

Adam Nathan



**Full Color**

Figures and code  
appear as they do  
in Visual Studio.

# Universal Windows<sup>®</sup> Apps with XAML and C#

**UNLEASHED**

**SAMS**

**FREE SAMPLE CHAPTER**

SHARE WITH OTHERS



Adam Nathan

# Universal Windows® Apps with XAML and C#

UNLEASHED

**SAMS**

800 East 96th Street, Indianapolis, Indiana 46240 USA

## **Universal Windows® Apps with XAML and C# Unleashed**

**Copyright © 2015 by Pearson Education, Inc.**

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

ISBN-13: 978-0-672-33726-0

ISBN-10: 0-672-33726-6

Library of Congress Control Number: 2014919777

Printed in the United States of America

First Printing February 2015

### **Trademarks**

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

### **Warning and Disclaimer**

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

### **Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

### **EDITOR-IN-CHIEF**

Greg Wiegand

### **ACQUISITIONS EDITOR**

Joan Murray

### **DEVELOPMENT EDITOR**

Mark Renfrow

### **MANAGING EDITOR**

Kristy Hart

### **SENIOR PROJECT**

### **EDITOR**

Betsy Gratner

### **INDEXER**

Lisa Stumpf

### **PROOFREADER**

Kathy Ruiz

### **TECHNICAL EDITOR**

Bill Wagner

### **EDITORIAL ASSISTANT**

Cindy Teeters

### **COVER DESIGNER**

Mark Shirar

### **COMPOSITOR**

Nonie Ratcliff

# Contents at a Glance

Introduction .....	1
Part I Getting Started	
1 Hello, <i>Real World</i> ! .....	7
2 Mastering XAML .....	43
Part II Building an App	
3 Sizing, Positioning, and Transforming Elements.....	63
4 Layout.....	83
5 Handling Input: Touch, Mouse, Pen, and Keyboard.....	117
Part III Working with the App Model	
6 App Lifecycle .....	161
7 Threading, Windows, and Pages.....	181
8 The Many Ways to Earn Money.....	199
Part IV Understanding Controls	
9 Content Controls .....	227
10 Items Controls.....	259
11 Text .....	283
12 Images .....	315
13 Audio, Video, and Speech.....	355
14 Other Controls.....	387

## Part V Leveraging the Richness of XAML

15	Vector Graphics.....	421
16	Animation.....	453
17	Styles, Templates, and Visual States .....	499
18	Data Binding.....	529

## Part VI Exploiting Windows

19	Working with Data.....	555
20	Supporting App Commands .....	583
21	Leveraging Contracts .....	613
22	Reading from Sensors.....	647
23	Controlling Devices .....	663
24	Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen .....	687
	Index.....	723

# Table of Contents

Introduction	1	Part II Building an App	
Who Should Read This Book?	3	3 Sizing, Positioning, and Transforming Elements	63
Software Requirements	3	Controlling Size	64
Code Examples	3	Controlling Position	68
How This Book Is Organized	3	Applying 2D Transforms	72
Conventions Used in This Book	5	Applying 3D Transforms	79
		Summary	82
Part I Getting Started		4 Layout	83
1 Hello, <i>Real</i> World!	7	Discovering Your Window Size and Location	84
Creating, Deploying, and Profiling an App	7	Pane1s	88
Understanding the App Packages	10	Handling Content Overflow	103
Updating XAML and C# Code	21	Summary	115
Making the App World-Ready	29	5 Handling Input: Touch, Mouse, Pen, and Keyboard	117
Making the App Accessible	35	Touch Input	118
Submitting to the Windows Store	40	Mouse Input	141
Summary	42	Pen Input	144
2 Mastering XAML	43	Keyboard Input	153
Elements and Attributes	44	Summary	159
Namespaces	45	Part III Working with the App Model	
Property Elements	47	6 App Lifecycle	161
Type Converters	49	Killing	163
Markup Extensions	49	Suspending	164
Children of Object Elements	52	Resuming	166
Mixing XAML with C#	56	Terminating	167
XAML Keywords	59		
Summary	60		

Launching	168	AppBarToggleButton	243
Activating	171	CheckBox	244
Managing Session State with SuspensionManager	173	RadioButton	245
Programmatically Launching Apps	176	ToolTip	246
Summary	179	App Bars	249
		Summary	257
7 Threading, Windows, and Pages	181	10 Items Controls	259
Understanding the Threading Model for Universal Apps	181	Items in the Control	260
Displaying Multiple Windows	186	Items Panels	262
Navigating Between Pages	189	ComboBox	265
Summary	198	ListBox	267
		ListView	269
8 The Many Ways to Earn Money	199	GridView	273
Adding Advertisements to Your App	200	FlipView	274
Supporting a Free Trial	205	SemanticZoom	276
Supporting In-App Purchases	210	MenuFlyout	279
Validating Windows Store Receipts	218	Summary	281
Testing Windows Store Features	220	11 Text	283
Summary	225	TextBlock	283
		RichTextBlock	296
Part IV Understanding Controls		TextBox	301
9 Content Controls	227	RichEditBox	309
Button	230	PasswordBox	311
AppBarButton	234	Summary	313
HyperlinkButton	241	12 Images	315
RepeatButton	242	The Image Element	316
ToggleButton	243	Multiple Files for Multiple Environments	325
		Decoding Images	330
		Encoding Images	339
		Rendering PDF Content as an Image	347
		Summary	353



