rendered output is assembled before it is presented to the user. Perhaps the most important benefit of retained mode graphics is flicker-free animation, as you’ll witness for yourself toward the end of this chapter and in much of the remainder of this book.

Although the graphics system in the Windows Runtime is very different from earlier versions of Windows, in another sense a Windows 8 application is similar to its earlier brethren. Once a program is loaded into memory and starts running, it spends most of its time generally sitting dormant in memory, waiting for something interesting to happen. These notifications take the form of events and
callbacks. Often these events signal user input, but there might be other interesting activity as well. One such callback is the OnNavigatedTo method. In a simple single-page program, this method is called soon after the constructor returns.

Another event that might be of interest to a Windows 8 application—particularly one that does what the old WHATSIZE program did—is named SizeChanged. Here's the XAML file for the Windows 8 WhatSize program. Notice that the root element defines a handler for the SizeChanged event:

Project: WhatSize | File: MainPage.xaml (excerpt)

```xml
<Page x:Class="WhatSize.MainPage"
     ...>
  FontSize="36"
  SizeChanged="OnPageSizeChanged">
  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
        >
    <TextBlock HorizontalAlignment="Center"
               VerticalAlignment="Top">
      &x21A4; <Run x:Name="widthText" /> pixels &x21A6;
    </TextBlock>
    <TextBlock HorizontalAlignment="Center"
               VerticalAlignment="Center"
               TextAlignment="Center">
      &x21A5;
      <LineBreak />
      <Run x:Name="heightText" /> pixels
      <LineBreak />
      &x21A7;
    </TextBlock>
  </Grid>
</Page>

The remainder of the XAML file defines two TextBlock elements containing some Run objects surrounded by arrow characters. (You'll see what they look like soon.) It might seem excessive to set three properties to Center in the second TextBlock, but they're all necessary. The first two center the TextBlock in the page; setting TextAlignment to Center results in the two arrows being centered relative to the text. The two Run elements are given x:Name attributes so that the Text properties can be set in code. This happens in the SizeChanged event handler:

Project: WhatSize | File: MainPage.xaml.cs (excerpt)

```csharp
public sealed partial class MainPage : Page
{
  public MainPage()
  {
    this.InitializeComponent();
  }

  void OnPageSizeChanged(object sender, SizeChangedEventArgs args)
  {
    widthText.Text = args.NewSize.Width.ToString();
    heightText.Text = args.NewSize.Height.ToString();
  }
}
```

84 PART 1 Elementals
Very conveniently, the event arguments supply the new size in the form of a `Size` structure, and the handler simply converts the `Width` and `Height` properties to strings and sets them to the `Text` properties of the two `Run` elements:

If you’re running the program on a device that responds to orientation changes, you can try flipping the screen and observe how the numbers change. You can also sweep your finger from the left of the screen to invoke the snapped views and then divide the screen between this program and another to see how the width value changes.

You don’t need to set the `SizeChanged` event handler in XAML. You can set it in code, perhaps during the `Page` constructor:

```csharp
this.SizeChanged += OnPageSizeChanged;
```

`SizeChanged` is defined by `FrameworkElement` and inherited by all descendent classes. Despite the fact that `SizeChangedEventArgs` derives from `RoutedEventArgs`, this is not a routed event. You can tell it’s not a routed event because the `OriginalSource` property of the event arguments is always `null`; there is no `SizeChangedEvent` property; and whatever element you set this event on, that’s the element’s size you get. But you can set `SizeChanged` handlers on any element. Generally, the order the events are fired proceeds down the visual tree: `MainPage` first (in this example), and then `Grid` and `TextBlock`.

If you need the rendered size of an element other than in the context of a `SizeChanged` handler, that information is available from the `ActualWidth` and `ActualHeight` properties defined by `FrameworkElement`. Indeed, the `SizeChanged` handler in `WhatSize` is actually a little shorter when accessing those properties:

```csharp
void OnPageSizeChanged(object sender, SizeChangedEventArgs args)
{
    widthText.Text = this.ActualWidth.ToString();
    heightText.Text = this.ActualHeight.ToString();
}
```
What you probably do not want are the Width and Height properties. Those properties are also defined by FrameworkElement, but they have default values of “not a number” or NaN. A program can set Width and Height to explicit values (such as in the TextFormatting project in Chapter 2, “XAML Syntax”), but usually these properties remain at their default values and they are of no use in determining how large an element actually is. FrameworkElement also defines MinWidth, MaxWidth, MinHeight, and MaxHeight properties with default NaN values, but these aren’t used very often.

If you access the ActualWidth and ActualHeight properties in the page’s constructor, however, you’ll find they have values of zero. Despite the fact that InitializeComponent has constructed the visual tree, that visual tree has not yet gone through a layout process. After the constructor finishes, the page gets several events in sequence:

- OnNavigatedTo
- SizeChanged
- LayoutUpdated
- Loaded

If the page later changes size, additional SizeChanged events and LayoutUpdated events are fired. LayoutUpdated can also be fired if elements are added to or removed from the visual tree or if an element is changed so as to affect layout.

If you need a place to perform initialization after initial layout when all the elements in the visual tree have nonzero sizes, the event you want is Loaded. It is very common for a Page derivative to attach a handler for the Loaded event. Generally, the Loaded event occurs only once during the lifetime of a Page object. I say “generally” because if the Page object is detached from its parent (a Frame) and reattached, the Loaded event will occur again. But this won’t happen unless you deliberately make it happen. Also, the Unloaded event can let you know if the page has been detached from the visual tree.

Every FrameworkElement derivative has a Loaded event. As a visual tree is built, the Loaded events occur in a sequence going up the visual tree, ending with the Page derivative. When that Page object gets a Loaded event, it can assume that all its children have fired their own Loaded events and everything has been correctly sized.

Handling a Loaded event in a Page class is so common that some programmers perform Loaded processing right in the constructor using an anonymous handler:

```csharp
public MainPage()
{
    this.InitializeComponent();

    Loaded += (sender, args) =>
    {
        ...
    };
}  
```
Sometimes Windows 8 applications need to know when the orientation of the screen changes. In Chapter 1, "Markup and Code," I showed an InternationalHelloWorld program that looks fine in landscape mode but probably results in overlapping text if switched to portrait mode. To fix that, the ScalableInternationalHelloWorld program code-behind file changes the page’s FontSize property to 24 in portrait mode:

Project: ScalableInternationalHelloWorld | File: MainPage.xaml.cs

```csharp
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
       SetFont();
        DisplayProperties.OrientationChanged += OnDisplayPropertiesOrientationChanged;
    }

    void OnDisplayPropertiesOrientationChanged(object sender)
    {
        SetFont();
    }

    void SetFont()
    {
        bool isLandscape =
            DisplayProperties.CurrentOrientation == DisplayOrientations.Landscape ||
            DisplayProperties.CurrentOrientation == DisplayOrientations.LandscapeFlipped;

        this.FontSize = isLandscape ? 40 : 24;
    }
}
```

The DisplayProperties class and DisplayOrientations enumeration are defined in the Windows.Graphics.Display namespace. DisplayProperties.OrientationChanged is a static event, and when that event is fired, the static DisplayProperties.CurrentOrientation property provides the current orientation.

Somewhat more information, including snapped states, is provided by the ViewStateChanged event of the ApplicationView class in the Windows.UI.ViewManagement namespace, but working with this event must await Chapter 12, "Pages and Navigation."

**Bindings to Run?**

In Chapter 2 I discussed data bindings. Data bindings can link properties of two elements so that when a source property changes, the target property also changes. Data bindings are particularly satisfying when they eliminate the need for event handlers.

Is it possible to rewrite WhatSize to use data bindings rather than a SizeChanged handler? It’s worth a try.
In the WhatSize project, remove the OnPageSizeChanged handler from the MainPage.xaml.cs file (or just comment it out if you don’t want to do too much damage to the file). In the root tag of the MainPage.xaml file, remove the SizeChanged attribute and give MainPage a name of “page.” Then, set Binding markup extensions on the two Run objects referencing the ActualWidth and ActualHeight properties of the page:

```xml
<Page ...
    FontSize="36"
    Name="page">
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
        <TextBlock HorizontalAlignment="Center"
            VerticalAlignment="Top">
            &lt;Run Text="{Binding ElementName=page, Path=ActualWidth}" />
            pixels &lt;21A6;
        </TextBlock>
        <TextBlock HorizontalAlignment="Center"
            VerticalAlignment="Center"
            TextAlignment="Center">
            &lt;LineBreak />
            <Run Text="{Binding ElementName=page, Path=ActualHeight}" /> pixels
            <LineBreak />
            &lt;21A7;
        </TextBlock>
    </Grid>
</Page>
```

The program compiles fine, and it runs smoothly without any run-time exceptions. The only problem is: Where the numbers should appear is a discouraging 0.

This is likely to seem odd, particularly when you set the same bindings on the Text property of TextBlock instead of Run:

```xml
<Page ...
    FontSize="36"
    Name="page">
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
        <TextBlock HorizontalAlignment="Center"
            VerticalAlignment="Top"
            Text="{Binding ElementName=page, Path=ActualWidth}" />

        <TextBlock HorizontalAlignment="Center"
            VerticalAlignment="Center"
            TextAlignment="Center"
            Text="{Binding ElementName=page, Path=ActualHeight}" />
    </Grid>
</Page>
```
This works:

At least it appears to work at first. With the version of Windows 8 that I’m using to write this chapter, the numbers are not updated as you change the orientation or size of the page, and they really should be. In theory, a data binding is notified when a source property changes so that it can change the target property, but the application source code appears to have no event handlers and no moving parts. This is what is supposed to make data bindings so great.

Unfortunately, by giving up on the bindings to Run we’ve also lost the informative arrows. So, why do the data bindings work (or almost work) on the Text property of TextBlock but not at all on the Text property of Run?

It’s very simple. The target of a data binding must be a dependency property. This fact is obvious when you define a data binding in code by using the SetBinding method. That’s the difference: The Text property of TextBlock is backed by the TextProperty dependency property, but the Text property of Run is not. The Run version of Text is a plain old property that cannot serve as a target for a data binding. The XAML parser probably shouldn’t allow a binding to be set on the Text property of Run, but it does.

In Chapter 4 I’ll show you how to use a StackPanel to get the arrows back in a version of WhatSize that uses data bindings, and in Chapter 16, “Rich Text,” I’ll demonstrate a technique using RichTextBlock.

Timers and Animation

Sometimes a Windows 8 application needs to receive periodic events at a fixed interval. A clock application, for example, probably needs to update its display every second. The ideal class for this job is DispatcherTimer. Set a timer interval, set a handler for the Tick event, and go.
Here's the XAML file for a digital clock application. It's just a big `TextBlock`:

Project: DigitalClock | File: MainPage.xaml (excerpt)

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
    <TextBlock Name="txtblk"
        FontFamily="Lucida Console"
        FontSize="120"
        HorizontalAlignment="Center"
        VerticalAlignment="Center" />
</Grid>
```

The code-behind file creates the `DispatcherTimer` with a 1-second interval and sets the `Text` property of the `TextBlock` in the event handler:

Project: DigitalClock | File: MainPage.xaml.cs (excerpt)

```csharp
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();

        DispatcherTimer timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromSeconds(1);
        timer.Tick += OnTimerTick;
        timer.Start();
    }

    void OnTimerTick(object sender, object e)
    {
        txtblk.Text = DateTime.Now.ToString("h:mm:ss tt");
    }
}
```

And here it is:

7:43:14 PM
Calls to the *Tick* handler occur in the same execution thread as the rest of the user interface, so if the program is busy doing something in that thread, the calls won’t interrupt that work and might become somewhat irregular and even skip a few beats. In a multipage application, you might want to start the timer in the *OnNavigatedTo* override and stop it in *OnNavigatedFrom* to avoid the program wasting time doing work when the page is not visible.

This is a good illustration of the difference in how a desktop Windows application and a Windows 8 application update the video display. Both types of applications use a timer for implementing a clock, but rather than drawing and redrawing text every second by invalidating the contents of the window, the Windows 8 application changes the visual appearance of an existing element simply by changing one of its properties.

You can set the *DispatcherTimer* for an interval as low as you want, but you’re not going to get calls to the *Tick* handler faster than the frame rate of the video display, which is probably 60 Hertz or about a 17-millisecond period. Of course, it doesn’t make sense to update the video display faster than the frame rate. Updating the display precisely at the frame rate gives you as smooth an animation as possible. If you want to perform an animation in this way, don’t use *DispatcherTimer*. A better choice is the static *CompositionTarget.Rendering* event, which is specifically designed to be called prior to a screen refresh.

Even better than *CompositionTarget.Rendering* are all the animation classes provided as part of the Windows Runtime. These classes let you define animations in XAML or code, they have lots of options, and some of them are performed in background threads.

But until I cover the animation classes in Chapter 9, “Animation”—and perhaps even after I do—the *CompositionTarget.Rendering* event is well suited for performing animations. These are sometimes called “manual” animations because the program itself has to carry out some calculations based on elapsed time.

Here’s a little project called ExpandingText that changes the *FontSize* of a *TextBlock* in the *CompositionTarget.Rendering* event handler, making the text larger and smaller. The XAML file simply instantiates a *TextBlock*:

**Project:** ExpandingText | **File:** MainPage.xaml (excerpt)

```xml
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
   <TextBlock Name="txtblk"
     Text="Hello, Windows 8!"
     HorizontalAlignment="Center"
     VerticalAlignment="Center" />
</Grid>
```

In the code-behind file, the constructor starts a *CompositionTarget.Rendering* event simply by setting an event handler. The second argument to that handler is defined as type *object*, but it is
actually of type `RenderingEventArgs`, which has a property named `RenderingTime` of type `TimeSpan`, giving you an elapsed time since the app was started:

Project: ExpandingText | File: MainPage.xaml.cs (excerpt)
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        CompositionTarget.Rendering += OnCompositionTargetRendering;
    }

    void OnCompositionTargetRendering(object sender, object args)
    {
        RenderingEventArgs renderArgs = args as RenderingEventArgs;
        double t = (0.25 * renderArgs.RenderingTime.TotalSeconds) % 1;
        double scale = t < 0.5 ? 2 * t : 2 - 2 * t;
        txtblk.FontSize = 1 + scale * 143;
    }
}

I’ve attempted to generalize this code somewhat. The calculation of \( t \) causes it to repeatedly increase from 0 to 1 over the course of 4 seconds. During those same 4 seconds, the value of \( scale \) goes from 0 to 1 and back to 0, so `FontSize` ranges from 1 to 144 and back to 1. (The code ensures that the `FontSize` is never set to zero, which would raise an exception.) When you run this program, you might see a little jerkiness at first because fonts need to be rasterized at a bunch of different sizes. But after it settles into a rhythm, it’s fairly smooth and there is definitely no flickering.

It’s also possible to animate color, and I’ll show you two different ways to do it. The second way is better than the first, but I want to make a point here, so here’s the XAML file for the ManualBrushAnimation project:

Project: ManualBrushAnimation | File: MainPage.xaml (excerpt)
<Grid Name="contentGrid">
    <TextBlock Name="txtblk"
        Text="Hello, Windows 8!"
        FontFamily="Times New Roman"
        FontSize="96"
        FontWeight="Bold"
        HorizontalAlignment="Center"
        VerticalAlignment="Center" />
</Grid>

Neither the `Grid` nor the `TextBlock` have explicit brushes defined. Creating those brushes based on animated colors is the job of the `CompositionTarget.Rendering` event handler:

Project: ManualBrushAnimation | File: MainPage.xaml.cs (excerpt)
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        CompositionTarget.Rendering += OnCompositionTargetRendering;
    }
}
As the background color of the Grid goes from black to white and back, the foreground color of the TextBlock goes from white to black and back, meeting halfway through.

The effect is nice, but notice that two SolidColorBrush objects are being created at the frame rate of the video display (which is probably about 60 times a second) and these objects are just as quickly discarded. This is not necessary. A much better approach is to create two SolidColorBrush objects initially in the XAML file:

Project: ManualColorAnimation | File: MainPage.xaml (excerpt)

```xml
<Grid>
  <Grid.Background>
    <SolidColorBrush x:Name="gridBrush" />
  </Grid.Background>

  <TextBlock Text="Hello, Windows 8!"
    FontFamily="Times New Roman"
    FontSize="96"
    FontWeight="Bold"
    HorizontalAlignment="Center"
    VerticalAlignment="Center">
    <TextBlock.Foreground>
      <SolidColorBrush x:Name="txtblkBrush" />
    </TextBlock.Foreground>
  </TextBlock>
</Grid>
```

These SolidColorBrush objects exist for the entire duration of the program, and they are given names for easy access from the CompositionTarget.Rendering handler:

Project: ManualColorAnimation | File: MainPage.xaml.cs (excerpt)

```csharp
void OnCompositionTargetRendering(object sender, object args)
{
    RenderingEventArgs renderingArgs = args as RenderingEventArgs;
    double t = (0.25 * renderingArgs.RenderingTime.TotalSeconds) % 1;
    t = t < 0.5 ? 2 * t : 2 - 2 * t;

    // Background
    byte gray = (byte)(255 * t);
    Color clr = Color.FromArgb(255, gray, gray, gray);
    contentGrid.Background = new SolidColorBrush(clr);

    // Foreground
    gray = (byte)(255 - gray);
    clr = Color.FromArgb(255, gray, gray, gray);
    txtblk.Foreground = new SolidColorBrush(clr);
}
```
At first this might not seem a whole lot different because two Color objects are being created
and discarded at the video frame rate. But it’s wrong to speak of objects here because Color is a structure
rather than a class. It is more correct to speak of Color values. These Color values are stored on the
stack rather than requiring a memory allocation from the heap.

It’s best to avoid frequent allocations from the heap whenever possible, and particularly at the rate
of 60 times per second. But what I like most about this example is the idea of SolidColorBrush objects
remaining alive in the Windows Runtime composition system. This program is effectively reaching
down into that composition layer and changing a property of the brush so that it renders differently.

This program also illustrates part of the wonders of dependency properties. Dependency
properties are built to respond to changes in a very structured manner. As you’ll discover, the built-in
animation facilities of the Windows Runtime can target only dependency properties, and “manual”
animations using CompositionTarget.Rendering have pretty much the same limitation. Fortunately,
the Foreground property of TextBlock and the Background property of Grid are both dependency
properties of type Brush, and the Color property of the SolidColorBrush is also a dependency
property.

Indeed, whenever you encounter a dependency property, you might ask yourself, “How can I
animate that?” For example, the Offset property in the GradientStop class is a dependency property,
and you can animate it for some interesting effects.