21	Leveraging Contracts	613	24	Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen	687
	Account Picture Provider	615		Live Tiles	687
	AutoPlay Content and AutoPlay Device	617		Badges	701
	File Type Associations	620		Secondary Tiles	703
	Protocol	623		Toast Notifications	705
	File Open Picker	624		Setting Up Push Notifications	711
	File Save Picker	627		The Lock Screen	719
	Contact Picker	628		Summary	721
	The Contact Contract	631		Index	723
	The Appointments Provider Contract	635			
	Background Tasks	637			
	Summary	646			
22	Reading from Sensors	647			
	Accelerometer	647			
	Gyrometer	651			
	Inclinometer	651			
	Compass	651			
	Light Sensor	651			
	Orientation	652			
	Location	652			
	Proximity	659			
	Summary	662			
23	Controlling Devices	663			
	Fingerprint Readers	664			
	Image Scanners	664			
	Barcode Scanners	668			
	Magnetic Stripe Readers	671			
	Custom Bluetooth Devices	673			
	Custom Bluetooth Smart Devices	676			
	Custom USB Devices	679			
	Custom HID Devices	682			
	Custom Wi-Fi Direct Devices	684			
	Summary	686			

## About the Author

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

Adam's books are considered required reading by many inside Microsoft and throughout the industry. Adam is the author of *Windows 8.1 Apps with XAML and C# Unleashed* (Sams, 2013), *101 Windows Phone 7 Apps* (Sams, 2011), *WPF 4.5 Unleashed* (Sams, 2013), *.NET and COM: The Complete Interoperability Guide* (Sams, 2002), and several other books. You can find Adam online at [www.adamnathan.net](http://www.adamnathan.net), or @adamnathan on Twitter.

# Dedication

*To Tyler and Ryan.*

# Acknowledgments

I'd like to thank Ashish Shetty, Tim Heuer, Mark Rideout, Jonathan Russ, Joe Duffy, Chris Brumme, Eric Rudder, Neil Rowe, Betsy Gratner, Ginny Munroe, Bill Chiles, Valery Sarkisov, Joan Murray, Patrick Wong, Jacqueline Ting, and Michelle McCarthy. As always, I thank my parents for having the foresight to introduce me to Basic programming on our IBM PCjr when I was in elementary school.

## We Want to Hear from You!

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

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

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

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

Email:                    [feedback@sampublishing.com](mailto:feedback@sampublishing.com)

Mail:                     Joan Murray  
                              Acquisitions Editor  
                              Sams Publishing  
                              800 East 96th Street  
                              Indianapolis, IN 46240 USA

## Reader Services

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

*This page intentionally left blank*

# INTRODUCTION

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

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

With universal Windows apps, it's finally possible to create an app that targets both PCs (desktops, laptops, tablets, and hybrids) and phones simultaneously. Universal apps also represent a consolidation of XAML-based technologies. First there was Windows Presentation Foundation (WPF) for traditional desktop apps, then Silverlight for the Web, then Silverlight for the phone, then the XAML UI Framework for Windows Store apps. All of these frameworks are similar but frustratingly not quite the same. The technology behind universal apps now has enough momentum that the need for these older frameworks should fade away.

## In This Chapter

- Who Should Read This Book?
- Software Requirements
- Code Examples
- How This Book Is Organized
- Conventions Used in This Book

Universal apps run on the Windows Runtime, or WinRT for short. WinRT is actually based on Microsoft's Component Object Model (COM) that has been around since 1993, but most of the time you can't tell. And most of the time, it doesn't matter. This is a modern, friendlier version of COM that is more amenable to *automatic* correct usage from environments such as C#. (Contrast this to over a decade ago, when I wrote a book about mixing COM with .NET. This topic alone required over 1,600 pages!)

WinRT APIs are automatically *projected* into the programming language you use, so they look natural for that language. Projections are more than just exposing the raw APIs, however. Core WinRT data types such as `String`, collection types, and a few others are mapped to appropriate data types for the target environment. For C# or other .NET languages, this means exposing them as `System.String`, `System.Collections.Generic.IList<T>`, and so on.



Although WinRT APIs are not .NET APIs, they have metadata in the standardized format used by .NET. Therefore, you can browse them directly with familiar .NET tools, such as the IL Disassembler (ILDASM). You can find these on your computer as `.winmd` files. Visual Studio's "Object Browser" is also a convenient way to search and browse WinRT APIs.

In the set of APIs exposed by Windows:

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

As you dig into the framework, you notice that the XAML-specific and .NET-specific APIs are indeed the most natural to use from C# and XAML. General-purpose WinRT APIs follow slightly different conventions and can sometimes look a little odd to developers familiar with .NET. For example, they tend to be exception-heavy for situations that normally don't warrant an exception (such as the user cancelling an action). Artifacts like this are caused by the projection mechanism mapping HRESULTs (COM error codes) into .NET exceptions.

I wrote this book with the following goals in mind:

- To provide a solid grounding in the underlying concepts, in a practical and approachable fashion
- To answer the questions most people have when learning how to write universal apps and to show how commonly desired tasks are accomplished
- To be an authoritative source, thanks to input from members of the team who designed, implemented, and tested Windows and Visual Studio
- To be clear about where the technology falls short rather than blindly singing its praises

- To optimize for concise, easy-to-understand code rather than enforcing architectural patterns that can be impractical or increase the number of concepts to understand
- To be an easily navigated reference that you can constantly come back to

To elaborate on the second-to-last point: You won't find examples of patterns such as Model-View-ViewModel (MVVM) in this book. I *am* a fan of applying such patterns to code, but I don't want to distract from the core lessons in each chapter.