Here’s the XAML file for the RainbowEight project:

Project: RainbowEight | File: MainPage.xaml (excerpt)
murad 2016/02/19

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock Name="txtblk"
    Text="8"
    FontFamily="CooperBlack"
    FontSize="1"
    HorizontalAlignment="Center">
    <TextBlock.Foreground>
      <LinearGradientBrush x:Name="gradientBrush">
        <GradientStop Offset="0.00" Color="Red" />
        <GradientStop Offset="0.14" Color="Orange" />
        <GradientStop Offset="0.28" Color="Yellow" />
        <GradientStop Offset="0.43" Color="Green" />
        <GradientStop Offset="0.57" Color="Blue" />
        <GradientStop Offset="0.71" Color="Indigo" />
        <GradientStop Offset="0.86" Color="Violet" />
        <GradientStop Offset="1.00" Color="Red" />
        <GradientStop Offset="1.14" Color="Orange" />
      </LinearGradientBrush>
    </TextBlock.Foreground>
  </TextBlock>
<GradientStop Offset="1.28" Color="Yellow" />
<GradientStop Offset="1.43" Color="Green" />
<GradientStop Offset="1.57" Color="Blue" />
<GradientStop Offset="1.71" Color="Indigo" />
<GradientStop Offset="1.86" Color="Violet" />
<GradientStop Offset="2.00" Color="Red" />
</LinearGradientBrush>
</TextBlock.Foreground>
</TextBlock>
</Grid>

A bunch of those GradientStop objects have Offset values above 1, so they’re not going to be visible. Moreover, the TextBlock itself won’t be very obvious because it has a FontSize of 1. However, during its Loaded event, the Page class obtains the ActualHeight of that tiny TextBlock and saves it in a field. It then starts a CompositionTarget.Rendering event going:

Project: RainbowEight | File: MainPage.xaml (excerpt)

```csharp
public sealed partial class MainPage : Page
{
    double txtblkBaseSize;  // ie, for 1-pixel FontSize

    public MainPage()
    {
        this.InitializeComponent();
        Loaded += OnPageLoaded;
    }

    void OnPageLoaded(object sender, RoutedEventArgs args)
    {
        txtblkBaseSize = txtblk.ActualHeight;
        CompositionTarget.Rendering += OnCompositionTargetRendering;
    }

    void OnCompositionTargetRendering(object sender, object args)
    {
        // Set FontSize as large as it can be
        txtblk.FontSize = this.ActualHeight / txtblkBaseSize;

        // Calculate t from 0 to 1 repetitively
        RenderingEventArgs renderingArgs = args as RenderingEventArgs;
        double t = (0.25 * renderingArgs.RenderingTime.TotalSeconds) % 1;

        // Loop through GradientStop objects
        for (int index = 0; index < gradientBrush.GradientStops.Count; index++)
        {
            gradientBrush.GradientStops[index].Offset = index / 7.0 - t;
        }
    }
}
```

In the CompositionTarget.Rendering handler, the FontSize of the TextBlock is increased based on the ActualHeight property of the Page, rather like a manual version of Viewbox. It won’t be the full height of the page because the ActualHeight of the TextBlock includes space for descenders and diacriticals, but it will be as large as is convenient to make it, and it will change when the display switches orientation.
Moreover, the CompositionTarget.Rendering handler goes on to change all the Offset properties of the LinearGradientBrush for an animated rainbow effect that I’m afraid can’t quite be rendered on the static page of this book:

![Image of animated rainbow effect](image.png)

You might wonder: Isn’t it inefficient to change the FontSize property of the TextBlock at the frame rate of the video display? Wouldn’t it make more sense to set a SizeChanged handler for the Page and do it then?

Perhaps a little. But it is another feature of dependency properties that the object doesn’t register a change unless the property really changes. If the property is being set to the value it already is, nothing happens, as you can verify by attaching a SizeChanged handler on the TextBlock itself.
This page intentionally left blank
Index

A
About box, 894–897
abstract classes, 117
acceleration
   calculating, 972
   two-dimensional, 970
acceleration vectors
   of bubble level, 968
   converting to 2D coordinates, 968
   magnitude, 968
   of rapid movement, 966
   X and Y components, 968–969
AccelerometerAndSimpleOrientation program, 962–969
Accelerometer class
   GetCurrentReading method, 958
   GetDefault method, 958
   instantiating, 958
   MinimumReportInterval property, 959
   ReadingChanged handler, 959
   Shaken event, 966
   SimpleOrientationSensor, correspondence between, 965
AccelerometerReadingChangedEventArgs, 959
AccelerometerReading class, 958
   AccelerationX, AccelerationY, and AccelerationZ properties, 959–960
accelerometers, 958–969
   compass data and, 976–986
   current value, obtaining, 958
   vector readings, 959–960
AcceptsReturn property, 185
AccumulateDelta method, 675
Action delegates, 214
   Action<object> delegate, 214
ActualHeight property, 85–86
ActualWidth property, 85–86
Add buttons, 320–321
AddCharacterCommand property, 215–216
AddHandler method, 79
Add method, 468
AddPagesComplete events, 910
AddPagesEventArgs, 911
AddPages handler, 910, 920
affine transforms, 379. See also transforms
   angles between lines, 406
   parallel line preservation, 406
   standard matrix representation, 414
   two-dimensional, 437
Alice's Adventures in Wonderland (Carroll), 345, 859
Alignment property, 884
AllColorsItemsControl project, 493–495
   snap view, 495
AllowDrop property, 599
All property, 474
AlphabetBlocks program, 187–192
Alt+F4 key combination, 245–246
altitude, 986–999
AnalogClock program, 401–405
   angles, calculating, 405
   path markup syntax, 403
   positioning on screen, 401–402
   second hand, 405–406
AngleIncrement constant, 649
Angle property
   of RotateTransform class, 378
   setting in XAML, 380
   setting with data binding, 380–381
   AngleX and AngleY properties, 406–409
AnimateDashOffset project, 343–345
AnimatedPieSlice project, 364–367
AnimateStrokeThickness project, 342–343
animation classes, 91
AnimationEaseGrapher project, 351–358
   code-behind file, 354–357
   XAML file, 352–354
animations

animations, 329–376
all-XAML animations, 359–363
of attached properties, 347–350
autoreversing, 335
in background threads, 330
basic animations, 330–333
on button visual states, 516
code, defining in, 338–340
of colors, 92–96
completion, notification of, 338
CompositionTarget.Rendering events, 91–92
control appearance, changing, 329
of custom classes, 364–367
of dependency properties, 94–95
double animations, 340–347
duration, 332, 369
easing functions, 336–337, 350–359
of Ellipse class, 341–342
event handling, 89–96
of fill, 371–372
flicker-free, 83
frame-based animations, 330
on gradients, 702
heap allocations and, 94
jiggling, 386–387
key frame animations, 367–371
linear, 329, 333–336, 350
manual, 329
of Object class, 371–372
objects, creating in XAML file, 93–94
on Opacity property, 345
predefined, 373–376
repeating, 335–336
restarting, 333, 334
in secondary threads, 331
speed, 330
speeding up, 338
springiness, 337
target, 330, 339
target property, releasing, 334
time-based, 330
transfer functions, 351–358
triggering, 387, 436–437
triggering in Loaded event, 339, 341, 359–363
Triggers section, defining in, 372
in user-interface thread, 331
values outside From and To settings, 357–358
of visibility, 371–372
on visual state elements, 513
zooming, 998
anonymous methods, 288
antialiasing, 758
AppBarButtonStyle style, 271–276
Setter object, 271
TargetType, 283
AppBar class, 261, 268–271. See also application bars
BottomAppBar property, 268
Content property, 268–269
height, 269
IsOpen property, 705–706
IsSticky property, 270
Opened and Closed events, 271, 292
TopAppBar property, 268
in visual tree, 270
AppBarPad project, 286–293
file I/O logic, 292–293
font size increases and decreases, 288–289
Wrap Options button, 289
Wrap Options button handler, 290–291
App class, 16, 30
keyboard events, 573
mouse accelerators, 573
navigation state, saving and restoring, 569
OnLaunched method, 557
page shared data, 575
view models, instantiating, 612
application bars, 261, 268–271. See also AppBar class
buttons, 282–283, 286–293
buttons, positioning, 287
button styles, 271–276
button styles, listing, 273–276
CheckBox controls, listing, 273–276
CheckButton controls on, 283–286
coloring, 270
controls on, 269–270
dismissing, 270–271, 292
height, 269
in Internet Explorer, 269
location, 268
New (or Add) buttons, 287
PopupMenu and Popup with, 285–286, 289–290
program options, positioning, 287
RadioButton controls, 283–286
Segoe UI Symbol font for, 276–283
visibility, 268
Application class, 47
LoadComponent method, 25
Resources dictionary, 46
Resources property, 43
Resuming events, 246–247
Suspending events, 246–247
ApplicationData class, 234, 800
LocalFolder, RoamingFolder, and TemporaryFolder properties, 234
LocalSettings and RoamingSettings properties, 234
ApplicationDataContainer, 234
container feature, 571
Values property, 240
ApplicationForegroundThemeBrush color, 520
ApplicationForegroundThemeBrush resource identifier, 47
application isolated storage, 249, 308
application local storage, 234
font files in, 852–856
InkManager contents, saving, 1041–1043
page state, saving, 571–572
saving unsaved data in, 246
settings, saving, 733
TempState directory, 773
application packages, 824
ApplicationPageBackgroundThemeBrush color, 520
ApplicationPageBackgroundThemeBrush resource identifier, 16, 47
applications
About box, 894–897
Application derivative, 16
business logic, 193, 206
code, 31. See also code content, 193
CoreWindow objects, 184
data providers, 193
deploying, 824
display modes, 549–554
event processing, 221
full screen mode, 539
images, binding to, 12
language interoperability, 129
layers of, 193
layout, 7
libraries, referencing, 128, 130
library projects in, 127
lifecycle issues, 245–249
markup, 31. See also markup
multipage, 561–562
orientation awareness, 539, 554–557
orientation preferences, 556
Page derivative, 16
page-navigation structure, 5
page structure, 5
permissions, 894
presentation layer, 193
private fonts, 847
registering for printing, 904
separation of concerns, 193, 200
settings, locating, 240
settings, saving, 288, 308–311, 742, 802
settings, storing, 234, 240, 308
Share providers, 903
Share Targets, 903
snap mode awareness, 539
snap modes, 549–554
suspension, 246–247
termination, 245–246
termination, abnormal, 246
themes, setting, 128
unsaved data, saving, 245–248
windows, 5
application state
restoring, 568–572
saving, 568–572
ApplicationStateSave project, 569–572
OnLaunched method, 573
ApplicationView class
TryUnsnap method, 554
Value property, 549
ApplicationViewState enumeration, 550–551
Filled value, 551, 553
FullScreenLandscape value, 551
FullScreenPortrait value, 550
Snapped value, 552, 553
application view transitions, 329
AppSettings class, 308, 733–734, 736
AutoParsing property, 316
bindings to, 325
EditOrientation property, 308
Orientation property, 311
properties of program settings, 308–311
SwapEditAndDisplay property, 311
App.xaml.cs files, 16
OnLaunched override, 29
App.xaml files, 16
resources, defining, 46
style definitions, 937
architectural patterns, 193
arcs
algorithms for WriteableBitmap, 722–747
rendering, 101
ArcSegment class, 55
IsLargeArc property, 642
Point type properties, 361
ArcSegment structure, 726–727
ARGB colors, 17
ARM-based machines, 823
Arrange method, 485
ArrangeOverride method, 485, 487, 492
finalSize method return, 493
arrow keys as accelerators, 575
AsAsyncAction method, 243
Ascender Corporation, 847
Asin method, 641
aspect ratio

aspect ratio
  Grid, responding to changes in, 295
  ignoring, 11, 58
  layout, adjusting, 152–154
  maintaining, 10
  preserving, 396
  properties backed by dependency properties, 296
Assembly objects, 115–116
Assets folder, 9, 13
AsStringStream method, 686
AsTask method, 243
asynchronous methods, 221–222
  cancellation in, 231–233, 252, 260
  custom, 250–260
  data, returning to program, 225
  for disk access, 233
  error trapping, 230
  for file I/O, 243–245
  for file picker display, 234
  interface hierarchy, 226
OpenReadAsync method, 238
page initialization, calling during, 231
progress reporting in, 231, 252–254, 260
RunAsync method, 225
asynchronous processing, 222
  application lifecycle issues, 245–249
  Async calls, consolidating, 241–243
  await operator, 229–231
  callbacks as lambda expressions, 228
  cancelling operations, 231–233
  errors and, 231
  exception handling, 240–241
  file I/O, 233–235
  MessageDialog examples, 222–227
  .NET support, 242–243
  progress reporting, 231
async keyword, 230
  in lambda functions, 251, 256
Async method calls, 222–223
asyncOp objects, saving as field, 232
AsyncStatus enumeration, 224, 232
Async suffix, 221, 222
Atan2 method, 724
attached properties, 132–136
  animating, 347–350
  AutomationProperties class, 272
  creating, 168
  in custom panels, 484
Grid.Row and Grid.Column properties, 147, 149
Grid.RowSpan and Grid.ColumnSpan properties, 147, 150
  text-enhancing, 856–857
ZIndex, 136–137

AttachedPropertyAnimation project, 347–350
audio files, 22
AutoImageSelection project, 546–549
automation peers, 890
AutomationProperties class, 272
AutoReverse property, 335, 341
AutoRotationPreferences property, 556, 957
availableSize argument, 486, 487
  finite Width and Height properties, 493
  infinite Height property, 496
await operator, 229–231
  callback method creation, 241
  deferral objects, 230
  flagging methods for, 230
  for long-running jobs, 250–251
  restrictions on, 230
  in Task.Run, 256
  in Task.Yield, 259–260
  in try blocks, 232
AxisAngleRotation program, 981–986
azimuth, 986–999
  calculating, 990

B

Back buttons, 562–563, 585
  Click handler, 895–896
  disabled, 601–602
BackButtonStyle, 585
BackEase class, 357
background
  colors contrasting with, 47
  of Grid, 7, 16
  light, 16
  of margins, 101
BackgroundCheckedGlyph, 283
Background property, 31
  null default value, 82
  of Slider controls, 754
  TemplateBinding on, 508
  Transparent setting, 82
background threads. See also secondary threads
  animations running in, 330
Backspace key processing, 887–888
back stack, 562–563
  saving and restoring, 568–571
BackStackDepth property, 563, 566
bar charts, 497–499
BareBonesSlider project, 522–524
TemplateBinding, 524
BasedOn property, 63, 65
BaselineTiltedShadow project, 829–831
bitmaps

base method calls, 76
Begin method, 332, 433
BeginStoryboard class, 360
BeginTime property, 359
BerniceBobsHerHair project, 867–869
BetterBorderedText project, 99–100
BetterCharacterInput project, 888–892
BezierControlPoint1 property, 1019
BezierControlPoint2 property, 1019
Bézier curves
  connecting, 344–345
  Cubic Bézier, 345
  ink as, 1018
  InkStrokeRenderingSegment objects, 1014–1015
  line thickness, 1023
  Polyline, converting from, 1021
  quarter-circle, 344–345
  skewing, 406
  Smooth Bézier, 345
BezierSegment class, 55–56, 361
BezierSegment properties, 1023
Bézier splines, 371
BindableBase class, 201–202
binding. See data bindings
Binding class
  ConverterLanguage property, 110
  ConverterParameter property, 110
  Converter property, 109–111
  ElementName property, 67
  Mode property, 199
  Path property, 68
  syntax, 112
binding converters, 154
  BooleanToVisibilityConverter binding
  converter, 461–462
Binding markup extension, 66, 88, 112, 474
  on Fill property, 452
  in property-element syntax, 67
Bing Maps
  map tiles, 1005–1006
  quadkey numbering, 1006–1007
Bing Maps SOAP service, 1000
  manual use, 1002
  quadkey, obtaining, 1007
BitmapAlphaMode property, 709
BitmapCodecInformation objects
  FriendlyName property, 710
BitmapEncoder class, 704
  CreateAsync method, 712
  file format GUIDs, 704
  GetEncoderInformationEnumerator method, 710
  SetPixelData method, 712
BitmapFrame class
  BitmapAlphaMode property, 709
  BitmapPixelFormat property, 709
BitmapImage class, 27, 683
  creating in code, 28
  BitmapPixelFormat property, 709
  BitmapPrintDocument class, 948–951
  current bitmap, obtaining, 949
  instantiating, 949
bitmaps, 683–777. See also images
  Alpha setting, 691
  A, R, G, and B value formulas, 691
  in buttons, 450
  camera photos, capturing, 772–777
  in celestial sphere, 992–999
  clipboard support, 898
  color format, 684
  creating, 12
  displaying, 9–13, 687
  displaying from code, 27–29
  file formats, 703–704
  HSL color selection, 747–758
  images on top of, 758
  invalidating, 687
  larger than screen, viewing, 992–999
  line drawing on, 724–747
  loading, 692–694, 703–714, 736–742
  monochromizing, 714–721
  Pictures library, 763–772
  pixel bits, 684–690
  pixel dimensions, 104
  pixel dimensions, calculating, 692
  pixel formats, 709
  pixel sizes, 545–546
  posterizing, 714–721
  premultiplied alphas, 684, 691–696
  printing, 948–951
  program logo bitmaps, 13
  radial gradient brushes, 696–703
  resetting, 687–688
  resolution scale, autoselection of, 546–549
  resolution settings, saving, 706, 708
  reverse painting, 758–762
  saving, 646, 703–714, 736–742
  stretching, 58–60
  transparency, 691–696
  updating pixels, 688
  zooming, 998
**BitmapSource class**

BitmapSource class, 683, 692
- PixelHeight and PixelWidth properties, 684
- BITRES value, 1007
- Blank App template, 3, 557
- Block class, 41
- Blocks property, 858
- Bold class, 42
- BooleanToVisibilityConverter binding converter, 461–462

BorderBrush, 505
- predefined identifiers, 128

Border element, 97–101
- Child property, 98
- for context menus, 268
- in ControlTemplate, 504–505
- Grid in, 146
- HorizontalAlignment and VerticalAlignment properties, 99
- Loaded handler on, 267
- Padding property, 100–101
- for printer pages, 911
- RadioButton controls in, 161
- for StackPanel, 266
- TextBlock elements in, 165
- width, 165

BorderThickness property, 98, 505
- Bosch, Hieronymus, 993

BottomAppBar property, 268
- BOUNCE constant, 972

browser keys, 574–575

Brush class
- RelativeTransform property, 422
- Transform property, 422

brushes
- animating, 423–427
- class hierarchy of, 31
- defining with styles, 62
- ImageBrush, 690
- radial gradient brush, 696–703
- sharing, 43–47
- SpreadMethod setting, 425
- ToString representation, 451
- transforms on, 422–427
- for visual states, 521

Brush type, 31
- BubbleLevel program, 966–969
- Buffer class fully qualified name, 694
- bulk access, 235, 237
- business logic, 193
- isolating, 206

ButtonBase class
- classes deriving from, 159
- Click events, 140, 161
- ClickMode property, 161
- CommandParameter property, 212
- Command property, 212
- Content property, 162
- HorizontalAlignment and VerticalAlignment properties, 160
- Margin property, 160

Button controls, 139, 159–167
- appearance, manipulating, 162
- on application bars, 271–276, 286–293
- BorderBrush property, 505
- BorderThickness property, 505
- Cancel buttons, 238, 318
- Click handlers, 166, 237–238, 258, 566
- Command bindings on, 218–219
- Command property, 212–213
- content of, 141
- Content property, 450–459
- ContentTemplate property, 451–452
- Copy and Paste buttons, 898
- custom, defining, 222
- data content, 450–459
- default appearances and functions, 159–162, 223–224
- dependency properties, defining, 167–177
- Don’t Save buttons, 318
- enabling and disabling, 258, 933–934, 1011
- EntranceThemeTransition, 512
- forward and back buttons, 558–559
- images in, 162
- implicit style, 165–166
- JiggleButton, 386–387
- keyboard input focus, 513
- MVVM pattern and, 212–213
- names, displaying, 272
- OnCharButtonClick event handler, 167
- Open buttons, 320–321
- Paste buttons, 898–899
- RenderTransformOrigin property, 387
- RenderTransform property, 387
- Resources property, 387
- Save As buttons, 318–319
- Save buttons, 318–319
- Segoe UI Symbol font for, 271–272
- static visuals, 504
- style, overriding, 165
- View Model, calling into, 212–213
- visual states, 513–520
- in visual tree, 503

ButtonVarieties program, 159–162

By property, 337
- ByteToHexStringConverter converter, 470
Charms, 893
  Devices charm, 904–911
delays.
  displaying, 286
  hooking into, 896
  program invocation, 928
  Settings charms, 894–897
  Share charm, 898–902

Canvas.
  Canvas.Left and Canvas.Top attached properties, animating, 347–350, 358–359
  Canvas.Left and Canvas.Top attributes, 133
  Canvas.SetLeft and Canvas.SetTop static methods, 135
  children, arrangement of, 97
  children, size of, 137
cing, avoiding, 137–138
  DependencyProperty property, 135
element-positioning properties, 132
  HorizontalAlignment and VerticalAlignment properties, 137
  layout in, 137–138
  pagination and, 921
  SetLeft and SetTop methods, 135, 136
  TextBlock, positioning, 306
  ZIndex property, 136–137, 435, 436
  capacitance pens, 1014
CaptureFileAsync method, 772–773
CapturePhotoToStorageFileAsync method, 775
CapturePhotoToStreamAsync method, 775
CapturePointer method, 628
Carroll, Lewis, 345, 859
Ceiling method, 921
celestial sphere, 986
alitude, 986
  azimuth, 986
  bitmaps in, 992–999
  horizontal coordinate, 987
  nadir, 986
CenteredTransforms project, 673–674
center of rotation, 378, 398. See also rotation
  protection radius around, 678
  for single-finger rotation, 678
  specifying, 383–386, 391–396
  for touch interfaces, 673–676
  CenterOfRotationX, CenterOfRotationY, CenterOfRotationZ properties, 434
  CenterX and CenterY properties
  handlers for, 385
    of RotateTransform class, 383–384
    of ScaleTransform class, 398
  CharacterFormat property, 877
  CharacterReceived events, 184, 887
  handling, 889–890
char data type, 277
cations, 893
  Devices charm, 904–911
delays.
  displaying, 286
  hooking into, 896
  program invocation, 928
  Settings charms, 894–897
  Share charm, 898–902

C
C#
  anonymous methods support, 228
  async keyword, 230
  in code-behind files, 6
  data type equivalence to Windows API, 780–781
  function declarations, 782
  .NET API access, 779
  platforms, selecting, 822
  public fields, 781
  static functions, 782
  structures, defining, 781
  Tapped event, 70
  wrapper DLLs, accessing, 779
C++
  native machine code, compiling to, 822
  Platform namespace runtime libraries and classes, 779
  platforms, 823
  public classes, 810
  wrapper DLLs, 779
CalculateImageScaleAndOffset method, 758
CalculateNewTransform method, 445
Calendar class, 930–931
calendars, 928–936
  callback methods, 223, 224
    IUICommand object, obtaining, 229
    as lambda expressions, 228
    running in UI thread, 227
callbacks, 83–84
  CallerMemberName attribute, 202
camera application, creating, 774–777
CameraCaptureUI class, 772
camera, capturing photos from, 772–777
Cancel buttons, 238, 318
CancelCommandIndex property, 224
CancellationToken type, 252–253, 255
Cancel method, 223
  of IAsyncInfo interface, 231–232
CanExecuteCalculate method, 215
CanExecuteChanged handler, 213
CanExecuteDeleteCharacter method, 218
CanExecute method, 213, 215
  Func<object, bool> delegate, 214
CanGoBack and CanGoForward properties, 557, 558
  as binding sources, 560
CanRecorderItems property, 599
Canvas, 132–136
  animating, 347–350
  attached properties, 132–136
Char structure

Char structure, 277
  ConvertFromUtf32, 281
char values, 887
CheckBox controls, 161
  styling, 285
Checked and Unchecked events, 161
Checked handlers, 180–181, 265, 274
CheckIfOKToTrashFile method, 320, 900–901
CheshireCat project, 345–347
  children
    dependency property, setting on, 135–136
    and parents, balancing needs, 97
    size of, calculating, 490
    stacking, 103–104
Children property, 24, 103
  of Storyboard class, 337
  of TransformGroup class, 391
ChildrenTransitions property, animating, 376
CircleAnimation project, 358–359
circles. See also Ellipse class
  quarter-circle arcs, 344–345
  rendering, 102, 105
CircularGradient project, 688–690
Class1.cs files, 127
classes
  abstract classes, 117
  attached properties, creating, 168
  content properties, 38–41
    dependency properties, defining, 168–177
    nesting, 42
    order of elements in code, 172
  protected constructors, 117
class hierarchies, 114–118
Class Library template, 129
Click handlers, 166, 212, 928
  animations, triggering, 387
  for Back buttons, 895–896
  for buttons, 566
  in context menus, 265
  for file picker Open button, 237–238
  lambda functions in, 228
  null values, 242
  for Start buttons, 258
clicking, enabling, 599
ClickMode property, 161
clipboard
  bitmaps, copying and pasting, 898
  contents, checking, 899
  copies and pastes, 877
  Ctrl+C and Ctrl+V support, 902
  ink, copying to, 1037
  pasting into InkManager, 1029
  sharing data with, 898–902
Clipboard class, 899
  GetContent method, 899
  SetContent method, 899
ClockButton project, 456–459
Clock class, 456–459
ClockRack program, 786–808
  Add menu item, 803
  clock size, 794
  Delete item, 804–805
  DistributedUniformGrid class, 799–800
  Edit and Delete options, 802–803
  Edit option, 804–805
Closed event handler, 327
Close method, 223
code, 31
  animations, defining in, 338
  application bars, dismissing, 270–271
  attached properties, setting, 133
  Auto or star sizes, specifying, 147
  Binding object, creating, 67–68
  callback methods, 223
  element properties, changing, 69
  font files, referencing, 856
  FontStyle property, setting, 852
  gradient brush, 31–33
  Grid, accessing, 28
  images, displaying, 27–29
  items, generating, 467–468
  ItemsSource property, setting, 471–472
  Path element in, 56–57
  PopupMenu objects, constructing, 262
  private fonts, referencing, 850
  queuing, 225
  RichTextBoxOverflow, generating, 867
  styles, defining, 62
  test, setting, 184
  TextBlock, creating, 24–27
  View Model, instantiating, 205
code-behind files, 6
  elements, accessing, 69
  event handlers in, 71
  Grid, accessing from, 23
  minimizing use of, 194
  of XamlCruncher, 313–315
codecs, 704
CollectionChanged events, 475, 591
collection controls, 463–474
collections, 474
  Dictionary<TKey, TValue> collections, 474–475
  displaying, 141
  of groups, 611
  items, accessing, 474
  items, adding and removing, 591
collections (continued)
   items, displaying, 449, 464, 477–478, 596–597, 600
   items, grouping, 608–611
   ItemsSource property, binding to, 473
   List<T> collections, 474–475
   movement of items, 329
   selecting items, 475–480
CollectionViewSource class, 611
ColorAnimation class, 361
ColorAnimationUsingKeyFrames class, 369–370
ColorItems project, 467–469
ColorItemsSource project, 470–472
ColorItemsSourceWithBinding project, 473–474
ColorKeyFrame class, 369
ColorList1 project, 122–124
ColorList2 project, 124–126
ColorList3 project, 127–129
Color property, 157
   animating, 331
   binding on, 453
   public, 755
   setting from outside, 757
colors
   animating, 92–96, 331
   color format of bitmaps, 684
   gradients between, 32
   highlight colors, 47
   HSL color selection, 747–758
   Hue, 747
   Lightness, 747
   listing, 122–124
   Saturation, 747
   specifying, 16–17
   transparent black, 762
   transparent white, 762
color schemes, 16
Colors class, 17, 588
   using directive, 23
ColorScrollWithDataContext project, 202–206
ColorScrollWithValueConverter project, 155–157
ColorScrollWithViewModel project, 196–201
COLORSCR program, 148
ColorSettingDialog class, 743–744, 757
Color structure, 835
ColorTextBoxes project, 206–208
ColorTextBoxesWithEvents project, 209–211
ColorToContrastColorConverter, 494
ColorWrap project, 130–131
ColumnDefinition objects, 146
ColumnDefinitions collection, 146
columns, 131
COM API, 779

ComboBox controls, 464, 805, 807
   processing, 1054–1056
ComboBoxItem class, 535
COM (Component Object Model), 779
commanding, MVVM architecture and, 193
command interface, 212–213
CommandParameter property, 212
Command property, 212–213, 218–219
commands
   on application bars, 268
   in context menus, 264
   navigating with Tab key, 268
   processing, 897
   sharing, 218
   for View and View Model interactions, 194
   in View Model, 213–219
CommandsRequested handler, 896
Common folder
   RichTextColumns class, 867
   StandardStyles.xaml file, 46
CommonMatrixTransforms project, 417–418
Compass class, 974–976, 980
   accelerometer data and, 976–986
CompassReading class, 974
Completed callback method, 224
Completed events, 338, 579
Completed handler, 226, 580
   running, 228
Completed property, 223
Complete method, 663
CompositeTransform class, 379, 419–421
composite transforms, 418–421
CompositionTarget.Rendering events, 91–96, 329, 802,
   841–843
ComPtr, 810
computers
   geographical location, 671–672, 999–1012
   screen resolution, 539–545
   sensory hardware, 953–1012
ConditionalClockButton project, 460–463
conditional execution in XAML, 460–463
constructor
   defining, 126
   DependencyProperty objects, creating, 168
   initializing components in, 150
   Loaded processing, 86
   public parameterless, 474
ContactRect property, 633
Container property, 677
containers, filling with images, 11–12
content, 193. See also data
docking in Grid, 158
ContentControl class

- ContentControl class, 141, 472
  - AppBar, 261
  - ContentPresenter class, 509, 667–669
  - Content property, 141, 509
  - ContentTemplate property, 451, 467
- ContentPresenter class, 509–510, 667–669
  - bindings on, 510
  - Content property, 509
  - ContentTemplate property, 510
  - ContentTransitions property, 512
  - HorizontalAlignment and VerticalAlignment properties, 511
  - Margin property, 510
  - positioning within parent, 512
- content properties, 37–41
  - definition, 39
  - TextBlock content property, 41–43
- Content property, 28, 38, 97, 564
  - of AppBar class, 268–269
  - of Button class, 162–163, 450–459
  - of ContentControl class, 141, 509
  - of ContentPresenter class, 509
  - of RadioButton controls, 160
  - of UserControl, 125
- ContentProperty attribute, 38
- ContentPropertyAttribute class, 38
- ContentTemplate property, 451–452, 467
  - of ContentPresenter, 510
  - DataTemplate on, 459
- ContentTransitions property
  - animating, 375
  - bindings on, 512
- ContextMenuOpening events, 862
- context menus, 261–264
  - Border for, 268
  - commands in, 264
  - creating, 631
  - dismissing, 267
  - displaying, 267, 632
  - horizontal lines, 263
  - keyboard interface, 264
  - location, 263–264, 266
  - navigating with Tab key, 268
  - positioning, 267
  - of TextBox control, 262
- continuation handlers, 594
- Control class, 139–141
  - classes deriving from, 139–141
  - event interface, 75
  - Focus method, 140
  - FontFamily property, 845
  - Foreground property, 162
  - HorizontalContentAlignment property, 140, 511
- IsEnabledChanged event, 140
- IsEnabled property, 140
- IsTabStop, TabIndex, and TabNavigation properties, 140
- OnGotFocus and OnLostFocus virtual methods, 140
- On methods, 615, 656
- Padding property, 510
- properties of, 140
- protected virtual methods, 140
- Template property, 139, 502, 503
- VerticalContentAlignment property, 140, 511
- controls
  - appearance, defining, 329
  - appearance, redefining, 139, 140
  - on application bars, 270
  - automation peers, 890
  - bindings to elements, 506–508
  - buttons, 159–167. See also Button controls
  - chrome, 502–503
  - collection controls, 463–474
  - in context menus, 261
  - custom. See custom controls
  - dependency properties and, 167–177
  - disabled visual state, 516
  - vs. elements, 14
  - event handlers, sharing, 167, 177
  - hit testing, 649–650
  - identifying, 177
  - input focus, 140, 144, 184, 516
  - interaction with user, 139, 140
  - items in collections, accessing, 474
  - keyboard input, 184–187
  - pointer input, handling, 615
  - RadioButton, 177–183. See also RadioButton controls
  - separate instances of, 937
  - Setter objects on, 503
  - size, defining, 653
  - Slider, 141–145, 154–159. See also Slider controls
  - Style definitions, 521
  - templates on, 502–512
  - TextBox, 184–187
  - Thumb, 187–192
  - in View, binding to properties, 196
  - visual appearance, 14
  - visuals, customizing, 449
- ControlTemplate class, 449, 502–512
  - of AppBarButtonStyle, 271
  - Border in, 504–505
  - button visual states, 513–520
  - definitions in generic.xaml, 520
  - hard coding in, 508
  - as a resource, 503
custom panels

ControlTemplate class (continued)
for Slider controls, 754
as a Style, 506–507
TargetType property, 503
TemplateBinding, 506
converter class, 155–157
ConverterLanguage property, 110
Converter property, 109–110
Convert method
parameter and language arguments, 110–111
TargetType argument, 110
value argument, 110
coordinate systems
device, 981
Earth’s, 981
hardware, 960–961
right-hand rule, 960
translating between, 989–990
of windowing environment, 33, 49
Copy buttons, 898–899
Ctrl+C support, 902
Copy command, 898
CopySelectedToClipboard property, 1029
CoreDispatcher class, 225
callbacks, 228
RunAsync method, 225
CoreDispatcher objects, 594, 950
user-interface thread, interacting with, 954
CoreVirtualKeyStates enumeration, 575
CoreWindow class, 184, 616
AcceleratorKeyActivated event, 572–573
CharacterReceived events, 887
CornerRadius property, 508
C Programming Language, The (Kernighan and Ritchie), 3
CreateAsync method, 708, 712
CreateFileAsync method call, 248
CreateItemListOption method, 944
CreatePrintTask method, 907, 948
CreateTextOption method, 944
CreationCollisionOption, 248
.cs extension, 4
Ctrl+C and Ctrl+V support, 902
Cubic Bézier, 345
CubicEase class, 357
Cumulative property, 663
CurrentOrientation property, 87, 554
of DisplayProperties, 955
Current property, 234
cursor position, obtaining, 307, 882
curves, 53. See also Bézier curves
rendering, 101
CustomButtonTemplate project, 517–520
custom classes
animating, 364–367
BitmapPrintDocument class, 948–951
Clock class, 456–459
Dial class, 678–682
InkFileManager class, 1038–1039
Key class, 650–655
ManipulationManager class, 675–676
NamedColor class, 469–470
PieSlice class, 364–367
PointerInfo class, 622
RadialGradientBrushSimulator class, 696–702
SaveStatePage class, 565–568
SecondPage class, 557–559, 566
StudentBody class, 590–591
StudentGroup class, 609
StudentGroups class, 609–610
SurfaceImageSourceRenderer class, 831–843
TimeZoneManager class, 789
TwelveHourClock class, 460–463
YellowPadPage class, 1043
custom controls, 530–535
application projects, defining in, 535
arranging code, 172
creating, 141
default ControlTemplate, 530
default Style, 530
dependency properties in, 167–177
existing controls, adapting, 502–503
HslColorSelector control, 752–754
keyboard input, 184
library, 531
local prefix, 174
LoggerControl control, 622–624
ManipulableContentControl control, 763
ManipulationModeCheckBox control, 657–659
MonthYearSelect control, 930, 932
NewToggle control, 530–535
property-changed handlers, 173
RudimentaryTextBox, 889–892
UserControl, deriving from, 125, 175–177
XYSlider control, 667–672
CustomGradient project, 685–689
code-behind file, 685
CustomizableClassHierarchy project, 923–927
CustomListBoxItemStyle project, 536–538
CustomPageRange class, 940–942
custom panels, 484–497
Arrange method, 485
ArrangeOverride method, 485
attached properties in, 484
finalSize argument, 492
layout passes, 485, 489
custom panels (continued)

custom panels (continued)
Margin property, 490
Measure method, 485
MeasureOverride method, 485
properties handled automatically, 485
scrolling, 484
size, calculating, 490–491

D

data
automatically saving, 249
in buttons, 450–459
displaying, 194
notifications of updates, 194
passing and returning among pages, 575–581
sharing through clipboard, 898–902
updating upon input focus change, 208–209
data bindings, 66–68
Angle property, setting, 380–381
between dialogs and applications, 268
binding converters, sharing, 111–112
Button calls into View Model, 212–213
changing values, tracking, 144
Command property targets, 212
DataContext property of target, 204–206
data conversion, customizing, 109–111
in DataTemplate, 452–454
dependency property target, 89
document pages, chaining, 863–864
double types, 198
elements to controls, 506–508
items, accessing with, 597
ItemsSource property to collections, 473
MVVM architecture and, 193
notifications, 194–196
Path=, 205
RelativeSource bindings, 480, 506
to Run property, 87–89
source, 67, 177, 194–195
source, specifying, 194, 199, 200, 204–206
source, updating, 199
target, 67, 108–109, 157, 194
target, updating, 199
TemplateBinding bindings, 506
in templates, 449, 452–454
two-way, 506
View and View Model interactions, 194
in View Model, 194–196
on Visibility property, 462
XAML resources, referencing, 198–199

DataContext property, 204–206
bindings on, 805
propagation down visual tree, 204
data conversions, customizing, 109–111
data entry validation, 210–211
DataPackage objects, 899
SetBitmap method, 899–900
DataPackageView objects, 899
DataPassingAndReturning project, 575–581
Data property, 53, 56
null, 57
data providers, 193
DataReader class, 237
IDisposable interface, 238
DataRequested handler, 903
DataTemplate class, 449
adding, 597–598
bar charts, 497–499
of ContentTemplate property, 451–452, 459
data bindings in, 452–454
for ItemControl controls, 463–474
ItemTemplate property, setting to, 466–467
for ListBox controls, 479–480
object rendering, controlling, 458–459
property changes, responding to, 475
Resource section, defining in, 454
sharing, 600–601
DataTransferManager class, 903
DataWriter class, 237
StoreAsync method, 239
DataWriteStoreOperation objects, 239
DateTime property, 799
decoders, 704
DefaultCommandIndex property, 224
DefaultStyleKey property, 530
deferral objects, 230
DelegateCommand class, 213–219
RaiseCanExecuteChanged method, 214
delegates, predefined, 214
DeleteCharacterCommand property, 215–216
DeleteSelected method, 1029
Delta property, 660
DependencyObject class, 15
accessing from thread of execution, 225
Dispatcher property, 225
SetValue method, 135
thread safety, 224–225
DependencyObjectClassHierarchy project, 114–118
dependency properties, 15, 25–26
animating, 94
attached properties, 132–136
as backing for data binding targets, 89, 194
binding sources, 194
dependency properties (continued)
  as binding targets, 67
  children, setting on, 135–136
  for controls, 167–177
  on Canvas, 135
  default value, 15
  defining, 26, 167–177
  defining as private static fields, 169
  existing types for, 297
  in RadialGradientBrushSimulator, 696–698
  Matrix3D type, 443
  properties, specifying independently of class, 62
  property changes, 96
  target properties of animations, 330
DependencyProperties namespace, 174
DependencyPropertiesWithBindings project, 176–177
DependencyPropertyChangedEventArgs objects
  Property property, 171
  DependencyProperty class, 15
  RegisterAttached method, 168
  Register method, 168
  DepthText project, 390–391
designUnitsPerEm field, 827
DesiredDeceleration property, 664
DesiredDisplacement property, 664
DeviceInformation objects, 774
devices. See also printers; tablets
  orientation changes, 9
Devices charm, 904–911
  program invocation, 928
Dial class, 678–682
  Dial controls, 678–682
  Minimum and Maximum values, 680–681
  RotateTransform, 681
dialog boxes
  for file opens and saves, 234–235
  Popup class for, 265–268
DialogPage class, 575–581
  Completed event, 579–580
DialSketch project, 679–682
dictionary
  abandoned entries, 633
  back stack information, 572
  Color type values, 634
  instantiating, 565
  of MIME types, 710
  page state information, 564–567
  per-finger information, 622–627
  Pointer IDs, 618, 619, 621–622, 627
  Pointer IDs, removing, 630
  pointer information, 634, 1024
  removing entries, 567–568
  static pages dictionary, 565, 567
  Win32 function conversions in, 789
Dictionary class, 255
  Remove method, 567
Dictionary< TKey, TValue > collections, 474–475
DigitalClock project, 90
digital stylus, 1014. See also pens
digitizers, 1014. See also pens
Direct3D, 380
directory structure, displaying, 765–766
DirectWrite, 809–821
  DWRITE_FONT_METRICS, 827
  font family names, 815
  font metrics, 825–831, 856
  fonts, enumerating, 847
  pages, rendering, 921
DirectX, 48
  DirectWrite, 809–821
  drawing on bitmaps, 831–843
  HRESULT values, 811
  libraries, 809
  SharpDX library, 808
DirectXWrapper library, 808–809
  correspondence with DirectWrite interfaces, 809–810
  fonts, enumerating, 820–822
  referencing, 820
  SurfaceImageSourceRenderer class, 831–843
  WriteFactory class header file, 810
DirectXWrapper project, 808–809
DiscreteObjectKeyFrame class, 371
  Value property, 372
DiscretePointKeyFrame class, 368, 369
Dispatcher property, 225, 594
DispatcherTimer class, 89, 232
  interval, 405
  for long-running jobs, 250
  timer interval, 90
DisplayAndPrinterPrep method, 915–916
DisplayGrid property, 766
DisplayHighSchoolStudents project, 594–608
  Back button, 601–602
  GridView, 602
  ItemClick events, 605–606
  ListView, 602
  portrait mode, 605–606, 608
  Snapped mode, 604
  StudentPage.xaml file, 606–608
  Visual State Manager markup, 602–603
  visual state, setting, 600
DisplayInformation collection, 791
DisplayInformation property, 792
DisplayMatrix3D class, 442–444
DisplayMemberPath, 477
DisplayOrientations enumeration