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

## Who Should Read This Book?

This book is for software developers who are interested in creating apps for the Windows Store, whether they are for tablets, laptops, desktops, or phones. It does not teach you how to program, nor does it teach the basics of the C# language. However, it is designed to be understandable even for folks who are new to .NET, and does not require previous experience with XAML.

If you are already well versed in XAML, I'm confident that this book still has a lot of helpful information for you. At the very least, this book should be an invaluable reference for your bookshelf.

## Software Requirements

This book targets Windows 8.1, Windows Phone 8.1, and the corresponding developer tools. The tools are a free download at the Windows Dev Center: <http://dev.windows.com>. The download includes the Windows SDK, a version of Visual Studio Express for Windows, and miscellaneous tools.

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

## Code Examples

Source code for examples in this book can be downloaded from [www.informit.com/title/9780672337260](http://www.informit.com/title/9780672337260).

## How This Book Is Organized

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

## Part I: Getting Started

This part includes the following chapters:

- Chapter 1: Hello, *Real World*!
- Chapter 2: Mastering XAML

Part I provides the foundation for the rest of the book. Chapter 1 helps you understand all the tools available at your disposal, and even dives into topics such as accessibility and localization so you can be prepared to get the broadest set of customers possible for your app.

## Part II: Building an App

This part includes the following chapters:

- Chapter 3: Sizing, Positioning, and Transforming Elements
- Chapter 4: Layout
- Chapter 5: Handling Input: Touch, Mouse, Pen, and Keyboard

Part II equips you with the knowledge of how to place things on the screen, how to make them adjust to the wide variety of screen types, and how to interact with the user.

## Part III: Working with the App Model

This part includes the following chapters:

- Chapter 6: App Lifecycle
- Chapter 7: Threading, Windows, and Pages
- Chapter 8: The Many Ways to Earn Money

The app model for universal apps is significantly different from the app model for traditional desktop applications in a number of ways. It's important to understand how the app lifecycle works and how you need to interact with it in order to create a well-behaved app. But there are other pieces to what is sometimes called the *app model*: how one app can launch another, how to work with the Windows Store to enable free trials and in-app purchases, and how to deal with multiple windows and pages.

## Part IV: Understanding Controls

This part includes the following chapters:

- Chapter 9: Content Controls
- Chapter 10: Items Controls
- Chapter 11: Text

- Chapter 12: Images
- Chapter 13: Audio, Video, and Speech
- Chapter 14: Other Controls

Part IV provides a tour of the controls built into the XAML UI Framework. There are many controls that you expect to have available, plus several that you might not expect.

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

This part includes the following chapters:

- Chapter 15: Vector Graphics
- Chapter 16: Animation
- Chapter 17: Styles, Templates, and Visual States
- Chapter 18: Data Binding

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

## **Part VI: Exploiting Windows**

This part includes the following chapters:

- Chapter 19: Working with Data
- Chapter 20: Supporting App Commands
- Chapter 21: Leveraging Contracts
- Chapter 22: Reading from Sensors
- Chapter 23: Controlling Devices
- Chapter 24: Thinking Outside the App: Live Tiles, Notifications, and the Lock Screen

This part of the book covers unique and powerful Windows features that are not specific to XAML or C#, but they are things that all app developers should know.

## **Conventions Used in This Book**

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

Typeface	Meaning
<i>Italic</i>	Italic is used for new terms or phrases when they are initially defined and occasionally for emphasis.
Monospace	Monospace is used for screen messages, code listings, and filenames. In code listings, <i>italic monospace type</i> is used for placeholder text. Code listings are colorized similarly to the way they are colorized in Visual Studio. <code>Blue monospace type</code> is used for XML elements and C# keywords, <code>brown monospace type</code> is used for XML element names and C# strings, <code>green monospace type</code> is used for comments, <code>red monospace type</code> is used for XML attributes, and <code>teal monospace type</code> is used for type names in C#.
<b>Bold</b>	When appropriate, bold is used for code directly related to the main lesson(s) in a chapter.

When a line of code is too long to fit on a line in the printed book, a code-continuation arrow (➡) is used.

Throughout this book, and even in this introduction, you will find a number of sidebar elements:



#### What is a FAQ sidebar?

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

#### Digging Deeper



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



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



#### This is a warning!

A warning alerts you to an action or a condition that can lead to an unexpected or unpredictable result—and then tells you how to avoid it.

# Chapter 7

## THREADING, WINDOWS, AND PAGES

This chapter begins by examining a very important topic, although one that many developers take for granted: the threading model for universal apps. This background is especially helpful for the advanced feature of writing an app that displays multiple windows, which is the second topic in this chapter. The third and final topic—navigating between a window’s pages—is a feature leveraged by just about every real-world app.

### Understanding the Threading Model for Universal Apps

Universal apps have two types of threads that can run your code: UI threads and background threads. (Other types of threads exist, but they are implementation details.) As much as possible, a UI thread should be kept free to process input and update UI elements. Therefore, long-running work should always be scheduled on a background thread.

Typically, an app has a single UI thread, but that’s only because an app typically has a single window. Each window has its own UI thread, so an app with multiple windows (covered in the upcoming “Displaying Multiple Windows” section) has multiple UI threads.

### In This Chapter

- Understanding the Threading Model for Universal Apps
- Displaying Multiple Windows
- Navigating Between Pages

If you have a long-running computation to perform, which therefore isn't appropriate for a UI thread, you don't get to explicitly create a background thread for the task. Instead, you schedule it via a static `RunAsync` method on the `Windows.System.Threading.ThreadPool` class. Windows manages all background threads for you.

There is always a main UI thread, even if the corresponding main window has not yet been shown. For example, if an app is activated via a contract such as the File Picker contract (see Chapter 21, "Leveraging Contracts"), the app typically displays a special file-picking window and never displays its main window. Yet the app has two UI threads running in this scenario, so your code can always count on global state created by the main thread.

UI objects must be created and called on a UI thread. This includes every class deriving from `DependencyObject`, which is most classes in the XAML UI Framework. Outside of the XAML UI Framework, most Windows Runtime objects can be created and used on any thread, and you control their lifetime. This makes them very natural to use in C# without worrying about threading or COM-style apartments. Such objects are called *agile objects*.

### ASTA Threads ...

In documentation and error messages, UI threads are sometimes referred to as *ASTA threads*. ASTA stands for App Single-Threaded Apartment, which is a nod to COM's notion of single-threaded apartments (STA).

ASTA threads are similar to COM's STA threads in that they provide an easy-to-program, single-threaded experience. But they have an enhancement that COM's STA threads do not: they are not reentrant, unless the incoming call is logically connected to the one in progress. In other words, if you make a call from a UI thread to another thread (or process), and that thread needs to call back to the UI thread, the Windows Runtime does a lot of work to track this and allow it. On the other hand, arbitrary code is prevented from calling into the UI thread while it is doing work. This prevents a huge class of bugs that plague traditional desktop apps, and means that UI objects generally don't need locking to protect themselves. The Windows Runtime also prevents UI threads from calling each other directly, as that would be prone to deadlock.

## Awaiting an Asynchronous Operation

Windows Runtime APIs are designed to make it really hard to block a UI thread. Whenever the Windows Runtime exposes a potentially-long-running operation, it does so with an asynchronous method that performs its work on a background thread. You can easily identify such methods by their `Async` suffix. And they are everywhere. For example, showing a `MessageDialog` (discussed in Chapter 14, "Other Controls") requires a call to `ShowAsync`:

```
MessageDialog dialog = new MessageDialog("Title");
IAsyncOperation<IUICommand> operation = dialog.ShowAsync();
// The next line of code runs in parallel with ShowAsync's background work
MoreCode();
```

Asynchronous methods in the Windows Runtime return one of several interfaces such as `IAsyncOperation` or `IAsyncAction`. Asynchronous methods in .NET return a `Task`. These are two different abstractions for the same set of asynchronous patterns. The `System.WindowsRuntimeSystemExtensions` class provides several `AsTask` extension methods for converting one of these interfaces to a `Task`, as well as `AsAsyncOperation` and `AsAsyncAction` extension methods for converting in the opposite direction.

In the preceding code snippet, when `ShowAsync` is called in this manner, the call returns immediately. The next line of code can run in parallel with the work being done by `MessageDialog` on a different thread. When `ShowAsync`'s work is done (because the user dismissed the dialog or clicked one of its buttons), `MessageDialog` communicates what happened with an `IUICommand` instance. To get this result, the preceding code must set operation's `Completed` property to a delegate that gets called when the task has finished. This handler can then call operation's `GetResults` method to retrieve the `IUICommand`.

Of course, such code is pretty cumbersome to write, and the proliferation of asynchronous methods would result in an explosion of such code if it weren't for the C# `await` language feature. When a method returns one of the `IAsyncXXX` interfaces or a `Task`, C# enables you to hide the complexity of waiting for the task's completion. For the `ShowAsync` example, the resulting code can look like the following:

```
async Task ShowDialog()  
{  
    MessageDialog dialog = new MessageDialog("Title");  
    IUICommand command = await dialog.ShowAsync();  
    // The next line of code does not run until ShowAsync is completely done  
    MoreCodeThatCanUseTheCommand(command);  
}
```

When the `ShowAsync` call is made in this manner, the current method's execution stops—*without blocking the current thread*—and then resumes once the task has completed. This enables the code to retrieve the `IUICommand` object as if `ShowAsync` had synchronously returned it, rather than having to retrieve it from an intermediate object in a convoluted fashion. You can only use the `await` keyword in a method that is marked with an `async` keyword. The `async` designation triggers the C# compiler to rewrite the method's implementation as a state machine, which is necessary for providing the handy `await` illusion.

People commonly refer to this pattern as “awaiting a method,” but you're actually awaiting the returned `IAsyncXXX` or `Task` object. As before, the method actually returns promptly. This is clearer if the preceding code is expanded to the following equivalent code:

```
async Task ShowDialog()  
{  
    MessageDialog dialog = new MessageDialog("Title");  
    IAsyncOperation<IUICommand> operation = dialog.ShowAsync();  
    IUICommand command = await operation;
```

```
// The next line of code does not run until the operation is done
MoreCodeThatCanUseTheCommand(command);
}
```

It's also worth noting that the `async` designation does not appear in the metadata for a method when it is compiled. It is purely an implementation detail. Again, you're not awaiting a method; it simply happens to return a data type that supports being awaited.

Notice that the sample `ShowDialog` method returns a `Task`, which seems wrong because the method does not appear to return anything. However, the `async`-triggered rewriting done by the C# compiler does indeed return a `Task` object. This enables an asynchronous operation to be chained from one caller to the next. Because `ShowDialog` returns a `Task`, its caller could choose to await it.

If an `async` method actually returns something in its visible source code, such as the `command` object in the preceding code, then it must return `Task<T>`, where `T` is the type of the object being returned. In this example, it would be `Task<IUICommand>`. The C# compiler enforces that an `async` method must either return `Task`, `Task<T>`, or `void`. This means that `ShowDialog` could be rewritten with `async void` instead of `async Task` and it would still compile. You should avoid this, however, because it breaks the composition of asynchronous tasks.