DisplayOrientations enumeration, 87, 955
  LandscapeFlipped value, 554
  Landscape value, 554
  members, order of, 556
  None value, 554
  PortraitFlipped value, 554
  Portrait value, 554
DisplayProperties class, 16, 87, 540, 554–557
  AutoRotationPreferences property, 556, 957
  CurrentOrientation property, 554, 955
  LogicalDpiChanged event, 540
  LogicalDpi setting, 541
  NativeOrientation property, 554, 957
  OrientationChanged events, 554
  SimpleOrientationSensor correspondence, 957
DisplayText property, 215
  display, themes, 520
DistributedUniformGrid class, 799–800
  populating, 800–801
DIUs (device-independent units), 545
DllImportAttribute, 781–782
Document property, 877
don't Save buttons, 318
DoubleAnimation class, 330–331, 340–347
  AutoReverse property, 335, 348
  BeginTime property, 359
  By property, 337
  class hierarchy, 337
  Duration property, 339
  EasingFunction property, 336–337
  EnableDependentAnimation property, 337
  FillBehavior property, 334
  From value, 333–335, 348
  RepeatBehavior attribute, 335, 348
  reusing in animations, 339
  To value, 334–335, 349
DoubleAnimation objects, 387
  for brush animations, 427
  duration, 433
  From value, 410
  grouping, 396
  To value, 409, 424
DoubleAnimationUsingKeyFrames class, 369, 410, 436
  Double.IsPositiveInfinity method, 486
  DoubleKeyFrame class, 369
  DoubleTapped events, 615
  DoubleToStringHexByteConverter class, 278
double values
  animating, 331, 340–347
  converting to hexadecimal, 155–157
DPI (dots per inch), 539
  logical DPI, 540
DragCompleted events, 187
DragDelta events, 187, 190
DragEnter events, 70
DragLeave events, 70
DragOver events, 70
DragStarted events, 187
drawing. See also FingerPaint projects; pens
  redraw methods for, 303
DrawingAttributes property, 1019
drawings. See also FingerPaint projects; pens
  saving, 646
DrawLine method, 835
Drop events, 70
Duration property
  overriding, 351
  setting, 339
DWORD_PTR, 781
DWORD values, 780
DWriteCreateFactory function, 810
dynamic layout system, 97. See also panels
DYNAMIC_TIME_ZONE_INFORMATION structures
  TimeZoneKeyName field, 792
E
Earth
  coordinate system, 981
  coordinate system, transforming to rectangular coordinates, 990–999
  longitude and latitude, 986
EarthlyDelights program, 987–999
  Matrix3D inversion, 989–990
  tilt, calculating, 990–992
  zooming operation, 998
Ease function, 351
Ease method, 350
EasingColorKeyFrame, 371
EasingDoubleKeyFrame, 371
EasingFunctionBase class, 336–337
  EasingMode property, 337
  EasingFunction property, 336–337
  easing functions
    elapsed time, 351
    SineEase function, 358
    values outside range of 0 and 1, 357–358
    visual representation, 351
EasingMode function, 387
EasingMode property, 337, 358
EasingPointKeyFrame class, 369, 371
EasyCameraCapture program, 772–774
e-book readers, 870–877
page numbers, 871–873
e-books, creating, 119–122
EditBox controls, 805
EditOrientation enumeration, 322
EditOrientation property, 308
EditOrientationRadioButton controls, 322–327
ElasticEase animation, 387
EaselnOut mode, 393
ElasticEase class, 336, 357
Oscillations property, 337
Springiness property, 337
electromagnetic pens, 1014. See also pens
ElementName property, 67, 194, 204
elements
accessing from code-behind, 69
binding to controls, 506–508
centering, 107–108
clipping, avoiding, 137–138
vs. controls, 14
fading in and out, 345–347
flipping horizontally or vertically, 398
interaction with user, 139
jiggling, 386–387
keyboard focus, 630
layout of, 937
location, finding, 427–430
manipulation, enabling, 656
margins, 100–101
naming, 126
natural size, calculating, 490
offsetting from original position, 388–391
organizing, 20
overlapping, 19–20, 136–137
overlapping, preventing, 20–22
padding, 101
vs. panels, 14
pointer capture, 615, 622–630
pointer input, 627
positioning precisely, 132–136
Position property, 618
properties, linking, 66
rendered size, 85
rendering of, 83
resources, sharing, 43–47
reusing, 937
shearing, 406
size, increasing and decreasing, 396
spinning in space, 431–434
stacking, 19–20
stretching to parent size, 98
styles, 60–65
transforms on, 377. See also transforms
visual state elements, 513
in visual tree, accessing, 143
visual tree of, 14
width and height values, 98
Z order, 20
Eliot, George, 870
EllipseBlobAnimation project, 340–341
Ellipse class, 101–103
animating, 341–342
Height and Width values, 102, 104
rendering, 101–102
Stretch property, 105
StrokeDashArray property, 343
StrokeDashCap property, 343
StrokeDashOffset property, 343–344
EllipseGeometry class
animating, 361–363
positioning element, 970
ElPasoHighSchool project, 589–608
student.xml file, 591–592
El Paso Public Library, 589
EnableDependentAnimation property, 331–332, 337
leaving out, 348
EnclosureLocation property, 774
encoders, 704
EndPoint property, 33
EntranceThemeTransition, 512
EnumDynamicTimeZoneInformation function, 787, 791
EnumerateFonts project, 820–822
enumeration types, 24
EPUB, 856
erasing, 1014, 1023–1029
enabling, 1024
ErrorCode property, 232
ErrorText property, 926–927
error trapping in asynchronous methods, 230
Esc key
button triggered by, 224
processing, 629–630
Euler angles, 976–981
Euler, Leonhard, 976
event handlers
attaching to events, 72
in code-behind file, 71
naming, 71
private keyword, 71
properties and methods of event, 70, 72
sharing, 72–73, 142, 167, 177
source of event, 70, 72
strings, differentiating, 178
for Thumb controls, 300–301
event handling

event handling, 69–96
   AddHandler method, 79
   for animations, 89–96
   Handled property, 78
   Handled property override, 78–79
   for orientation changes, 87
   routed event handling, 72–78
   routed input handling, 74–75
   for size changes, 83–87
   structuring, 633
   timer events, 89–96
events, 69, 83–84
   application suspension, 246–247
   hiding consumer from provider with, 580
   processing, 221
   routing up visual tree, 78
   source of, 74–75
   Tapped event, 69–72
   virtual methods for, 75
EventTrigger class, 360
exception handling
   in file pickers, 240–241
   operation cancellation, 253
   in XamlCruncher, 316–318
exceptions, while saving files, 319
ExecuteAddCharacter method, 218
ExecuteCalculate method, 215
Execute method, 213, 215
   Action<object> delegate, 214
   add argument, 212
ExpandingText project, 91–92
ExpansionBehavior property, 664
Expansion property, 660
ExponentialEase function, 351

F

F5 key, 8
FastNotFluid project, 371–372
Figures property, 53
FileInformation class, 235
file I/O, 233–235
   file pickers and, 235–240
   streamlining, 243–245
   in XamlCruncher, 318–321
FileIO class, 243–245
ReadLinesAsync method, 243
ReadTextAsync method, 243
WriteTextAsync method, 319
FileIO methods, 292–293
FileOpenPicker class, 234–240
invoking, 237
objects, creating, 706–707
file pickers, 235–240
   async calls, consolidating, 241–243
   Cancel button, 238
   exception handling, 240–241
   MIME type, indicating, 886
   Open buttons, 237–238
   permissions for file access, 704
   Snapped state, 736
files
   automatically saving, 249
   canceling opening, 318
   canceling save option, 318
   loading, 238
   reading, 244–245, 254–260
   saving, 239, 318
   storing, 308
   uploading and downloading, 244
FileSavePicker class, 234–240
   objects, creating, 710
fileStream objects, 692–693
   reading, 694
FileTypeFilter collection, 237
FillBehavior property
   HoldEnd setting, 334
   Stop setting, 334
Fill property, 101
   animating, 371–372
   Binding markup extension on, 452
   of Glyphs, 851
FindAllAsync method, 774
FingerPaint projects, 619–646
   About box, 894–897
   AppSettings class, 733–734
   blank canvas, 735
   Color and Thickness buttons, 743–744
   ColorSettingDialog, 757
   CreateTaperedLineGeometry, 642–644
   Dictionary definition, 619
   editing features, 630–633
   Esc-key processing, 629–630
   existing files, drawing on, 735
   Grid name, 619
   HSL structure, 747–749
   input outside page, 622
   ItemTemplate, 746
   line taper, smoothing, 637–646
   MainPage/File.cs file, 736–742
   MainPage(Pointer.cs file, 731–732
   MainPage/xaml.cs file, 742–744
   MainPage/xaml file, 735
   multiple-finger polylines, 621
   OnMenuDelete method, 633
ForeverColorAnimation project

FingerPaint projects (continued)
OnPointerReleased and OnPointerCaptureLost overrides, 644–646
Pictures library access, 736
pointer capturing, 634–635
PointerInfo structure, 634–635
PointerPressed event handler, 631
popup menu logic, 631
pressure sensitivity, 633–637
printing, 948–951
RenderOnBitmap method, 729–731
RightTapped event handler, 631
saving, 722–747
Share charm support, 903
snap mode, 621–622
ThicknessSettingDialog, 743–746
Vector2 structure, 638–646
Visual State Manager markup, 736
fingers
consolidating input with Manipulation events, 616, 655
contact area bounding box, 633
multiple, drawing with, 839–843
multiple polylines from, 621
multiple, tracking, 669, 749–752
single-finger rotation, 676–682
tracking movement, 615, 618, 652–653
FitToCurve property, 1017–1018
Fitzgerald, F. Scott, 867
FixedPage class, 853
FlickAndBounce project, 665–667
flip panels, 434–437
FlipViewColors project, 500–502
FlipView controls, 464, 500–502
for calendar pages, 928, 930
Height and Width properties, 501
for pages, 587, 870–877, 1038, 1043, 1048–1049
SelectionChanged handler, 876
SizeChanged handler, 873–876
FlipViewItem class, 535
Focus method, 140, 630
FolderInformation class, 235
FolderPicker class, 234–235
font families, 845–846
font files in, 846
name, 849
obtaining, 813–821
FontFamily class, 845
FontFamily property, 24, 845, 847–848
ms-appx prefix, 848
font metrics, 824–831
text, positioning precisely, 856
font-related properties, 41, 845
setting in code, 26–27
setting in XAML, 7–8
FontRenderingEmSize property, 851
fonts
ascent line, 828
baseline, 828
boldface and simulated boldface, 849
caps height line, 828
defined, 845
descenders area, 828
design height, 15
enumerating, 820–822, 847
fixed-pitch, 315
in local storage, 852–856
permission to distribute, 847
pixel height values, 827
pixels and points, equivalence, 15
points, 15
private fonts, 847–850
referencing in local storage, 855–856
size, 845
size at printing, 910
style simulations, 851
terminology, 845
text-enhancing attached properties, 856–858
typographical enhancements, 856–858
URIs of, 848–849, 851
Windows font files, 846
Fonts directory, 855–856
FontSize property, 15
animating, 330–333
as dependency property, 15
inheritance of, 21
FontSizeProperty property, 15
FontStretch enumeration, 814
FontStyle enumeration, 814
FontStyle property, 849, 883
defining, 26
using directive, 23
FontUri property, 851
setting in code, 852
FontWeight property, 849
FontWeights class, 814
FontWeight structure, 814
foreach blocks, user interface interaction in, 259
Foreground property, 24, 31, 189
color names, 16–17
of Control class, 162
default value, 16
setting in code, 26–27
setting in XAML, 7–8
ForeverColorAnimation project, 360–361
FormatEffect enumeration

FormatEffect enumeration, 883
FormatRtf flag, 886
FormattedStringConverter class, 154
FormatText method, 166
formatting
   converting from decimal to hexadecimal, 155
   paragraph, 858–861, 872–875
text, 19, 882–883
Forward buttons, 562–563
fractionRead values, 876
Frame class, 29, 557
   back stack, maintaining, 563
   CanGoBack and CanGoForward properties, 557, 558, 560
   Content property, 557
   GoBack and GoForward methods, 557
   stack of visited pages, 557
Frame property, 557
FrameworkElementAutomationPeer class, 184
FrameworkElementAutomationPeer constructor, 890
   overriding, 890
FrameworkElement class, 14
   ActualWidth and ActualHeight properties, 85–86
classes deriving from, 139
   IsHitTestVisible property, 615
   layout properties, 139
   Loaded events, 86
   Name attribute, 23
   Resources property, 43
   Tag property, 167
   Visibility property, 615
   Width and Height properties, 86
FrameworkTemplate class, classes deriving from, 449
French, A. P., 970
FriendlyName property, 710
FromArgb method, 24
FullName property, 595
Func delegates, 214
   Func<BitmapSource> parameter, 949
   Func<object, bool> delegate, 214
future objects, 223

G

Garden of Earthly Delights, The, 993
GeneralTransform class
   TransformBounds method, 427
   TransformPoint method, 427
generated files, 25
generic.xaml file, 520–521
ListBoxItem style, 536
in Themes folder, 530

Geolocator class, 671–672, 999–1012
   starting sensor, 1005
Geometry class, 53, 421–422
   Transform property, 421
going, defining, 56
going transforms, 421–422
GestureRecognizer, 616
get accessor, 170
GetAllX method, 725–727
GetBitmapStream method, 900, 903
GetContent method, 899
GetCurrentOrientation method, 954
GetCurrentPoint method, 618, 1016
GetCurrentReading method, 958
GetDecoderInformationEnumerator method, 707
GetDefault method
   of Accelerometer class, 958
   of SimpleOrientationSensor, 954
GetEncoderInformationEnumerator method, 710
GetFamilyNames method, 815
GetFileAsync method, 248
GetFilesAsync method, 767
GetFirstMatchingFont method, 814
GetForCurrentView method, 904
GetGeopositionAsync call, 672
GetImageryMetadataAsync method, 1002
GetIntermediatePoints method, 618, 1016
GetKeyState method, 575
GetLongitudeAndLatitude method, 1007
GetMapUriAsync method, 1002
GetMetrics method, 819
GetNativeSystemInfo method, 781–782
GetNavigationState method, 568–569
GetPageDescription method, 911
GetPatternCore override, 890
GetPixelDataAsync method, 709
GetPositionFromIndex method, 307
GetPreviewPage handler, 910, 920
GetRenderingSegments method, 1019
GetResults method, 224
   on Completed operations, 232
GetSystemFontCollection method, 810–812
GetTemplateChild method, 521
GetThumbnailAsync method, 767
GetTimeZoneInformationForYear function, 788
GettingCharacterInput project, 887–888
GetTypeInfo method, 114
GetWordFrequenciesAsync method, 255–260
   return value, 255
GIF files. See also images
   frames, extracting, 708
   GlobalOffset properties, 434
   animating, 436
GroupName property

Grid panel, 7
  accessing from code, 28
  accessing from code-behind file, 23
  animations in, 340
  arrays, defining, 278–280
  aspect ratio changes, responding to, 295
  asterisk (or star) values, 146, 158
  Auto value, 147
  background, 289
  Background property, 31, 82
  in Border, 146
  buttons in, 163–167
  centering, 163
  children, 146, 149
  children, arrangement of, 97
  Children property, 24, 37
  contents size, adjusting to fit, 794
  docking content, 158
  Grid.Row and Grid.Column properties, 147, 149, 153
  Grid.RowSpan and Grid.ColumnSpan properties, 147, 150
  interaction with user, 146–151
  MaxWidth setting for printing, 940
  naming, 23
  nesting, 147, 152
  orientation changes, 9
  Path elements in, 59–60
  RowDefinition and ColumnDefinition objects, 146
  RowDefinitions and ColumnDefinitions collections, 146, 152
  rows and columns, 7
  rows and columns, defining, 149, 158
  rows and columns, sizing, 146, 147
  rows and columns, specifying, 146
  single-cell, 7
  SizeChanged handler, 190–192, 303
  spinning, 837–843
  styles, defining, 148
  TextBlock children, 19
  Transparent Background property, 82
  VerticalAlignment setting for printing, 940
  in Viewbox, 401–402
  visual objects, places for, 9

GridView controls, 582–588, 596–608
  grouping items, 608–612
  ItemsSource property, 612
  ObservableCollection type, 608–611
  with view models, 539
  GridViewItem class, 535
  GroupBySex project, 609–611
  GroupedItemsPage.xaml file, 583
  grouping items, 608–612
  GroupName property, 161
Handled property

Handled property, 78, 618
  overriding, 78–79
HarderCameraCapture project, 774–777
hardware coordinate systems, 960–961
HasOverflowContent property, 862, 869
HasThreadAccess property, 225
Hawthorne, Nathaniel, 863
Header property, 325
HeadingMagneticNorth property, 974–976
HeadingTrueNorth property, 974–976
heap allocations, 94
Height property, 86
HelloAudio project, 22
HelloCode project, 23–27
  constructor of MainPage class, 24
HelloImageCode project, 27–28
HelloImage project, 9–13
  image source, 9
HelloLocalImageCode project, 28–29
HelloLocalImage project, 12–13
HelloPrinter project, 905–911
Hello project, 3–9
  App.xaml and App.xaml.cs files, 16
  MainPage.xaml.cs file, 4–5
  MainPage.xaml file, 6, 14
  TextBlock class, 7–8
HelloVectorGraphicsPath project, 56–58
HelloVectorGraphics project, 54–55
HelloVideo program, 22
Heuer, Tim, 823, 953
hexadecimal
  displaying values in, 148–151
  double values, converting to, 155–157
Highlight color, 47
hit testing, 650
  elements, location and orientation, 427
  positioning of elements and, 383
HoldEnd enumeration, 334
Holding events, 69, 615, 632–633
homogenous coordinates, 414
HorizontalAlignment property, 8, 11, 97–101
  of Border element, 99
  of Canvas, 137
  Center setting, 108
  cropping shapes, 103
  default Stretch value, 98
  default value, 81
  of Ellipse, 102
HorizontalAlignment property, 140, 511
  horizontal coordinate, 987–999
  HorizontalCoordinate values, 997
HorizontalListBox project, 482–484
HorizontalScrollBarVisibility property, 113
HowToAsync1 project, 222, 226–227
HowToAsync2 project, 228
HowToAsync3 program, 229–231
  ShowAsync method call, 232
HowToCancelAsync program, 232–233
HRESULT values, 811
HSIColorSelector control, 752–754
HSL (Hue, Saturation, Lightness) color selection, 747–758
HSL structure, 747–749
HttpClient class, 244
Hungarian notation, 780
HyperlinkButton controls, 161, 895

IAsyncAction interface, 225
  AsTask method, 243
  Completed handler, 226
IAsyncInfo interface, 225
  Cancel method, 223, 231–232
  Close method, 223
  ErrorCode property, 223, 232
  Id property, 223
  interface hierarchy, 226
  Status property, 223, 231
IAsyncOperation interface, 223, 232
  objects, obtaining, 229
  Status property, 224
IAsyncOperation<T> interface, 223
IBuffer objects, 244, 686
ICommand interface, 212
  command validity, 212
ID2D1BitmapRenderTarget interface, 835–836
idCustomRangeEdit controls, 945–946
DelegateCommand interface, 214
Identity property, 416
Dictionary<TKey, TValue> interface, 475
IDisposable interface, 238
IDWriteFactory interface, 810
IEnumerable interface, 464, 474
IEnumerable<T> interface, 474
IFormattable interface, 110
IGeometrySegment interface, 724–725
Ignorable attribute, 7
IgnorePressure property, 1017–1018
IReadOnlyCollection<T> interface, 474
IL (Intermediate Language), 822
IList interface, 464
IList<T> interface, 475
implicit styles, 64–65, 459, 530. See also styles
inhibiting, 68
implicit typing, 25
Inclinometer class, 976–980
instantiating, 978, 983
starting sensor, 1005
Indeterminate events, 161
Indices property, 852
inertia, 187, 661, 663–667
acceleration, 664
deceleration, 664–665
stopping, 663
velocity, 663
InertiaTranslationBehavior class, 664
inheritance of properties, 21, 63–64
InitializeComponent method, 5
code, placing after, 25
ink, 1013
continuous strokes, 1018
copying, 1037
loading, 1041–1043, 1050
maintaining, 1014–1017
saving, 1049
InkAndErase project, 1024–1029
Bézier rendering code, 1025–1026
OnPointerMoved override, 1027–1028
OnPointerPressed override, 1026–1027
OnPointerReleased call, 1028
PointerCaptureLost handler, 1028–1029
InkDrawingAttributes class, 1014, 1017–1024
default values, 1017
properties, 1017
InkEraseSelect project, 1030–1038
application bar buttons, 1036–1037
Brush definition, 1030–1031
Copy logic, 1037
Grid elements, 1030
ink, pasting, 1037–1038
OnPointerMoved override, 1032
OnPointerPressed method, 1031–1032
OnPointerReleased override, 1033–1034
InkFileManager class, 1038–1039
hard-coded default values, 1055
LoadAsync and SaveAsync methods, 1041–1043
RenderTarget property, 1040
InkFileManager property, 1044–1047
InkManager class, 1014–1017
CopySelectedToClipboard property, 1029
default properties, setting, 1018
DeleteSelected method, 1029
erasing mode, setting, 1026–1027
ink, copying to clipboard, 1037
InkDrawingAttributes objects, 1014
InkManager class (continued)

InkManager class (continued)
InkStroke objects, 1014, 1018–1019
methods, 1018
Mode property, 1024
MoveSelected property, 1029
multiple pointers, tracking, 1016
overhead, 1017
pen input, collecting, 1015, 1016
pen thickness and color, saving, 1041
ProcessPointerDown property, 1016
ProcessPointerUpdate property, 1016
ProcessPointerUp property, 1016
saving contents, 1041–1043
selection mode, 1029–1038
SelectWithLine method, 1029
SelectWithPolyLine method, 1029
InkManipulationMode enumeration, 1024
InkStroke objects, 1014, 1018
Selected property, 1029
InkStrokeRenderingSegment objects, 1014–1015, 1019
Inline class, 41
InlineCollection type, 41
Inlines property, 41, 859
InlineUIContainer class, 43
RichTextBlock and, 861
InMemoryRandomAccessStream class, 694
INotifyCollectionChanged interface, 475
INotifyPropertyChanged interface, 196, 308, 456, 475,
589–591
definition, 195
implementing, 196–198, 202–204
input devices, 615. See also keyboard input; mouse; pens
input focus, 140, 184
changing, 144
data updates and, 208–209
InputScope property, 185
InputString property, 215
instances, accessing, 171
integerLatitude, 1007–1008
integerLongitude, 1007
IntelliSense, 8, 27
enumeration member options, 183
event handler name suggestions, 70
events suggestions, 70
properties suggestions, 70
interfaces, 474–475
InternationalHelloWorld program, 20–22
Internet Explorer application bars, 269
InvalidateArrange method, 489
InvalidateMeasure method, 489
InvalidatePreview method, 927
Inverse property, 427
IObservableVector interface, 464
IOrderedEnumerable type, 255
IPrintDocumentSource interface, 906
IProcess<T> type, 253
IPrintProgress type, 255
IRandomAccessStream objects, 238, 683
passing to SetSource method, 694
IsBarrelButtonPressed property, 1014
IsChecked property, 161
IsEnabledChanged events, 140
IsEnabled property, 140
IsEraser property, 1014
IsHitTestVisible property, 615
IsHoldingEnabled property, 632–633
IsIdentity property, 416
IsImageModified property, 735
IsInContact property, 618, 627, 650, 1016–1017
IsInertial property, 665
IsInRange property, 618, 627
IsInverted property, 1014
IsLargeArc property, 642
IsLightDismissEnabled property, 267, 897
IsModified property, 307, 308
isolated storage, 234
IsPressed property, 650–652
property-changed handler, 650–652
IsReadOnly property, 185
IsSticky property, 270
IsTextSelectionEnabled, 862
IsThreeState property, 161
ISurfaceImageSourceNative interface, 809
ItemClick events, 599, 602, 605–606
ItemCollection class, 464
Add method, 468
ItemContainerStyle property, 536
Style on, 536
ItemContainerTransitions property, animating, 376
items
displaying, 600
grouping, 582, 608–612
ItemsControl class, 141, 464
class hierarchy, 463
ItemContainerStyle property, 536
ItemsPanel property, 481
ItemsSource property, 472–473
in ScrollViewer, 465–466
templates for, 535–538
items controls, 464
bar charts, 497–499
changes to collection, recognizing, 475
items, adding, 468–469
items, displaying, 466–467
objects, adding, 464
items controls (continued)
  panels, specifying, 481
  SelectorItem derivatives and, 535
  String items in, 465
  tap or click interfaces, 476
ItemsPanel property, 481
ItemsPanelTemplate class, 449, 481–484, 500–502
ItemsPresenter element, 535
ItemsSource property
  binding to collections, 473
  setting, 471
  setting in code, 472–473
ItemTemplate property, 479
  DataTemplate, setting to, 466–467
ITextCharacterFormat interface, 877, 883
ITextDocument interface, 877
  Selection property, 877
ITextParagraphFormat interface, 877
  Alignment property, 884
ITextProvider interface, 184, 890
ITextRange interface, 877
IUICommand interface, 222–223
IUICommand objects, 222
  obtaining with await operator, 229
IValueConverter interface, 110, 155
IValueProvider interface, 184, 890
IVector<T> interface, 475

J
JiggleButton class, 386–387
JiggleButtonDemo program, 386–387
Jobs, Steve, 1013

K
Kernighan, Brian, 3
keyboard accelerators, 572–575
keyboard focus, 140
keyboard input, 184–187, 630, 887–892
  button appearance and, 513
  touch keyboard, 184
keyboard interface
  of context menus, 264
  SmallChange property, 145
Key class, 650–655
Key events, 69
key frame animations, 367–371
  Discrete item, 368
KeyModifiers property, 618
KeypadWithViewModel project, 214–219
KeyTime property, 369
KeyUp and KeyDown events, 184, 887
KeyUp events, 887
key values collections, 474

L
lambda functions
  for Action arguments, 250
  async declarations, 251, 256
  for callback methods, 228, 254
  nesting, 228
  in printing logic, 912, 922
LanguageFontGroup class, 315
language interoperability, 129
laptop coordinate system, 961
LargeChange property, 145
LastKeyVisible property, 654
layout
  aspect ratio, adjusting to, 152–154
  change events, 86
  changes, and animation, 331
  Grid for, 146
  invalidating, 489
  margins in panels, 490
  orientation changes, adjusting to, 152–154
  Panel child classes, 7
  process of, 937
  templates aware of, 539
LayoutAwarePage class, 581
LayoutKind enumeration, 781
layout properties, 139
layout system
  in Canvas, 137–138
  child-driven, 97, 98
  dynamic nature, 97
  parent-driven, 97, 98
LayoutTransform property, 382–383
LayoutUpdated events, 86
Left and Right arrow keys as accelerators, 575
libraries, 127–129
  Class1.cs files, 127
  Class Library template, 129
  implicit styles in, 530
  name, 127
  rebuilding, 130
  referencing, 128, 130, 820
  sharing, 127, 130–131
  user controls, adding, 127
LinearGradientBrush class, 31–32
  animating, 370–371
  element size and, 52
  EndPoint property, 33
LinearGradientBrush class (continued)

GradientStops property, 33, 36, 39
sharing, 44–47
StartPoint property, 33
in XAML, 36
LinearPointKeyFrame class, 368, 369
LineBreak element, 43
LineCapsAndJoins project, 178–181
LineCapsAndJoinsWithCustomClass project, 182–183
Line element, 101, 305
antialiasing, 758
erasing, 1024
stroke thickness based on pressure, 636
visible discontinuities, 637
Y1 and Y2 properties, 826–827
LineHeight property, 858
lines
algorithms for WriteableBitmap, 722–747
bitmaps, drawing on, 724–747
dashed lines, 345
dotted lines, 513–514
drawing on SurfaceImageSource, 831–843
rendered, 722–723
rendering, 101
slope-intercept equation, 723
spiral lines, 647
StrokeStartLineCap, StrokeEndLineCap, and
StrokeLineJoin properties, 177
stroke thickness, 633–637
tapers, smoothing, 637–646
LineSegment class
closing figures with, 642
Point type properties, 361
LineSegment structure, 725
LineStackingStrategy property, 858
ListBox controls, 464
background, 480
horizontal, 482–483
ItemsPanel property, 481
ItemTemplate property, 479
multiple selection, 480
ScrollViewer in, 477–478, 483–484
UniformGrid in, 495–497
virtualization of, 481
width, 477
ListBoxItem class, 535
ListBoxItem style, 536
ListBoxWithItemTemplate project, 479–480, 482–483
debugging code, 481
ListBoxWithUniformGrid project, 495–497
List controls, event IDs in, 652
lists, displaying items in, 791
List<T> collections, 474–475
ListViewBase class, 582
selection support, 599
ListView controls, 582–588, 600–608
grouping items, 608–612
ObservableCollection type, 608–611
with view models, 539
ListViewItem class, 535
LoadAsync method, 238
LoadBitmapAsync method overloads, 770–772
Loaded events, 86
animations, triggering in, 339, 341, 359
of MainPage, 247–248
saving unsaved data during, 248
Loaded handler, 315
anonymous method, defining as, 231
asynchronous methods, calling in, 231
button creation in, 716
controls, initializing, 180
properties, setting, 181
rotate transforms in, 384
Storyboard, starting, 349
translation tags, setting, 394–395
LoadFileAsync method, 241–242
LoadFileFromOpenPicker method, 320
LoadFromStream method, 882, 886
Load method, 293
LoadState method, 581
local namespace declaration, 327
LocalOffset properties, 434
local prefix, 7, 174, 198
local settings, precedence of, 63
local storage
application local storage, 234. See also application local storage
font files in, 852–856
locating, 240
saving unsaved data in, 245–248
local time, 787–788
obtaining, 802
UTC time, converting, 792
Location capabilities, 672
Location property, bindings on, 805
LoggerControl control, 622–626
logical DPI, 540
LogicalDpiChanged handler, 540–541
Log method, 622–625
LookAtAppBarButtonStyles program, 273–276
LookAtFontMetrics program, 824–831
LostFocus events, 70
LPVOID, 781
Matrix property

MadTeaParty project, 859–861
Main method, 25, 30
MainPage class, 4, 5
  Content property, 28, 557
data-sharing code, 898–899
  InitializeComponent method, 5
  navigating to, 559–560
  partial keyword, 5
  single instances of, 557, 561
MainPage.g.cs and MainPage.g.i.cs files, 25
  Connect method, 72
MainPage.xaml.cs files, 4–5
  namespace definitions, 5
  using directives, 5
MainPage.xaml files, 4, 6
ManipulableContentControl control, 763
ManipulationCompleted events, 656, 663
ManipulationDelta events, 655, 656
  handling, 661, 669
  overriding, 660
ManipulationDelta properties, 660
ManipulationDeltaRoutedEventArgs argument, 662–663
  Cumulative property, 663
  ManipulationDelta structure, 660
  edge-of-screen detection, 665
  Expansion property, 660
  Scale property, 660
  Translation property, 660
Manipulation events, 69, 187, 615, 655–663
  centers of scaling and rotation, 661, 673
  Container property, 677
  cumulative manipulation, 663
  horizontal movement, 662
  inertia, 663–667
  inhibiting, 672
  lag time, 616
  multiple fingers, consolidating, 616
  Pivot property, 677
  sequence of, 655–656
  Velocities property, 663
  vertical movement, 662
ManipulationInertiaStarting events, 655–656
  deceleration calculation, 667
ManipulationInertiaStartingRoutedEventArgs class, 664
ManipulationManager class, 675–676, 763–764
  creating objects, 765
ManipulationModeCheckBox control, 657–659
ManipulationMode property, 656, 659
  All setting, 662, 665
  non-default values for, 661
  setting, 662, 676
ManipulationModes enumeration, 656
  TranslateRailsX and TranslateRailsY, 662
ManipulationStarted events, 655, 656
  handling, 669
  movement required for, 672
ManipulationStarting events, 655
  Container property, 677
  manipulations, initializing, 676–677
ManipulationStartingRoutedEventArgs objects, 678
ManipulationTracker program, 656–660
manual animations, 91, 94
ManualBrushAnimation project, 92–93
maps. See also Bing Maps
  rotating with orientation of device, 1000–1012
Margin property, 100–101, 121, 858
  of ButtonBase class, 160
  in custom panels, 490
  spacing between paragraphs, 859
  TemplateBinding on, 510
margins, 100–101
  printable and unprintable areas, 911–914
  for printable pages, 922
markup, 31
  null, specifying, 68
  property settings in, 34–37
markup extensions, 44, 67
  Binding, 66–68
  RelativeSource, 68
  StaticResource, 44, 46, 61
  TemplateBinding, 68
  x:Null, 68
MarshalAs attribute, 791
Math class
  Atan2 method, 724
  Ceiling method, 921
  Cos static method, 50
  Sin static method, 50
Matrix3DHelper class, 430
Matrix3DProjection class, 379, 380
  ProjectionMatrix property, 439
  transform formulas, 440
Matrix3D structure, 430, 437–447
  fields of, 438
  inverting, 986, 989–990
  mapping to video display, 439
  multiplication operator, 447
  numbers, specifying, 439
  Z values, retaining, 439
matrix multiplication, 413–416
  order of multiplication, 415
  in TransformGroup class, 418
Matrix property, 427
Matrix structure

Matrix structure, 416
fields of, 437
identity matrix, 416
Identity property, 416
IsIdentity property, 416
OffsetX and OffsetY properties, 416
Transform method, 418
for two-dimensional affine transform, 437
MatrixTransform class, 379
matrix transforms, 377. See also transforms
MatrixTransform structure, 416–417
maxPageHeight value, 920
MaxWidth property, 121
Measure method, 305, 485, 490, 869
MeasureOverride method, 485–486, 489–490
availableSize argument, 486
validity checks, 489
Width and Height, testing for infinity, 486
MediaCapture class
CapturePhotoToStorageFileAsync method, 775
CapturePhotoToStreamAsync method, 775
MediaCaptureInitializationSettings objects, 775
MediaCapture objects, 775
MediaElement class, 22
memory allocations for objects, 366
menus
context menus, 261–264. See also context menus in previous versions of Windows, 261
MergedDictionaries collection, 47
MergedDictionaries property, 273
message boxes, 221
cancelling, 231–233
MessageDialog class, 221–227
CancelCommandIndex property, 224
cancelling, 232–233
DefaultCommandIndex property, 224
displaying, 223
invoking, 226–227
ShowAsync method, 221
method calls, await operator in, 230
methods
asynchronous, 221–222. See also asynchronous methods
renaming, 71
Microsoft Expression Blend, 7, 449
Microsoft PixelSense, 837, 843
Microsoft Prism framework, 213
Microsoft Surface, 635, 953, 1013
AxisAngleRotation on, 985
compass direction detection, 976
pointer input, 1015
remote deployment of applications on, 823
SensorRotationMatrix, 981
Microsoft Systems Journal, 83, 148
Microsoft Word, ink capabilities, 1037
MinimumReportInterval property, 959
MinWidth and MaxWidth properties, 147
Möbius, August, 414
Model layer, 193–194
Mode property, 676, 1024
OneTime setting, 199
OneWay setting, 199–200
TwoWay setting, 199
monthly calendars, 928–936
MonthYearSelect control, 930, 932
mouse
buttons, navigating with, 572–575
button states, obtaining, 573
input, distinguishing, 650
Manipulation events from, 656
PointerEntered events, 650
PointerMoved events, 617
Position property, 669
text selection, 862
mouse wheel, 617
movement, smoothing, 970
MoveSelected property, 1029
ms-appdata prefix, 244, 249
MVVM (Model-View-ViewModel) pattern, 193–194
buttons and, 212–213
calling hierarchy, 194
command interface, 212
events, 194
Model layer, 193
for small programs, 193, 200
View, 193
View Model, 193. See also View Model

N

NaiveBorderedText project, 98–99
Name attribute, 23, 69
NamedColor class, 469–470, 807
constructor, 474
IEnumerable interface, 474
namespace declarations, 6–7
local prefix, 7, 198
"x" prefix, 7
namespaces
discovering, 5
System.*, 779
System.* namespaces, 5
using directives, 5
Windows.* namespaces, 5
native code, compiling programs in, 822
OnPointerPressed method

- NativeOrientation property, 554, 957
- NativeUp program, 555–556
- Navigate method
calling, 570–571
in OnLaunched method, 557
Page Type argument, 557
NavigatingCancelEventArgs, 564
- navigation accelerators, 572–575
back stack, 562–563
cancelling, 564
events, 564–568
forward and back buttons, 558–560
with mouse buttons, 572–575
new page instances, creation of, 561
of pages, 557–562
page shared data, 575–581
state, saving, 568–571
Visual Studio templates, 581–588
NavigationCacheMode property, 561–562
Disabled setting, 564, 566
Enabled setting, 564, 580–581
Required setting, 564
NavigationEventArgs, 564
NavigationMode property, 564
.NET Framework
APIS, access to, 779
asynchronous processing support, 242–243
Dictionary objects, 255
serialization, 592
Stream objects, 244, 686
StreamReader objects, 255
Task-based Asynchronous Pattern, 222
New (file) operations, 318
NewToggle control, 530–535
class definition, 530
DefaultStyleKey property, 530
dependency properties, 531–532
NewToggleDemo project, 534
Newtonian Mechanics (French), 970
Newton, Isaac, 747, 958
NextBytes method, 71
NonAffineStretch project, 442–447
non-affine transforms, 440–447
notifications of data updates, 194–195

O

ObjectAnimationUsingKeyFrames class, 371
commenting out, 283
Object class animations, 331, 371–372

OnPointerPressed method

- objects, 35
defining as fields, 170
items, controls, adding to, 464
memory allocations for, 366
moving in circles, 358–359
releasing manually, 835
re-using or caching vs. re-creating, 366
setting to properties, 34–37
sharing, 43–47
sharing through data binding, 66–68
transforming, 377. See also transforms
ObservableCollection type, 590–591, 608–611
Octave.xaml file, 652–653
Offset property, 94–95
OffsetX and OffsetY properties, 416
OLED (organic light-emitting diode) technology, 16
OnApplyTemplate override, 521, 532
OnCharButtonClick handler, 167
OnColorChanged handler, 757
OnColorChanged method, 171
OnCreateAutomationPeer override, 184, 890
OnDragThumbDelta method, 300
OnGotFocus and OnLostFocus virtual method, 516
OnGotFocus and OnLostFocus virtual methods, 140
OnGotoButtonClick method, 559
OnInterpolationFactorChanged method, 999
OnKeyDown method, 184, 307, 630, 887
OnKeyUp method, 184, 887
OnLaunched method, 557
Navigate call, 570–571
SetNavigationState call, 570
OnLaunched override, 29–30
OnLoaded method, 315
OnManipulationDelta method, 660
OnManipulationStarting override, 677–678
OnMenuDelete method, 633
OnMessageDialogShowAsyncCompleted method, 225
On methods, 75–76, 615
OnNavigatedFrom method, 5, 561, 564
OnNavigatedTo method, 5, 84, 561, 564
overriding, 558, 566
OnNavigatingFrom method, 5, 561, 564
overriding, 568
OnPageTapped handler, 79
OnPointerCaptureLost method, 630
overriding, 644–646
OnPointerMoved method, 620, 733, 759
overriding, 621, 1027–1028, 1032
processing, 1016
OnPointerPressed method, 733
keyboard focus, 630
overriding, 620, 1016, 1026–1027, 1031–1032
OnPointerReleased method