**Avoid defining an `async` method with a `void` return type!**

If you do this, your callers cannot await or otherwise leverage an operation returned by your method (because it doesn't return anything), which makes it harder for their code to behave correctly. This cannot be avoided, however, on methods that must match a delegate signature, such as a `Button`'s `Click` handler.



**Do not use `Task.Wait`!**

The .NET `Task` object provides many useful abstractions for cancellation and advanced control flow. You can also schedule your own long-running task via `Task.Run`, which directly returns a `Task`, rather than using `ThreadPool.RunAsync`, which returns an `IAsyncAction` instead. (`Task.Run` should really be called `Task.RunAsync`.)

One feature that you should avoid is `Task`'s `Wait` method. Although waiting for a task to complete sounds similar to awaiting the task to complete, the `Wait` method blocks the current thread. Besides defeating the purpose of the background work, for cases such as showing a `MessageDialog`, this causes a deadlock:

```
void ShowDialog()
{
    MessageDialog dialog = new MessageDialog("Title");
    dialog.ShowAsync().AsTask().Wait(); // DEADLOCK!
}
```





You can leverage the nice `await` control flow for APIs that don't return a `Task` or `IAsyncXXX` by wrapping the use of the APIs with an object called `TaskCompletionSource`. This has a `Task` property that you can return to your callers, and methods that you can call at the appropriate time to signal that the `Task` has completed, failed, or been canceled. `TaskCompletionSource` is used later in this chapter to provide a nice way to create and show additional windows.

## Transitioning Between Threads

Occasions often arise when one thread needs to schedule work to be executed on another thread. For example, although events on XAML objects are raised on the same UI thread that created the object, this is usually not the case for non-UI objects in the Windows Runtime. Instead, they are raised on whatever background thread happens to be doing the work.

An example of this can be seen with the events defined by `MediaCapture`, a class described in Chapter 13, "Audio, Video, and Speech." The following code incorrectly tries to update the UI to notify the user about a failure to capture video from the camera:

```
// A handler for MediaCapture's Failed event
void Capture_Failed(MediaCapture sender, MediaCaptureFailedEventArgs e)
{
    // This throws an exception:
    this.textBlock.Text = "Failure capturing video.";
}
```

The exception thrown explains, "The application called an interface that was marshalled for a different thread. (Exception from HRESULT: 0x8001010E (RPC\_E\_WRONG\_THREAD))."

With `DependencyObject`'s `Dispatcher` property of type `CoreDispatcher`, however, you can marshal a call back to the proper UI thread needed to update the `TextBlock`. It can be used as follows:

```
// A handler for MediaCapture's Failed event
async void Capture_Failed(MediaCapture sender, MediaCaptureFailedEventArgs e)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        // This now works, because it's running on the UI thread:
        this.textBlock.Text = "Failure capturing video.";
    });
}
```

Here, an anonymous method is used for `RunAsync`'s second parameter (which must be a parameterless `DispatchedHandler` delegate) to keep the code as concise as possible. The code must be scheduled to run at one of the following priorities, from highest to lowest:

High (which should never be used by app code), Normal, Low, and Idle (which waits until the destination thread is idle with no pending input).

This `CoreDispatcher` mechanism is also how one window can communicate with another window. Each `Window`, along with related Windows Runtime abstractions, expose a `Dispatcher` property that can schedule a delegate to run on its own UI thread.

## Displaying Multiple Windows

Universal apps, even when running on Windows 8.1, are hosted in a window. Not only that, but an app running on a PC can use multiple windows simultaneously. Although they are called *windows* in XAML-specific APIs, windows are often called *views* in Windows Runtime APIs. In Windows Runtime terminology, a view is the union of a window and its UI thread.

Apps show a primary window when activated, but you can create and show any number of secondary windows on a PC. You create a secondary window by calling `CoreApplicationView.CreateNewView`. This returns a `CoreApplicationView` instance representing the new window and its UI thread, but you can't interact with it yet. You must wait for `Application.OnWindowCreated` to be called, which occurs on the new UI thread. On this thread, you can initialize the window much like you would initialize your primary window. Once it is initialized, you can show it with a PC-only `ApplicationViewSwitcher` class—back on the original UI thread.

Because of the convoluted control flow, this is a perfect opportunity to use the `TaskCompletionSource` type mentioned earlier in this chapter. Listing 7.1 adds an await-friendly `CreateWindowAsync` method to `App.xaml.cs`, inspired by the Multiple Views Sample project provided by the Windows SDK. This portion of the code compiles for both PC and phone.

**LISTING 7.1** `App.xaml.cs`: Providing an await-Friendly `CreateWindowAsync` Method

---

```
using System;
using System.Collections.Concurrent;
using System.Threading.Tasks;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.ApplicationModel.Core;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace MultipleWindows
{
    sealed partial class App : Application
    {
        // The pending tasks created by CreateWindowAsync
        ConcurrentQueue<TaskCompletionSource<Window>> taskWrappers
            = new ConcurrentQueue<TaskCompletionSource<Window>>();
    }
}
```

```

// Create a new window.
// This wrapper method enables awaiting.
public Task<Window> CreateWindowAsync()
{
    // Create a Task that the caller can await
    TaskCompletionSource<Window> taskWrapper
        = new TaskCompletionSource<Window>();
    this.taskWrappers.Enqueue(taskWrapper);

    // Create the secondary window, which calls Application.OnWindowCreated
    // on its own UI thread
    CoreApplication.CreateNewView(null, null);

    // Return the Task
    return taskWrapper.Task;
}

protected override void OnWindowCreated(WindowCreatedEventArgs args)
{
    CoreApplicationView view = CoreApplication.GetCurrentView();
    if (!view.IsMain)
    {
        // This is a secondary window, so mark the in-progress Task as complete
        // and "return" the relevant XAML-specific Window object
        TaskCompletionSource<Window> taskWrapper;
        if (!taskWrappers.TryDequeue(out taskWrapper) ||
            !taskWrapper.TrySetResult(args.Window))
            taskWrapper.SetException(new InvalidOperationException());
    }
}

...
}
}

```

The code inside `OnWindowCreated` can easily check whether it is being invoked for the main window or a secondary window by obtaining the current `CoreApplicationView` and examining its `IsMain` property.