OnPointerReleased method
  overriding, 621, 644–646, 1033–1034
  processing, 620
OnPrintTaskSourceRequested method
  handler, 908–909
  overriding, 942
OnPropertyChanged method, 196, 198
OnSaveAsAppBarButtonClick method, 710
OnSuspending method, 569
OnTapped method, 140
  overriding, 76–78, 133
OnTextBlockTapped method, 78
OnThumbDragStarted method, 300
Opacity property, 17
  animating, 345–347
Open buttons, 320–321
open (file) operations, 318
OpenIfExists enumeration, 248
OpenIfExists method, 248
OpenReadAsync method, 238
OperationCanceledException exceptions, 253
OppositelyScaledText project, 397–398
OptionChanged events, 924–925
OptionChanged handler, 925–926, 946
OptionId property, 925
OrderByDescending function, 255
Organize Usings command, 24
OrientableColorScroll project, 152–154
OrientationAndOrientation project, 955–958
OrientationChanged events, 87, 554–556, 954
OrientationChanged handler, 954
orientation, hardware
  rotating maps with, 1000–1012
  SimpleOrientationSensor, 953–958
  Windows compensation for, 955, 957, 966
Orientation property, 297, 311, 954
  Horizontal setting, 106, 496–497
  of Slider controls, 143
  of UniformGrid class, 487, 493
  Vertical setting, 130–131
OrientationSensor class, 980–986
  instantiating, 983
  rotation matrix, 981
orientation, software, 9, 955
  application awareness, 554–557
  auto-rotation, preventing, 958
  of book pages, 870–871
  detecting, 958–969
  event handling, 87
  layout, adjusting, 152–154
  maintaining, 554
  native, 554
  portrait mode, 550
preferences for, 957, 966
preferences, requesting, 556
saving settings, 308
Slider control templates and, 521
of StackPanel, 553
text reformatting for changes, 19
OriginalSource property, 74–75, 618
Oscillations property, 337
OuterColor property, 698
OverflowContentTarget property, 862–863
OverlappedStackedText project, 19–20
override keyword, 76

P

Package.appxmanifest file, 235
  application permissions, 894
  Location capabilities, 672, 999
  Webcam, 772
Padding property, 100–101, 911
  TemplateBinding on, 510
Page class, 5
  abandoning instances, 561
  adding to project, 557
  attributes set to, 14
  Content property, 37, 97
  event handlers, attaching and detaching, 561
  Frame property, 557
  Loaded events, 86
  multiple derivatives, 539
  NavigationCacheMode property, 561–562
  new instances, 561
  OnNavigatedFrom method, 561, 564
  OnNavigatedTo method, 561
  OnNavigatingFrom method, 561
  resources, obtaining and releasing, 561
  state, saving and restoring, 564
  TopAppBar and BottomAppBar properties, 268
PageHeaderTextStyle, 583
pageHeight value, 920
PageMapping property, 940–942
page-navigation structure, 5
PageNumber property, 910
pages, 539–612
  ActualWidth and ActualHeight properties, 541
  back stack, 562–563
  back stack, position in, 563
  Border objects, 911
  data, passing and returning, 575–581
  dictionary, sharing, 565
  events, sequence of, 86
  FlipView controls for, 587, 1038, 1043, 1048–1049
PathSegment class

items, displaying in, 481
library projects, 127–129
nesting, 104, 147
orientation changes and, 97
overlapping elements, preventing, 137
page size changes and, 97
Rectangle class in, 101–103
scrolling, 112–118
StackPanel class, 103–106. See also StackPanel
panel
VariableSizedWrapGrid panel, 130–131
virtualizing, 481–484
ParagraphAlignment enumeration, 884
Paragraph class, 41, 859–861, 873–875
ParagraphFormat property, 877
parallel processing, 222, 243
Parameter property, 564
Parent property, 47
parents
and children, balancing needs, 97
layout relationship, 937
partial keyword, 5
PassData class, 577–578
PasswordBox controls, 184, 187
Paste buttons, 898–899
Ctrl+V support, 902
logic, 900–901
Paste command, 898
PasteFromClipboard method, 1037–1038
Path element, 53–57, 101
antialiasing, 758
brushes, applying, 424
in code, 56–57
Data property, 53
drawing smooth lines with, 640–646
in Grid, 59–60
as ink stroke rendering elements, 1019–1020, 1022
Path=, 453
Stretch property, 55
tick marks, 403–404
transforms, applying, 421–422
PathFigure objects, 53, 54
PathGeometry class, 53
constructing, 642
PathIO class, 243–245
string URIs, passing, 244
path markup syntax
in analog clock program, 403
geometry, defining, 55–57
PathMarkupSyntaxCode project, 56–57
Path property, 68
PathSegment class, 53
Point type properties, 361
Polyline element

pixel bits (continued)
  premultiplied-alpha format, 684
  saving as field, 701
  updating, 688
PixelBuffer property, 683, 686
PixelHeight and PixelWidth properties, 684
pixels
  device-independent, 545
  and points, equivalence, 15
pixels per inch, 539
pixelStream objects, 694
Placement enumeration, 264
PlacementTarget property, 499
PlaneProjection class, 379, 430–434
  CenterOfRotationX, CenterOfRotationY,
  CenterOfRotationZ properties, 434
  GlobalOffset properties, 434
  LocalOffset properties, 434
PointAnimationUsingKeyFrames class, 368, 370–371
  KeyFrames property, 369
PointCollection, 49
PointerCanceled events, 617
PointerCaptureLost events, 617
  responding to, 628–629
PointerCaptureLost handler, 1028
PointerCaptures method, 628
Pointer class, 618
PointerDeviceType property, 618
PointerEntered events, 616, 617, 650–655
PointerEventArgs class, 617
Pointer events, 69, 615, 616–619
  dictionary for, 618–619. See also dictionary
  for finger touches, 616–617
  ID number, 615
  logging sequence of, 622–628
  missing data, 628
  for pen input, 617
  tracking fingers with, 615–616
  for XYSlider control, 749–752
PointerExited events, 616, 617, 650–655
PointerId property, 618
PointerInfo class, 622
PointerInfo structure, 634–635
pointer input, 615. See also mouse; pen input; touch
  all, supporting, 1016
  Bézier curves, 1018
  on elements, 627
  receiving, 615
  tracking, 628
PointerLog program, 622–628
PointerMoved events, 616, 617
  Angle value, 649
  distinguishing between mouse and pen input, 627
  finger outside page, 622
  frequency, 637
  LastPoint value, 649
PointerMoved method, RenderOnBitmap calls, 762
PointerPoint class, 618
PointerPoint objects, 1016
PointerPointProperties class, 619, 633–634, 1014
  IsBarrelButtonPressed property, 1014
  IsEraser property, 1014
  IsInverted property, 1014
PointerPressed events, 572–573, 616, 617
  CapturePointer calls, 628
  topmost element, association with, 631
PointerPressed handler, 573, 631
Pointer property, 618
PointerReleased events, 616, 617
  missing, 628
PointerRoutedEventArgs, 617–618, 1016
  GetCurrentPoint method, 618
  GetIntermediatePoints method, 618
  Handled property, 618
  KeyModifiers property, 618
  members, 618
  OriginalSource property, 618
  Pointer property, 618
pointers. See also mouse; pens; touch
  capture by elements, 615, 622–630, 634–635
  capturing, 1023, 1027
  to COM objects, 810
dictionary information, 1024
to DirectX objects, 835
position, calculating, 669
position, returning, 618
PointerWheelChanged events, 617
PointKeyFrame class, 369
Point property
  animating, 331, 361–363
  of XYSlider, 667
points, coordinate, 960
  specifying, 135
Points property, 49–50
Point structure, 49
points, typeface, equivalence to pixels, 15
Point type, 33
PolyBezierSegment, 1022
Polygon class, 101, 188
Polyline element, 48–49, 101
  antialiasing, 758
  Bézier curves, converting to, 1021
  creating, 630
dictionary, adding to, 630
in finger-painting projects, 619–646
initializing, 630
Polyline element (continued)

Polyline element (continued)
Points property, 49–50
for selection, 1031–1032
Stretch property, 51
StrokeThickness property, 634
whirligig functionality, 647–649
Popup class, 261, 265–268, 286–293
with application bars, 285–286, 289–290
as child of UserControl, 268
Closed event handler, 327
displaying, 267
displaying popup, 267
HorizontalOffset and VerticalOffset properties, 267
IsLightDismissEnabled property, 267, 897
Opened and Closed events, 292
positioning, 267
for Settings list items, 894
UIElement Child property, 265
PopupMenu class, 261–264, 632
with application bars, 285–286, 289–290
constructing in code, 262
Opened and Closed events, 292

popups
canvas menus, 261–264. See also context menus
dismissing, 267, 292, 895, 897
Popup class, 265–268
PopupMenu class, 261–264
positioning, 291
PortraitBackButtonStyle, 586
position
calculating, 972
two-dimensional, 970
Position property, 618, 669, 1019
scaling and rotation centers, determining, 673
Posterizer program, 714–721
application bar, 714–716
toggle bar, 714–716
control panel, 714–716
Image element, 714–716
pixel arrays, 714
RadioButton event handler, 719–720
Potter, Beatrix, 119, 937
PowerEase class, 357
Power property, 357
PreconfiguredAnimations project, 373–376
presentation layer, 193
Pressure property, 634
of InkStrokeRenderingSegment, 1019
stroke thickness based on, 636
pressure, touch, 1014–1015, 1019
ignoring, 1022
PreviewPageCountType enumeration, 948
PrimitivePad program, 235–240
Click handler for Open button, 237–238
file-saving logic, 238–239, 244
text wrapping option, 239–240
primitive types, 24, 44
PrintableClassHierarchy program, 915–921
PrintableTomKitten program, 942–947
PrintCustomItemListOptionDetails, 944
PrintDocument class, 904, 906
deriving classes from, 948
event handlers, 906
InvalidatePreview method, 927
PrintDocument method, 909
printers
application registration, 904
asynchronous jobs, 948
big jobs, 948
bitmaps, 948–951
Border objects for pages, 911
custom properties, 911, 922–927
enabling, 932–933
FingerPaint projects, 948–951
font size, 910
margins, 911–914
monthly calendars, 928–936
number of pages, 909
page orientation, 950
page preview, 909–910, 920
page ranges, 937–947
pagination process, 915–921
printable and unprintable areas, 911–914
StackPanel on pages, 915
text color, 910
PrintManager, 904
obtaining current instance, 904
PrintMonthlyPlanner program, 928–936
PrintOrientation enumeration, 923
PrintPageDescription structure, 911
ImageableRect property, 911
PrintPrintableArea program, 912–914
PrintTask class, 904
saving as local variable, 922
QuinticEase class

in solutions, 3
promises (future objects), 223
Propeller project, 395–396
properties
attached properties, 132–136
dependency properties, 15. See also dependency properties
handlers, 223
linking, 66
propagating in visual tree, 14
recursive changes in, 200–201
setting, 15
specifying independently of class, 62
static get-only, 15
property attributes, 35
PropertyChangedEventArgs class, 195
PropertyChanged events, 195–198
ignoring while setting properties, 211
property-changed handlers, 173, 195, 209–210, 487–489
avoiding, 175–176
calls to instance handler, 297–299
property-changed methods, 171
property elements, 35
content before or after, 40–41
Tag property, 178
property-element syntax, 34–37
quotation marks, 67
start and end tags, 35
with Value property, 62
property inheritance, 21, 63–64
property initialization, 33
C# 3.0 style, 24–25
properties, accessing, 170
PropertyMetadata constructor, 168–169
PropertyName property, 195
Property property, 171, 173
protected constructors, 117
pseudo-3D transforms, 380. See also projection transforms
public constructors, 832
public methods, 835

Q
quadkey numbering, 1006–1009
QuadraticBezierSegment class, 361
QuadraticEase class, 357
QuarticEase class, 357
quaternions, 981
QuickNotes project, 249
QuinticEase class, 357
RadialGradientBrushDemo project

RadialGradientBrushDemo project, 696–703
dependency properties, 696–698
MainPage.xaml file, 702
RefreshBitmap method, 699–701
RadialGradientBrushSimulator class, 696–702
animating, 702
instantiating, 702
radians variable, 50
RadioButton controls, 161, 177–183
in Border, 161
Checked handlers, 265, 274
Content property, 160
creating, 716
customizing, 182–183
GroupName property, 161
managing, 325–327
properties, setting, 180–181
styling, 285
Tag property, 576
in UserControl, 289
RainbowAnimation project, 369–370
RainbowEightTransform project, 423–427
RaiseCanExecuteChanged method, 214
RandomAccessStreamReference, 244
creating, 900
Random objects, NextBytes method, 71
RangeBase controls, 141
RangeBaseValueChangedEventArgs, 143
ranges, Slider controls for, 142–145
raster lines, 723
ReadBufferAsync method, 244
ReadingChanged handler, 959
ReadLinesAsync method, 243
Read methods, 238
ReadTextAsync method, 243
ReadTextAsync method calls, 248
RecalculateValue method, 669
Rectangle class, 101–103
properties of, 103
RectangleGeometry transforms, 421–422
redraw methods, 303
RedrawRuler, 305
reentrancy, 319
ref keyword, 810–812
RefreshBitmap method, 698–701
RefreshDisplay method, 1010–1011
RegisterAttached method, 168
Register method, 168
RelativeSource bindings, 506
syntax, 480
RelativeSource markup extension, 68, 204
RelativeTransform property, 422
ReleasePointerCapture method, 628
ReleasePointerCaptures events, 629–630
ReleasePointerCaptures method, 628
Remove method, 567
Remove Unused Usings command, 24
RenderAll method, 1033
RenderBeziers method, 1034–1036
Rendering events, 329, 802
handling, 841
for long-running jobs, 250
Rendering handler, 92–93, 95–96
RenderOnBitmap method, 729–731, 759–762
RenderStroke method, 1034–1036
RenderTarget property, 1040
RenderTransformOrigin property, 377, 384, 387, 396, 400
specifying, 420–421
RenderTransform property, 377, 382, 387
setting to Transform derivative, 378, 379
TransformGroup setting, 673
RepeatBehavior attribute, 335–336, 341
RepeatButton controls, 161
RequestedTheme attribute, 16
resolution
of printers, 904, 911
scaling and, 545
for snap modes, 551
Resolve command, 23
ResourceDictionary class, 43
as root element, 46
resources
binding converters, 111–112
ControlTemplate as, 503
defining, 46
including in projects, 47
predefined, 47
referencing in bindings, 198–199
sharing, 43–48
templates as, 449
View Model as, 198, 206
Resources collection, accessing objects in, 204
Resources dictionary
keys, 46
placement in XAML file, 46
Resources dictionary (continued)
referencing from code, 47
retrieving items, 47
x:Key attribute, 44
Resources property, 43–44, 44, 387
Resources section
animation definitions in, 330
classes in, 198
DataTemplate definition in, 454
MergedDictionaries property, 273
StandardStyles.xaml, 271
Resume method, 433
Resuming events, 246–247
logging, 247–248
RgbaBarChart project, 497–499
RGB (red, green, blue) color, 17
HSL, conversion routines, 747–749, 755–756
RgbViewModel class, 198
RichEditBox controls, 184, 877–886
file loading and saving option, 877
rich text, 845. See also text
Glyphs element, 850–852
paragraph formatting, 858–861, 872–875
private fonts, 847–850
RichTextBox, 877–886
text input, 887–892
text selection, 862
typographical enhancements, 856–858
RichTextBox element
Blocks property, 858
ContextMenuOpening events, 862
HasOverflowContent property, 862
InlineUIContainer and, 861
Measure pass, 862, 869
OverflowContentTarget property, 862–863
pagination and overflow, 862–869
paragraphs, 858–861
repaginating on orientation changes, 871
in ScrollViewer, 859–861
SelectedText property, 862
SelectionChanged events, 862
SelectionEnd property, 862
SelectionStart property, 862
SizeChanged handler, 868
RichTextBoxOverflow element, 862–869
adding and removing as needed, 875
generating in code, 867
HasOverflowContent property, 862
hidden, 866
OverflowContentTarget property, 862
repaginating on orientation changes, 871
RichTextColumn class, 867
RichTextBox project, 878–887
Bold, Italic, and Underline properties, 883
ComboBox handler, 884
file-saving and loading logic, 885–887
FontStyle property, 883
FormatRtf flag, 886
LoadFromStream and SaveToStream methods, 882
paragraph formatting, 882, 884
Suspending handler, 880, 882
text formatting buttons, 880
text-formatting items, initializing, 882–883
right-hand rule, 960
RightTapped events, 69, 262, 615
RightTapped handler, 262, 265–267, 631
Ritchie, Dennis, 3
RotateAroundCenter project, 384–386
rotate gesture, 655
RotateTheText project, 381–384
RotateTransform class, 379
Angle property, 378, 380
CenterX and CenterY handlers, 385
CenterX and CenterY properties, 383–384
RenderTransform property, 382
transform formulas, 415
RotatingMap program, 1000–1012
longitude and latitude calculations, 1007–1008
MainPage constructor, 1002
map tiles, 1000–1002
number of tiles to display, 1009
quadkey construction, 1008–1009
Web service access, 1003–1004
zoom buttons, 1011–1012
rotation, 380–386
2D elements in 3D space, 378, 431
angle of, calculating, 841
Angle property, bindings on, 380–381
around Z axis, 968
centered, 673–676
center of rotation, 378, 383–386, 391–396
centered, 673–676
composite transforms, 391
direction of, 431
of images, 392–393
of images at save, 712–714
MatrixTransform for, 417
origin, 383
RenderTransformOrigin property, 384
single-finger, 676–682
rotation (continued)

rotation (continued)
three-dimensional, 379, 976–986. See also three-dimensional rotation
transform formulas, 411–412
rotation axis, 379–380, 431
RotationBehavior property, 664
RotationCenterDemo project, 394–395
rotation matrices, 981
rotation of device, counteracting, 555
RotationX, RotationY, RotationZ properties, 431
animating, 431–434
RoundCappedLine structure, 727–728
RoundCappedPath structure, 728–730
RoutedEventArgs class, 75
routed event handling, 72–78
OriginalSource property, 75
RoutedEvent property, 360
routed events
Point events, 617
PointerPressed events, 631
RoutedEventArgs projects, 72–83
AddHandler method, 79
event handlers, sharing, 72–73
Grid background color changes, 76–77
HorizontalAlignment and VerticalAlignment, default values, 81–82
no event handlers in XAML file, 75
OnTapped method, 76
OnTextBlockTapped method Handled property, 78
separate processing for events, 80–81
Tapped events on TextBlocks, 77
Tapped handler on parent element, 74
RoutedEventHandler type, 79
routed input handling, 74–75
RowDefinition objects, 146
_MinHeight and MaxHeight properties, 147
RowDefinitions collection, 146
RTF files, 877
loading and saving, 882
RudimentaryTextBox control, 888–892
OnCreateAutomationPeer override, 890
RulerContainer controls, 301
Child property, 301
RunAsync method, 225
Run class, 42
mimicking with TextBlock elements, 108–110
Text property, 41–42
Run method, 250
Run property, bindings on, 87–89