Listing 7.2 shows the code-behind for the following `MainPage.xaml` that leverages `CreateWindowAsync` to show a new window every time its `Button` is clicked:

```

<Page x:Class="MultipleWindows.MainPage" ...>
    <Viewbox>
        <Button Click="Button_Click">Show a New Window</Button>
    </Viewbox>
</Page>

```

**LISTING 7.2** MainPage.xaml.cs: Using CreateWindowAsync to Create Then Show a New Window

```
using System;
using Windows.UI.Core;
using Windows.UI.ViewManagement;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace MultipleWindows
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            InitializeComponent();
        }

        async void Button_Click(object sender, RoutedEventArgs e)
        {
            int newWindowId = 0;

            // Create the new window with our handy helper method
            Window newWindow = await (App.Current as App).CreateWindowAsync();

            // Initialize the new window on its UI thread
            await newWindow.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                // In this context, Window.Current is the new window.
                // Navigate its content to a different page.
                Frame frame = new Frame();
                frame.Navigate(typeof(SecondPage));
                Window.Current.Content = frame;

                // Set a different title
                ApplicationView.GetForCurrentView().Title = "NEW";

                newWindowId = ApplicationView.GetApplicationViewIdForWindow(
                    newWindow.CoreWindow);
            });

            // Back on the original UI thread, show the new window alongside this one
            // (PC only)
        }
    }
}
```

```
bool success =  
    await ApplicationViewSwitcher.TryShowAsStandaloneAsync(newWindowId);  
}  
}  
}
```

Once `Button_Click` retrieves the new window instance, which actually came from `App's OnWindowCreated` method, it can schedule its initialization on its UI thread. It awaits this work's completion, because the next step requires a window ID that must be retrieved from that window's UI thread. With the ID, the original code can then call `ApplicationViewSwitcher.TryShowAsStandaloneAsync` to show the new window.

Each new window is a top-level window to be managed by the user, just like the app's main window. `TryShowAsStandaloneAsync` has overloads that enable you to specify a `ViewSizePreference` for the target window or for both windows, just like when launching an app. You can also swap one window with another in-place by calling `SwitchAsync` instead of `TryShowAsStandaloneAsync`.



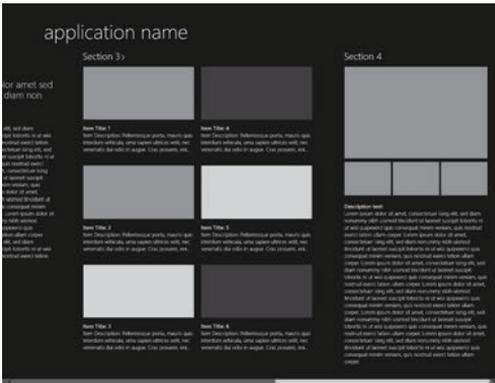
For several Windows Runtime classes, you obtain an instance by calling their static `GetForCurrentView` methods. Such objects are designed to have a single instance per window. In other words, they are not agile. If you have multiple windows, you should make sure to use the appropriate instance from each UI thread.

## Navigating Between Pages

Although simple apps might have only one `Page` per window, most windows in real-world apps leverage multiple `Pages`. The XAML UI Framework contains quite a bit of functionality to make it easy to navigate from one page to another (and back), much like in a Web browser. Visual Studio templates also give you a lot of code in a `Common` folder to handle many small details, such as applying standard keyboard navigation to page navigation, and automatic integration of session state.

Although a Blank App project is given a single page by default, you can add more pages by right-clicking the project in Solution Explorer then selecting **Add, New Item...**, and one of the many `Page` choices. The different choices are mostly distinguished by different preconfigured layouts and controls.

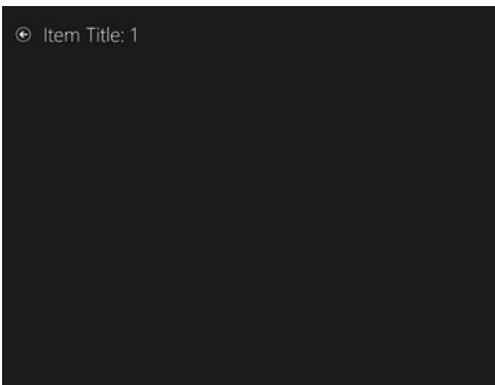
In addition, if you create a Hub App project, it is already set up as an app with a multi-`Page` window. Figures 7.1 and 7.2 show the behavior of the Hub App project before any customizations are made. Separate `Pages` are provided for phone versus PC, which explains a number of differences in the content and style.



HubPage



SectionPage



ItemPage

FIGURE 7.1 A Hub App project, shown here on a PC, contains three pages: one that shows sections, and one that shows the items inside each section, and one that shows item details.

Selecting a certain section on the first Page (HubPage) automatically navigates to its details on the second page (SectionPage). Selecting an item in the section navigates to the third page (ItemPage). When the user clicks the back button in the corner of the window on a PC, or the hardware back button on a phone, the window navigates back to the previous page.

## Basic Navigation and Passing Data

Although it's natural to think of a Page as the root element of a window (especially for single-page windows), all Pages are contained in a Frame. Frame provides several members to enable Page-to-Page navigation. It is often accessed from the Frame property defined on each Page.

To navigate from one page to another, you call Frame's Navigate method with the type (not an instance) of the destination page. An instance of the new page is automatically created and navigated to, complete with a standard Windows animation.

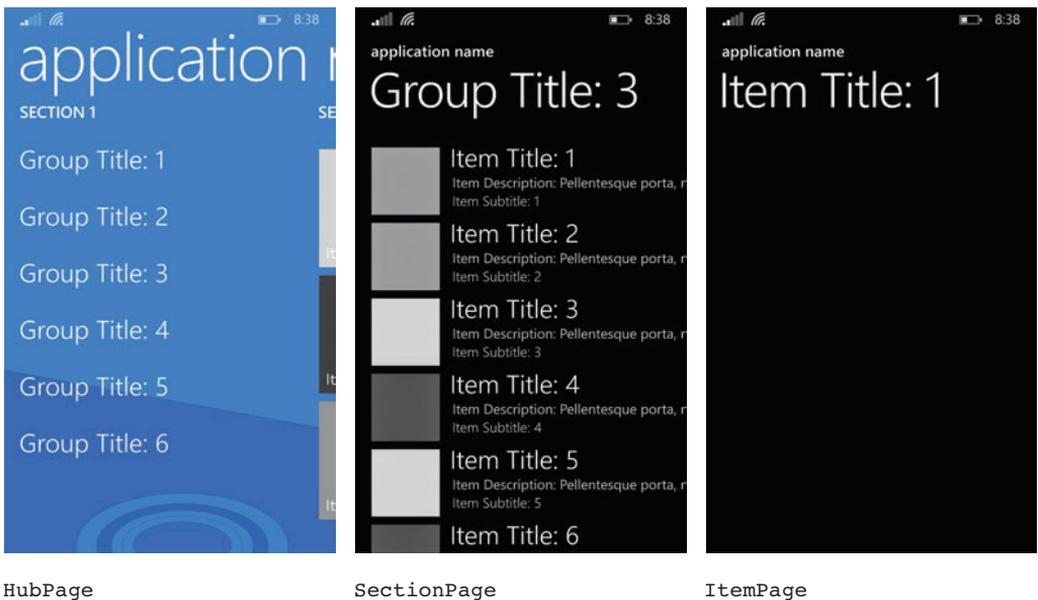


FIGURE 7.2 A Hub App project, shown here on a phone, contains three pages: one that shows sections, and one that shows the items inside each section, and one that shows item details.

For example, when an item is clicked in a Hub App's `SectionPage`, it navigates to a new instance of `ItemPage` as follows:

```
void ItemView_ItemClick(object sender, ItemClickEventArgs e)
{
    // Navigate to the appropriate destination page, configuring the new page
    // by passing required information as a navigation parameter
    var itemId = ((SampleDataItem)e.ClickedItem).UniqueId;
    this.Frame.Navigate(typeof(ItemPage), itemId);
}
```

`Navigate` has two overloads, one that accepts only the type of the destination page, and one that also accepts a custom `System.Object` that gets passed along to the destination page. In this case, this second parameter is used to tell the second page which item was just clicked. If you use `SuspensionManager` in your project, its automatic management of navigation state means that whatever you pass as the custom `Object` for `Navigate` must be serializable.

The target `ItemPage` receives this custom parameter via the `NavigationEventArgs` instance passed to the Page's `OnNavigatedTo` method. It exposes the object with its `Parameter` property.

A call to `Navigate` raises a sequence of events defined on `Frame`. First is `Navigating`, which happens before the navigation begins. It enables the handler to cancel navigation by setting the passed-in `NavigatingCancelEventArgs` instance's `Cancel` property to `true`.

Then, if it isn't canceled, one of three events will be raised: `Navigated` if navigation completes successfully, `NavigationFailed` if it fails, or `NavigationStopped` if `Navigate` is called again before the current navigation finishes.

`Page` has three virtual methods that correspond to some of these events. `OnNavigatingFrom` enables the current page to cancel navigation. `OnNavigatedFrom` and `OnNavigatedTo` correspond to both ends of a successful navigation. If you want to respond to a navigation failure or get details about the error, you must handle the events on `Frame`.

## Navigating Forward and Back

Just like a Web browser, the `Frame` maintains a back stack and a forward stack. In addition to the `Navigate` method, it exposes `GoBack` and `GoForward` methods. Table 7.1 explains the behavior of these three methods and their impact on the back and forward stacks.

TABLE 7.1 Navigation Effects on the Back and Forward Stacks

Action	Result
<code>Navigate</code>	Pushes the current page onto the back stack, empties the forward stack, and navigates to the desired page
<code>GoBack</code>	Pushes the current page onto the forward stack, pops a page off the back stack, and navigates to it
<code>GoForward</code>	Pushes the current page onto the back stack, pops a page off the forward stack, and navigates to it

`GoBack` throws an exception when the back stack is empty (which means you're currently on the window's initial page), and `GoForward` throws an exception when the forward stack is empty. If a piece of code is not certain what the states of these stacks are, it can check the Boolean `CanGoBack` and `CanGoForward` properties first. `Frame` also exposes a `BackStackDepth` readonly property that reveals the number of `Pages` currently on the back stack.