S
Samsung 700T tablet, 1013–1014
SensorRotationMatrix, 981
Save As buttons, 318–319
Save As dialog box, 318
Save buttons, 318–319
SaveState method, 581
SaveStatePage class, 565–568
SaveToStream method, 882, 886
ScalableInternationalHelloWorld program, 87
Scale property, 660
ScaleToBitmap method, 731
ScaleTransform class, 379, 396–400
CenterX and CenterY properties, 398, 411
ScaleX and ScaleY properties, 396, 411
transform formulas, 411, 415
scale transforms, 396–400
center point, 398
composite scaling formulas, 411
negative scaling factors, 398
skew, combining with, 419–421
ScaleX and ScaleY properties, 396
scaling, 545–549, 993
centered, 673–676
isotropic scaling, 660, 674
scaling center, 398, 411
for touch elements, 673–676
scan lines, 723–724
horizontal boundary lines, 725
updating, 731
screen resolution, 539–545
scRGB color space, 17
ScrollBar controls, 112
template for, 142
Value property, 141–142
visibility, 113
scrolling
horizontal, 113, 131, 496, 598
ScrollBar for, 114
vertical, 113, 119, 496
ScrollView, 112–118
Content property, 112
desired height, calculating, 118–119
HorizontalAlignment setting, 123
inertia and bounce, 114
ItemsControl in, 465–466
in ListBox controls, 477–478, 483–484
pinch interface, 112
RichTextBlock in, 859–861
in TextBox controls, 185
sealed keyword, 129
Settings charms

secondary threads, 221, 224
_animations in, 331
_long-running jobs in, 250
_transform animations in, 382
SecondPage class, 557–559, 566
Seek calls, 687
Segments property, 53
SegoeSymbols project, 277–283
_characters, specifying, 282
_DoubleToStringHexByteConverter class, 278
_Grid, 278–280
_ValueChanged handler, 281
_Segoe UI Symbol font, 271–272, 276–283
_emoji characters, 276
_emoticon characters, 277
_folder icon, 767
_mixed symbols, 276
_Style definitions, 272
_transportation and map symbols, 277
SelectedIndex property, 476
SelectedItem objects, 479
SelectedItem property, 476
_as binding source, 477
Selected property, 1029
SelectedText property, 184, 862
SelectedItemPath property, 478–479
SelectedItem property, 477, 479
_selection
_highlighting, 535
_item clicking, enabling, 599
_in ListViewBase, 599
_of pen input, 1029–1038
_selected items, highlighting, 1029
_by swiping, 582
_vs. tapping, 476
_turning off, 599
_SelectionChanged events, 185, 477, 862
_handling, 876
_of RichTextBlock, 862
_SelectionEnd property, 862
_SelectionMode property
_Extended setting, 480
_Multiple setting, 480, 599
_None setting, 599
_Selection property, 877
_SelectionStart and SelectionLength properties, 184–185, 862
_SelectionStates group, 536
_Selector class, 464, 535
_SelectedIndex property, 476
_SelectedItem property, 476
_SelectedValue property, 477
_SelectionChanged events, 477
_SelectedItem class, 535
_SelectWithLine method, 1029
_SelectWithPolyLine method, 1029
_semantic zoom, 582
_sender argument, casting, 72, 73, 151
 SensorQuaternion class, 980–981
 SensorRotationMatrix class, 980–981
_as identity matrix, 981
_sensors, 953–1012
_accelerometer, 958–969
_Compass, 974–976
_Geolocator, 999–1012
_of horizontal coordinate, 987–999
_inclinometer, 976–980
_of magnetic north and true north, 974–976
_members, 959
_OrientationSensor, 980–986
_SimpleOrientationSensor, 953–958
_software interface, 959
_serialization, 592
_set accessor, 26, 170
_functionality, 201
_protected, 198, 200, 311
_public, 202–204
_SetProperty calls, 202
_SetBinding method, 67
_SetBitmap method, 899–900
_SetBubble method, 968
_SetContent method, 899
_SetCrossHair method, 669
_SetDefaultDrawingAttributes method, 1018
_SetDefaultXamlFile method, 315
_SetNavigationState method, 568–569
_SetPixelData method, 712
_SetPreviewPageCount method, 910, 948
_SetPreviewPage method, 910
_SetProperty method, 202
_SetSource method, 683, 692
_IRandomAccessStream objects, passing in, 694–695
_SetTarget property, 339
_Setter class
_for controls, 503
_tags, 61
_Value property, 62
_settings
_precedence of, 63
_storing, 308. See also applications
_SettingsAppBarButtonStyle style, 286
_Settings button, 286
_Settings charms, 894–897
_commands, adding to Settings pane, 896
_PC Settings program, 520
_Permissions item, 894

1093
SettingsDialog class

- SettingsDialog class, 322–327, 804–807
- SettingsPane class, 896
  - Show method, 896
- SetValue method, 26
  - calling, 135
- Shaken events, 966
- Shape class, 48
  - classes deriving from, 101
  - Fill property, 101
  - Stretch property, 51, 58, 102
  - Stroke property, 101
  - StrokeStartLineCap, StrokeEndLineCap, and StrokeLineJoin properties, 177
  - StrokeThickness property, 101
- shapes, filling, 723–724
- Shapes library, 48
- Share charm, 898–902
  - hooking into, 903
  - Share providers, 903
- SharedBrush project, 44–47
- SharedBrushWithBinding project, 66–68
- SharedBrushWithStyle project, 60–64
  - Resources section, 60–61
- SharedStyleWithDataTemplate project, 455–456
- Share Targets, 903
- SharpDX library, 808
- shearing, 406
- ShowAsync method, 221–223, 263, 632
  - Point value, 263
- ShowForSelectionAsync method, 264
- Show method, 896
- ShowPrintUIAsync calls, 929–930
- SilasMarner program, 870–877
  - page fraction, 872
- Simonyi, Charles, 780
- SimpleAnimationCode project
  - Click handler, 339
  - XAML file, 338
- SimpleAnimation project, 330–333
- SimpleColorScroll project, 148–151
- SimpleCompass project, 974–976
- SimpleContextDialog project, 265–268
- SimpleContextMenu project, 262–264
- SimpleEllipse project, 101–102
- SimpleHorizontalStack program, 106–108
- SimpleInking program, 1015–1022
- SimpleKeyFrameAnimation project, 367–369
- SimpleKeypad project, 163–167
- SimpleListBox project, 477–478
- SimpleOrientation enumeration, 954
- SimpleOrientationSensor class, 953–958
  - Accelerometer, correspondence between, 965
  - current orientation, 954
  - DisplayProperties.CurrentOrientation, correspondence between, 957
  - GetDefault method, 954
  - instantiating, 953–954
  - instantiating as field, 956
  - SimplePageNavigation project, 557–562
    - forward and back buttons, 558–559
  - SimpleProjection project, 379–380
  - SimpleRotate project, 378
  - SimpleVerticalStack program, 103–105
  - SineEase function, 358
  - SingleFingerRotate project, 677–678
  - singleton pattern, 474
  - SizeChangedEventHandler, 85
  - SizeChanged events, 84–85
  - SizeChanged handler, 84, 190, 540–541, 742–743, 873–876
    - in code, 85
    - for Grid, 303
  - SkewPlusSkew project, 407–409
  - SkewSlideInText project, 409–410
  - SkewTransform class, 379, 406–409
    - AngleX and AngleY properties, 406–409
    - transform formulas, 412, 415
  - skew transforms, 406–409
    - scaling, combining with, 419–421
    - transform formulas, 412
  - SliderBindings project, 144–145
  - Slider controls, 139, 141–145, 154–157
    - Background and Foreground properties, 145, 754
    - barebones template, 522–524
    - bindings to Value properties, 144–145
    - ControlTemplate, 754
    - default height and width, 143
    - default manifestation, 142
    - default range, 143
    - IsDirectionReversed property, 145
    - LargeChange property, 145
    - manipulating, 143–144
    - Margin properties, 158
    - Minimum and Maximum properties, 143, 158, 278
    - Orientation property, 143
    - sketching with, 157–159
    - SmallChange property, 145
    - spring-loaded slider, 524–527
    - StepFrequency property, 145
    - template for, 521–529
    - thickness, 143
  - Thumb, 523
  - ThumbTipValueChangedConverter property, 155–157
  - TickFrequency and TickPlacement properties, 145
  - tooltips, 154–157
  - ValueChanged event, 140, 143
Slider controls (continued)
  ValueChanged event handler, 142, 151, 156, 281
  Value property, 141–142, 142, 143
  width, 523
SliderEvents project, 142–143
SmallChange property, 145
smart pointers, 810
Smooth Bézier, 345
smoothing
  line taper, 637–646
  movement, 970
SnappedBackButtonStyle, 586
SnappedPageHeaderTextStyle, 583
snap views, 495, 549–554, 604
  adjusting to, 154
  landscape mode, 551
  Snapped state, 583
SolidColorBrush, 31
  animating, 361
  bindings on, 472, 478
  Color property, 157
  definitions in themeresources.xaml, 520
  Highlight color, 47
  in XAML, 34
Solution Explorer
  library projects, adding, 127
  project files, 3
solutions, Visual Studio
  Intermediate Language, compiling in, 822
  libraries, adding, 127, 130–131
  projects in, 3
  referencing, 470–471
  Solution Name, 3
SourcePageType property, 564, 579
Source property, 13, 27, 28, 199, 200, 609–611, 831
Span class
  Inlines property, 42
  shortcut classes, 42–43
SpeedometerProgressBar project, 527–529
SpeedRatio property, 337–338
SpinPaint project, 836–843
Spiral program, 49–51
SplineColorKeyFrame class, 371
SplineDoubleKeyFrame class, 371
SplinePointKeyFrame class, 369, 371
Split App projects, 539, 581, 867
SplitContainer class, 295
  bindings to the Orientation and
  SwapEditAndDisplay properties, 313
Split method, 710
SpreadMethod setting, 425
Springiness property, 337
SpringLoadedSlider project, 524–527
SquaringTheCircle project, 361–363
sRGB (standard RGB) color space, 17
stack, 562
StackPanel panel, 103–106
  background, 105
  border, 266
  building, 123–124
  Buttons in, 265–266
  centering, 123
  children, arrangement of, 97, 105
  children, sizing, 487
  desired height, calculating, 118–119
  displaying items in, 482
  height, 107–108
  HorizontalAlignment and VerticalAlignment properties, 105–106, 108
  horizontal stacking, 106–108
  Image elements in, 119–122
  lists, displaying, 258–260
  MaxWidth property, 121
  nesting, 123–124
  orientation, 553
  Orientation property, 106–107
  RadioButtons in, 266
  scrolling, 112–118
  text wrapping, 119
  vertical stacking, 103–105
  width, 106
StackPanelWithScrolling project, 113–115, 118
  StackPanel in ScrollViewer, 113
StandardPrintTaskOptions class, 923
StandardStyles.xaml file, 46, 60, 271
  application bar button styles, listing, 273–276
  BackButtonStyle, 585
  font, font color, and font size styles, 878
  PageHeaderTextStyle, 583
  PortraitBackButtonStyle, 586
  SettingsAppBarButtonStyle style, 286
  SnappedBackButtonStyle, 586
  SnappedPageHeaderTextStyle, 583
  TextBlock styles, 65
Start Debugging command, 8
StartPoint property, 33
StaticResource markup extension, 44, 46, 61
StepFrequency property, 145
Stop Debugging command, 9
Stop method, 433
storage
  application isolated storage, 249, 308
  application local storage, 234. See also application local storage
  bulk access, 235
StorageFile objects

- **StorageFile objects**, 234, 237
- **ContentType field**, 711
- creating, 249, 736
- **StorageFolder class**, 234
- **GetFilesAsync method**, 767
- **StoreAsync method**, 239

Storyboard class

- animating children of, 331
- **AutoReverse property**, 350
- **Begin method**, 433
- **Children property**, 337
- class hierarchy, 337
- **duration**, 350
- **Pause method**, 433
- **RepeatBehavior property**, 350
- **Resume method**, 433
- reusing in animations, 339
- **SetTarget property**, 339
- **Stop method**, 433
- **TargetName property**, 331, 337
- **TargetProperty property**, 331, 337
- **TargetProperty property**, animating, 347–350
- triggering, 332, 359
- **Triggers section**, defining in, 372
- in visual states, 516

Stream objects, 237

- disposal, 238
- loading and saving to, 877
- saving as field, 701
- writing byte array to, 686–687

StreamReader objects, 255

- **StretchDirection property**, 59
- **StretchedSpiral project**, 51–52
- **Stretch property**, 10–11, 51, 52, 55, 58
  - None setting, 11, 104, 994
  - of Shape class, 102
  - Uniform setting, 10–11, 102, 104–105
  - UniformToFit setting, 11–12, 102
- **Stretch settings**, 98
- **String class**, 812
- **Split method**, 710
- **strings**, special characters, 17–18
- **StrippedDownHello project**, 29–30
- **StrokeDashArray property**, 343, 403
- **StrokeDashCap property**, 343
- **StrokeDashOffset property**, animating, 343–344
- **StrokeEndLineCap property**, 177–183, 636
- **StrokeLineJoin property**, 177–183
- **Stroke property**, 49, 101
- **strokes**
  - rendering, 1033–1036
  - selecting, 1029–1038
- **StrokeStartLineCap property**, 177–183, 636
- **StrokeThickness property**, 49, 101, 634
  - animating, 342–343
  - calculating, 345
- **StructLayoutAttribute class**, 781
- **StudentBody class**, 590–591
  - accessing through bindings, 597
  - **Students property**, 595
  - **StudentBodyPresenter class**, 592–594, 611
  - instantiating, 594
  - **StudentBody property**, 594
- **Student class**
  - **FullName property**, 595
  - **PhotoFilename property**, 595
  - **StudentGroup class**, 609
  - **StudentGroups class**, 609–611
  - **Student objects**, displaying, 596
  - **Students property**, 595
  - **Style class**, 60
    - **BasedOn property**, 63
    - for controls, 141
    - **ControlTemplate as**, 506–507
    - for custom controls, 530
    - default, 520
    - on **ItemContainerStyle property**, 536
    - for ListBox items, 536
    - **Setter tags**, 61
    - **TargetType attribute**, 61
    - **Template property**, defining, 503
    - templates defined in, 455
    - templates, referencing, 500
    - **x:Key attribute**, 61, 64
  - **styles**, 60–65
    - **AppBarButtonStyle style**, 271–276
    - **App.xaml**, definitions in, 937
    - brushes, defining with, 62
    - defining in code, 62
    - dependency properties, targeting, 63
    - for images, 120–121
    - for text, 120–121
    - implicit styles, 64–65, 68, 459, 530
    - inheritance of property settings, 63–64
    - keys, 64
    - referencing, 61
    - **SettingsAppBarButtonStyle**, 286
    - sharing, 43
    - **TextBlock style(s)**, 65
  - **StyleSimulations enumeration**, 851
  - **style**, 1013. See also pens
- **SurfaceImageSource class**, 808
  - line drawing on, 831–843
  - **SurfaceImageSourceRenderer class**, 831–836, 838
  - surrogates, 277, 281

1096
SuspendingDeferral objects, 248
Suspending events, 246–247, 288
logging, 247–248
Suspending handler, 571–572, 880
SuspendResumeLog program, 247–248
SuspensionManager class, 581
suspension, saving settings on, 742
SwapEditAndDisplay property, 311
swiping, 599
disabling, 599
SystemInfoPInvoke project, 782–785
SYSTEM_INFO structure, 780
System.IO namespace, 233
Path class, 57
System.* namespaces, 5, 779
System.Runtime.InteropServices namespace, 781
System.Runtime
.InteropServices.WindowsRuntime namespace, 686
System.Threading.Tasks namespace, 222
SYSTEMTIME structure, 788
SystemTimeToTimeZone function, 788

T
TabbableTextBox class, 306–307
GetPositionFromIndex method, 307
IsModified property, 307
Tab key
detecting press of, 306
keyboard input focus, 144
navigating with, 140, 268
tables
compass orientation, 974–976
coordinate system, 960
display sizes, 549
geographic location, 999–1012
orientation indicators, correspondence between, 957
SensorRotationMatrix, 981
sensory hardware, 953–1012
yaw, pitch, and roll angles, 976–980
TabSpaces property, 307, 325
Tag property, 167, 177–183
as property element, 178
setting, 182
TapAndShowPoint project, 133–136
TappedEventHandler, 70
TappedEvent property, 79
Tapped events, 69–72, 615
Tapped handler, 72–73, 475–476
TappedRoutedEventArgs class, 74–75
OriginalSource property, 74
tapping
positioning elements at, 133–135
vs. selecting, 476
TAP (Task-based Asynchronous Pattern), 222
TapTextBlock program, 70–72
TapToFlip project, 434–437
Storyboard definitions, 435–436
TargetName property, 331, 337
target properties of animations, 330
releasing, 334
TargetProperty property, 331, 337
animating, 347–350
TargetType property, 61, 65, 503, 504
TaskCanceledException exceptions, 232
Task class, 222, 242
AsAsyncAction method, 243
Run method, 250–251
Run method with await operator, 256
WaitAll method, 1004
Yield property, 259–260
tasks
cancelling, 252
progress, reporting, 253–254
Task<string> type, 242
TemplateBinding markup extension, 68, 506
on Background property, 508
Templated Control projects, 530
TemplatedParent, 480
Template property
of Control class, 139, 502, 503
as property element, 503
templates, 329, 449
binding converters in, 462
button visual states, 513–520
ControlTemplate class, 502–512
default, 520
empty tags in, 515
generic.xaml file, 520–521
for ItemsControl derivatives, 535–538
layout-aware, 539
named parts, 521
selected states, defining, 536
sharing, 454
for Slider controls, 521–529
Style, defining in, 455
Style, referencing in, 500
visuals, dynamically updating, 456
Windows Runtime Component, 808–809
Tenniel, John, 345, 859
text. See also fonts; rich text
animating, 330–333
ASCII character codes, 277
centering, 8–9
text (continued)

text (continued)

text (continued)

character formatting, 880
characters, spacing, 850, 852
columns, 866
compressing to fit, 794
as content, 18–19
cut, copy, and paste interaction, 185
displaying, 7
font metrics, 824–831
font-related properties, 845
font size increases and decreases, 288–289
formatting, 41–43
high-performance display, 809–821
indentation, 121
indenting first line, 859
line breaks, 19, 43
listing, 273–276
orientation changes, reformating for, 19
overlapping, 19–20
page numbers, 871–873
paginating, 862–877
paragraph formatting, 858–861, 872–875
past into Visual Studio, 17
precise layout, 921
rich text, 845
scaling, 397–398, 545
scrolling, 859–861
selecting, 862
shadowing, 829–831
spacing between paragraphs, 859
stacking, 19–20
text-enhancing attached properties, 856–858
tilting, 829–831
translation effects, 388–391
underlining, 278, 285
validating, 210
wrapping, 119, 239–240, 861
TextAlignment property, 18, 82, 858
Center setting, 84
TextBlock content property, 41–43

TextBlock element, 7

ActualWidth and ActualHeight properties, 305–306
aligning, 856
background, 71
bindings on, 472
in Border, 98–101, 165
class derivation, 14
in constructor of MainPage class, 24
documentation, 15
embedding, 861
FontFamily property, 845, 847–848
Foreground property, 84, 31
formatting features, 42–43
height, 58
HorizontalAlignment attribute, 8, 81
with Inlines collection, 42
Inlines property, 41
IsHitTestVisible property, 72
IsTapEnabled property, 72
Margin property, 100–101, 121
Measure method, 305
orientation changes, 9
Padding property, 101, 911
paragraphs in, 119–122
positioning, 306
rendered size, obtaining, 305
Run objects, mimicking, 108–110
TextAlignment property, 82
text as content, 18
text string size, calculation of, 824, 828
TextWrapping property, 119
VerticalAlignment attribute, 8–9, 81
in Viewbox, 494
Visibility property, 72
Width and Height properties, 9
TextBox controls, 184–187
AcceptsReturn property, 185
binding behavior, 208–209
custom menu, 262
InputScope property, 185
IsReadOnly property, 185
multiline, 185–187
page state, saving, 565
read-only, 247
ScrollViewer in, 185
SelectedText property, 184
SelectionChanged events, 185
SelectionStart and SelectionLength properties, 184–185
TextChanged events, 140, 185
Text property, 184–185
TextWrapping property, 185
updating with event handlers, 209–211
TextBoxInputScopes program, 185–187
TextChanged events, 185
TextChanged handler, 209–211, 308
text characters
hexadecimal specification, 17
Unicode specification, 17
TextEffects project, 388–390
TextElement class, 41
TextElement class, 41
TextFormattingAppBar project, 283–286
TextFormatting project, 42–43
TextIndent property, 859
text input, 887–892
Text objects, 307
TextOnCanvas project, 132–133
TextPointer type, 862
Text property, 18, 41–42, 184–185
TextWrapping property, 18, 289–290
Wrap setting, 119, 185
ThemeAnimation animations, 373
themeressources.xaml file, 520
themes, 128
Themes folder, 530
TheTaleOfTomKitten project, 119–122
ThicknessSettingDialog class, 743–746
Thickness type, 100
this prefix, 27
ThreadPool class, 250
threads
for execution, 221, 224
UI thread, 221, 223–225. See also UI thread (user-interface thread)
ThreeDeeSpinningText program, 431–434
three-dimensional graphics, 379, 438–447
coordinate system, 960–961
fade-out, 398–400
light source, 388
perspective, 438
reflection effects, 398
three-dimensional rotation, 980–986
around 3D vector, 981–986
yaw, pitch, and roll angles, 976–980
3D Programming for Windows (Petzold), 981
ThrowIfCancellationRequested method, 252
Thumb controls, 187–192
DragStarted and DragDelta event handlers, 188
DragStarted, DragDelta, and DragCompleted events, 187–188, 190
event handlers for, 300–301
in non-affine transforms, 444–446
in XamlCruncher, 295
Thumbnail property, 769
ThumbToolTipValueConverter property, 155–157
Tick events, 89
TickFrequency and TickPlacement properties, 145
Tick handler, 91
tiles, program, 9
tilt
calculating, 990–992
effect on acceleration, velocity, position, 969–972
TiltAndBounce program, 972–973
TiltAndRoll program, 969–972
TiltedShadow program, 419–421
TiltX and TiltY properties, 1019
time
conversions between time zones, 788
converting between programming platforms, 793
local time, 787–788
local time, obtaining, 802
UTC time, converting to local, 792
time, elapsed, 92
Timeline class
AutoReverse property, 349
Completed event, 338
properties of, 337
RepeatBehavior property, 349
SpeedRatio property, 337–338
TimelineCollection type, 337
timer events, handling, 89–96
TimeZoneClock class, 794–797
TimeZoneInfo class, 789
TIME_ZONE_INFORMATION structure, 787
SYSTEMTIME structure, 788
TimeZoneKeyName field, 792
TimeZoneManager class, 789–791
constructor, 791
time zones
enumerating, 787
registry settings, 788
Windows 8 settings, 788
ToggleButton controls, 161, 239–240
Checked and Unchecked events, 161
indeterminate state, 161
IsChecked property, 161
IsThreeState property, 161
ToggleSwitch controls, 161
Header property, 325
OnContent and OffContent properties, 325
ToolTip controls, 499
PlacementTarget property, 499
tooltips
formatting, 155–157
in Slider controls, 524
TopAppBar property, 268
ToString property, 451
touch
centered scaling and rotation, 673–676
contact area bounding box, 633
editing features, 630–633
finger movement, tracking, 618
FingerPaint projects, 619–646
inertia, 187, 661, 663–667
line tapers, smoothing, 637–646
Manipulation events, 655–663
pointer capture, 622–630, 634–635
Pointer events, 616–619
pointer input, 615. See also pointer input
Position property, 669
pressure sensitivity, 633–637
right-clicking equivalent, 632
touch (continued)

TouchInfo class
- Angle value
- LastPoint value

Touch keyboard, 184
- input from, 888
- invoking, 890
- types, 185

To value
- adjusting on Canvas size, 703
- of DoubleAnimation, 424

TransformBounds method, 427, 430
Transform class
- animating, 347
- classes deriving from, 379

TransformGroup class, 379, 391–396
- Children property, 391
- matrix multiplication, 418
- Matrix value, 427
- Value property, 674

Transform method, 418
TransformPoint method, 427
Transform property, 53
- of Brush class, 422
- of Geometry class, 421

transforms, 377–447
- affine transforms, 379
- analog clock project, 400–405
- brush transforms, 422–427
- cascading, 401
- composite transform formulas, 411
- composite transforms, 418–421
- final location, 409–410
- flip panels, 434–437
- geometry transforms, 421–422
- grouping, 391–396
- homogenous coordinates, 414
- mathematics, 410–418, 437–447
- Matrix3D, 437–447
- matrix multiplication, 413–416, 437–447
- matrix transform formulas, 414, 437–447
- matrix transform, specifying, 416
- non-affine transforms, 440–447
- order of, 418–421
- origin, 400
- on page load, 409–410
- projection transforms, 430–437
- rotation, 380–386
- rotation axis, 378, 379–380
- rotation transform formulas, 411–413, 415
- scale transform formulas, 415
- scale transforms, 396–400
- in secondary threads, 382
- skew, 406–409
- skew transform formulas, 412, 413, 415
- standard matrix representation, 414
- three-dimensional effects, 379
- translation, 388–391
- translation formulas, 413–415
- two-dimensional affine transforms, 440
- as visual feedback, 386–387

TransformToVisual method, 291, 427–430
Transition class, animations deriving from, 375–376
TransitionCollection type, 375
Transitions property, animating, 375
TranslateTransform class, 379, 388–393
- To value of DoubleAnimation, 424
- transform formulas and translation factors, 411, 415
- X and Y properties, 388, 394
- translation, 388–391

TranslationBehavior property, 664
Translation property, 660
transparency, bitmap, 691–696
Triggers property, 359–360
- defining on Path, 361

Triggers section
- animation definition in, 372
- Storyboard definition in, 372
try and catch blocks, 233
- await operator in, 230, 232
- file access in, 258
TryParse method, 211
TryUnsnap method, 554
.TTF (TrueType font) extension, 846
TwelveHourClock class, 460–463
Twist property, 1019
typefaces, 845
- attributes of, 845
TypeInfo objects, 114
Typography class, 856–858

U

UICommand class
- buttons, defining, 222
- context menu commands, 263
UICommandInvokedHandler type, 227
UICommandSeparator objects, 263
velocity

UIElement, 14
  CapturePointer method, 628
  ManipulationMode property, 656
  pointer-input events, 615
  Projection property, 377
  RenderTransformOrigin property, 377, 384
  RenderTransform property, 377
  TransformToVisual method, 427
  user-input events, 69, 140
UIElementCollection class, 24, 38
UI thread (user-interface thread), 224–225
  animation callback methods in, 329
  animations in, 331
  callback methods, running in, 227
  PrintDocument object, creating and accessing, 911
  queuing methods to run, 225
  WriteableBitmap modifications in, 721
UnconventionalAppBar project, 269–271
underlining, 278, 285
UnhandledException event handler, 317
Unicode, 277
surrogates, 277, 281
UnicodeString property, 851
UniformGrid panel, 487
  columns, calculating number, 496
  in ListBox, 495–497
  Orientation property, 487, 493
units
  device-independent, 545
  screen size in, 553
Unloaded events, 86
UpdateBitmap method, 717, 719–721
UpdateImageTransforms method, 997, 999
Update method, 836
UpdateValues method, 366
UserControl class
  classes deriving from, 188
  Content property, 38, 125
  custom controls, deriving from, 175–177
  deriving from, 124–126, 141
  Popup child instances, 268
  with RadioButton controls, 289
user input
  consolidation of touch, mouse, and pen input, 615
  from margin area, 101
  pen input, 616
  pointer input, 615. See also pointer input; touch
  transferring to View Model, 194
user-input events, 69, 80–83
  controls, processing of, 140
  routed input handling, 74–75
  Tapped event, 69–72
virtual methods for, 75
user interface
  controls and elements of, accessing, 224–225
  dependency properties, 15
  threads for, 221, 223–225
user settings, saving, 934
using directives, 5
  organizing, 23–24
  removing unused, 23–24
UTF-8 encoding, 277, 886
UTF-16 encoding, 277, 886
UTF-32 (32-bit Unicode Transformation Format), 277

V

Validate method, 211
ValueChanged events, 143, 667
  handler, 671
ValueChanged handler, 142, 151, 281
Value property, 62
  of DiscreteObjectKeyFrame, 372
  of PointKeyFrame class, 369
  of TransformGroup, 674
VariableSizedWrapGrid panel, 130–131
  children, arrangement of, 97
  Orientation property, 130–131
var keyword, 25
Vector2 structure, 638–646
  Rotate method, 641–642
vector graphics, 48–57
  centering objects, 50
  curves, 53
  element height and width, 49
  filling screen with object, 51
  HorizontalAlignment and VerticalAlignment, 49, 51
  image brushes, 52–53
  negative coordinates, 400
  Path element, 53–57
  PathGeometry element, 53–57
  scaling, 545
  stretching, 58–60
vectors
  acceleration vectors, 966, 968
  accelerometer readings, 959–960
  defined, 959–960
  magnitude, 960, 961, 965
  two-dimensional, 970
Velocities property, 663
velocity
  calculating, 972
  two-dimensional, 970
VirtualAlignment property

VirtualAlignment property, 8–9, 11, 97–101
of Border element, 99
of Canvas, 137
Center setting, 108
cropping shapes, 103
default Stretch value, 98
default value, 81
of Ellipse, 102
VerticalContentAlignment property, 140, 511
VerticalScrollBarVisibility property, 113
video display
  resolution, 15–16
  updating, 91
videos, playing, 22
View, 193–194
Viewbox, 58–60
  Grid in, 401–402
  StretchDirection property, 59
  Stretch property, 58
  stroke width, 60
  TextBlock in, 494
View Model, 193–194
  AddCharacterCommand property, 215–216
  application settings, saving, 308–311
  Button calls into, 212–213
  commands in, 213
  commands, processing, 213–219
  data bindings in, 194–196
  DataContext property, 205
  data entry validation, 210–211
  DeleteCharacterCommand property, 215–216
  DisplayText property, 215
  Execute and CanExecute methods, defining, 213
  ExecuteCalculate and CanExecuteCalculate
  methods, 215
  InputString property, 215
  instantiating in code, 205
  instantiating in Resources collection, 206
  as resource, defining, 198, 218
  updates, blocking, 211
view model classes, 474
view models
  creating, 594
  with data bindings, 794–797
defining, 600–601
instantiating, 612
items, accessing, 594
ObservableCollection type, 608–611
ViewParams class, 1003
virtualization of panels, 481–484
VirtualizingStackPanel, 260, 482, 501, 1050
VirtualKey enumeration, 574
VirtualKey values, 887
virtual methods
  overriding, 76, 461
  for user-input events, 75
virtual protected methods, 615
Visibility property, 72, 461
  animating, 371–372
  bindings on, 462–463
  Collapsed setting, 516, 654
  Visible setting, 615
VisitedPageSave program, 565–568
Visual Basic, .NET API access, 779
visual feedback, transforms for, 386–387
visuals, control, defining, 125–126
VisualStateGroups section, 514–515
Visual State Manager, 513–520
  markup for, 602–603
  Snapped state, responding to, 583
  views, responding to, 583–584
VisualStateManager class
  empty tags, 515
  GoToState method, 513, 516, 606
  VisualStateGroups section, 514–515
visual states, 513–520
  brushes for, 521
  empty tags, 515–516
  SelectionStates group, 536
  triggering, 652
Visual Studio
  application packages, creating, 824
  Blank App, 581
  Configuration Manager, 823
  design view, 8
  generic.xaml file generation, 530
  Get Started tab, 3
  Grid App projects, 539
  IntelliSense feature, 8. See also IntelliSense
  intermediate files, 25
  multiple instances, 130
  navigation classes, 581–588
  New Project dialog box, 3
  proxy classes for Web services, 1002
  sample view model, 581
  Simulator, 9
  Solution Configurations box, 821–822
  Solution Platforms box, 822
  Split App projects, 539
  Suspend, Resume, or Suspend And Shutdown
  commands, 246, 248
text, pasting into, 17
virtual methods, overriding, 76
Visual Studio debugger, 8
application suspension and resumption, 246
data, saving and restoring, 248
exceptions, reporting on, 318
Visual Studio Toolbox, 8
visual tree, 14
AppBar in, 270
Button in, 503
climbing, 47
DataContext property, propagation down, 204
defining, 139–140, 451–452
elements, accessing, 143
events routing up, 78
generating for items, 467, 473
implicit styles, propagation of, 64–65
Loaded events, 86
parent-child hierarchy, 97
precedence of settings, 63
for printing, 904
properties propagating through, 15
visual state elements, 513
VisualTreeHelper class, 47

W
WaitAll method, 1004
Web browsers
Back and Forward buttons, 562
Internet Explorer, 269
Web services, proxy classes for, 1002
WebViewBrush class, 31
WhatRes program, 540–545
WhatSize program, 84–86
WHATSIZE program, 83–84
WhatSizeWithBindingConverter project, 110–112
WhatSizeWithBindings project, 108–110
WhatSnap program, 549–554
WheresMyElement project, 427–430
Whirligig program, 647–649
Width property, 86
Win32 API
accessing, 779. See also P/Invoke (Platform Invoke)
data type equivalence to C#, 780–781
EnumDynamicTimeZoneInformation function, 787
functions, documentation of, 780
Hungarian notation, 780
WinBase.h header file, 780
Window class, 557
Current property, 594
Dispatcher property, 594
windowing coordinate system, 33, 49

Windows Store, uploading packages to
windows
application, 5
size changes, 539
size, obtaining, 541
UI thread, 224–225
Windows 8
accessing, programming language and, 779
documentation, 5
retained mode graphics system, 83
screen resolution adjustments, 540
Windows Runtime Components, 779
Windows 8 simulator, testing display sizes in, 543–544
Windows 8 Sound Recorder, 22
Windows 8 start screen
moving items, 582
selecting items, 582
width and height, 97
zooming items, 582
Windows.ApplicationModel.DataTransfer
namespace, 898
Windows.ApplicationModel.DataTransfer.ShareTarget
namespace, 898
Windows.Devices.Geolocation namespace, 953
Windows.Devices.Sensors namespace, 953
/Windows/Fonts directory, 846
Windows.Graphics.Display namespace, 554
Windows.Graphics.Imaging namespace, 703
Windows.Graphics.Printing namespace, 904
namespace, 904
Windows header files, 780
Windows.* namespaces, 5
Windows operating system
font files, 846
returning programmatic control to, 230
Windows Paint, 12
Windows Runtime
class hierarchy, 13
public classes, structures, and enumerations, 6–7
Windows Runtime Components, 129, 779
limitations on, 808
Windows.Storage namespaces, 233
Windows.Storage.BulkAccess namespace, 235
Windows.Storage.Pickers namespace, 234
Windows.Storage.Streams namespace, 694
Windows Store applications, 779
display resolution for, 549
fonts, 847
minimum screen size, 551
Windows Runtime Component template, 808–809
writing, 786–808
Windows Store, uploading packages to, 824
Windows.UI.Input.Inking namespace

- Windows.UI.Input.Inking namespace, 1013, 1014
- Windows.UI.Xaml.Controls namespace, 140
- Windows.UI.Xaml.Controls.Primitives namespace, 140
  - using directive, 333
- windows.ui.xaml.media.dxinterop.h header file, 809
- Windows.UI.Xaml.Media.Media3D namespace, 430
- Windows.UI.Xaml namespaces, 5
- Windows.UI.Xaml.Printing namespace, 904
- Windows.UI.Xaml.Shapes namespace, 57
- winnt.h header file, 780
- WordFreq project, 254–260
  - cancellation and progress reporting, 257–258
  - error reporting, 256–257
  - Start and Cancel buttons, 256–257
- WORD values, 780
- WPF (Windows Presentation Foundation), 6
- wProcessorArchitecture field, 784–785
  - formatting, 785
- WrapOptionsDialog objects, 290
- WrapedText project, 18–19
- wrapper DLLs, 779. See also DirectXWrapper library
- Windows Runtime Components, 779
- WriteableBitmap class, 646, 683
  - array size, 686
  - Color constructor, 762
  - color format, 684
  - constructor, 686
  - instantiating, 685
  - invalidating, 687
  - line- and arc-drawing algorithms, 722–747
  - PixelBuffer property, 683, 686
  - pixel formats, 709
  - premultiplied alphas, 684, 691, 762
  - SetSource method, 683, 692
  - sRGB color values, 684
  - updating pixels, 717
- WriteAsync method, 686
- WriteBufferAsync method, 244
- WriteFactory class
  - GetSystemFontCollection method, 812
  - header file, 810
- WriteFont class, 818–819
  - header file, 818
- WriteFontCollection class, 811–813
- WriteFontCollection.h header file, 811, 812
- WriteFontFamily class header file, 813–814
- WriteTextAsync method, 248, 249, 319
- wrl.h (Windows Runtime Library) header file, 809

X

XAML

- Angle property, setting, 380
- animations in, 395–363
- conditional execution, 460–463
- experimenting with, 293. See also XamlCruncher
- local namespace declaration, 327
- XamlCruncher, 293–308
  - Add button, 320
  - animations in, 361
  - application bar, 294
  - AppSettings class, 308–311
  - bindings in, 313
  - Button definitions, 313
  - CheckIfOkToTrashFile method, 320
  - code-behind file, 313–315
  - constructor, 313–315
  - dependency property definitions, 301–302
  - editor, 293
  - EditOrientation enumeration, 308
  - EditOrientationRadioButton controls, 322–327
  - exception handling, 316–318
  - file I/O, 318–321
  - file-saving logic, 307–308
  - Grid, 313
  - limitations, 327
  - Loaded handler, 313–315
  - MainPage.xaml, 311–315
  - Open button, 320
  - Orientation property, 297
  - page configuration, 293
  - property-changed handlers, 297–300, 303–305
  - Refresh button, 294
  - ruler and grid lines, 294, 301
  - Save and Save As buttons, 318–319
  - saving documents, 318–319
  - SetDefaultXamlFile method definition, 315
  - Settings dialog, 322–327
  - SplitContainer, 295, 313
  - status bar, 313
  - TabbableTextBox class, 306–307
  - Thumb control, 295
- XAML (eXtensible Application Markup Language), 6
  - attribute names, 133
  - content properties, 37–41
  - data bindings, 66–68
  - elements and controls, coding, 23–27
  - for layout and appearance of page, 69
  - markup extensions, 44, 67
  - MVVM pattern, 193
  - object elements, 35
  - Path Markup Syntax, 55–57
XAML (eXtensible Application Markup Language) 
(continued)
property attributes, 35
property elements, 35
property-element syntax, 34–37
resource sharing, 43–47
stretching graphics, 58–60
styles, 60–65
syntax, 19, 31–68
TextBlock content property, 41–43
vector graphics, 48–57
XAML files
compatibility, 6
count updating logic, 144
elements, instantiating, 38
images, referencing, 12
names in, 23
parsing, 25
resource definitions, 46
resources section, 44
root tags, placing content between, 177
templates in, 467–469
View and View Model interactions, 194
visual elements, defining, 6
visual tree of elements, 14
XAML parser, 24
XamlReader class, 293, 316–317
XamlReader.Load method, 56
XAML resources, 43–47
x:Boolean type, 44
x:Class attribute, 6, 7
x:Double type, 44
x:Int32 type, 44
x:Key attribute, 44, 61, 62
XML (eXtensible Markup Language) character
escaping, 17
XmlSerializer class, 592–594
x:Name attributes, 23, 69
on VisualStateGroup and VisualState tags, 515
x:Null markup extension, 68
"x" prefix, 7, 23, 44, 68
XPS packages, 852–853
XPS (XML Paper Specification), 852–853, 856
x:String type, 44
x:Uid type, 68
XYSlider controls, 667–672
with Pointer events, 749–752
Point property, 667
SetCrossHair method, 669
template, defining, 670–671
ValueChanged event handler, 671
XYSliderDemo project, 670–672
Y
YawDegrees property, 980
YawPitchRoll program, 976–980
YellowPadPage class, 1043
blue rule lines, 1043–1044
InkFileManager property, 1044
YellowPad project, 1038–1056
application bar items, 1038
Bézier rendering logic, 1039–1041
blue rule lines, 1043–1044
buttons, handling, 1050–1052
controls, initializing, 1052–1053
copy, cut, paste, and delete functions, 1053
current page index, 1049–1050
FlipView control, 1038, 1043
LoadAsync and SaveAsync methods, 1038
number of pages, 1049–1050
page numbers, 1048–1049
Paste logic, 1050–1052
pen width and color selection, 1050–1052
Yield property, 259–260
YoungGoodmanBrown project, 863–866
Z
Z axis, 413
rotation around, 431
z equals 1, 414
z-index, incrementing, 190
Zindex property, 136–137, 190, 435
ZoomMode property Disabled setting, 112
zoom, semantic, 582
Z order, 20
This page intentionally left blank