Therefore, you could imagine implementing `Page`-level `GoBack` and `GoForward` methods as follows:

```
void GoBack()
{
    if (this.Frame != null && this.Frame.CanGoBack) this.Frame.GoBack();
}

void GoForward()
{
    if (this.Frame != null && this.Frame.CanGoForward) this.Frame.GoForward();
}
```

For advanced scenarios, the entire back and forward stacks are exposed as `BackStack` and `ForwardStack` properties, which are both a list of `PageStackEntry` instances. With this,

you can completely customize the navigation experience, and do things such as removing Pages from the back stack that are meant to be transient.

On a PC, apps with multiple pages typically provide a back button in the corner of the window. On a phone, apps should rely on the hardware back button instead. To respond to presses of the hardware back button, you attach a handler to a static `BackPressed` event on a phone-specific `HardwareButtons` class:

```
#if WINDOWS_PHONE_APP
    Windows.Phone.UI.Input.HardwareButtons.BackPressed +=
        HardwareButtons_BackPressed;
#endif
```

In your handler, you can perform the same `GoBack` logic shown earlier:

```
#if WINDOWS_PHONE_APP
void HardwareButtons_BackPressed(object sender,
    Windows.Phone.UI.Input.BackPressedEventArgs e)
{
    if (this.Frame != null && this.Frame.CanGoBack)
    {
        e.Handled = true;
        this.Frame.GoBack();
    }
}
#endif
```

Setting the `BackPressedEventArgs` instance's `Handled` property to `true` is critical, as it disables the default behavior that closes your app. Here, that only happens once the back stack is empty.



The `HardwareButtons` class also exposes events for when the camera button is half-pressed, pressed, and released.



How do I pass data from one page to another when navigating backward?

Sometimes an app uses a scheme that navigates to a new page in order to have the user select something or fill out a form, and then that data needs to be communicated *back* to the original page when the new page is dismissed. You've already seen how to pass data to the next page when calling `Navigate`, but there is no equivalent mechanism for passing data to the preceding page when calling `GoBack`. (The same is true for `GoForward`.)

Instead, you must find a shared place to store the data where both pages know to look. For example, this could be your own static member on one of your classes, or perhaps even session state might be appropriate to use for this.

### Frame's Content Property

Instead of calling `Navigate`, you can place content in a `Frame` by setting its `Content` property. (This is what the Visual Studio-generated code in `App.xaml.cs` does.) This is much different than calling `Navigate`, however, because doing so clears the back and forward stacks. It also doesn't trigger the typical navigation animation.

Furthermore, the `Frame` control can hold arbitrary content via its `Content` property. This is not a normal thing to do, but using `Frame` in this way enables hosting the content in an isolated fashion. For example, properties that would normally be inherited down the element tree stop when they reach the `Frame`. In this respect, `Frame` acts like a frame in HTML.

## Page Caching

By default, `Page` instances are *not* kept alive on the back and forward stacks; a new instance gets created when you call `GoBack` or `GoForward`. This means you must take care to remember and restore their state, although you will probably already have code to do this in order to properly handle suspension.

You can change this behavior on a `Page`-by-`Page` basis by setting `Page`'s `NavigationCacheMode` property to one of the following values:

- **Disabled**—The default value that causes the page to be recreated every time.
- **Required**—Keeps the page alive and uses this cached instance every time (for `GoForward` and `GoBack`, not for `Navigate`).
- **Enabled**—Keeps the page alive and uses the cached instance only if the size of the `Frame`'s cache hasn't been exceeded. This size is controlled by `Frame`'s `CacheSize` property. This property represents a number of `Pages` and is set to 10 by default.

Using `Required` or `Enabled` can result in excessive memory usage, and it can waste CPU cycles if an inactive `Page` on the stack is doing unnecessary work (such as having code running on a timer). Such pages can use the `OnNavigatedFrom` method to pause its processing and the `OnNavigatedTo` method to resume it, to help mitigate this problem.

When you navigate to a `Page` by calling `Navigate`, you get a new instance of it, regardless of `NavigationCacheMode`. No special relationship exists between two instances of a `Page` other than the fact that they happen to come from the same source code. You can leverage this by reusing the same type of `Page` for multiple levels of a navigation hierarchy, each one dynamically initialized to have the appropriate content. However, if you want every instance of the same page to act as if it's the same page (and "remember" its data from the previously seen instance), then you need to manage this yourself, perhaps with static members on the relevant `Page` class.

## NavigationHelper

If you add any `Page` more sophisticated than a Blank `Page` to your project, it uses a `NavigationHelper` class whose source also gets included in your project. For convenience,

`NavigationHelper` defines `GoBack` and `GoForward` methods similar to the ones implemented earlier. It also adds phone-specific handling of the hardware back button, as well as standard keyboard and mouse shortcuts for navigation. It enables navigating back when the user presses `Alt+Left` and navigating forward when the user presses `Alt+Right`. For a mouse, it enables navigating back if `XButton1` is pressed and forward if `XButton2` is pressed. These two buttons are the browser-style previous and next buttons that appear on some mice.

`NavigationHelper` also hooks into some extra functionality exposed by `SuspensionManager` in order to automatically maintain navigation history as part of session state. To take advantage of this, you need to call one more method inside `OnLaunched` (or `OnWindowCreated`) to make `SuspensionManager` aware of the `Frame`:

```
var rootFrame = new Frame();  
SuspensionManager.RegisterFrame(rootFrame, "AppFrame");
```

Each `Page` should also call `NavigationHelper`'s `OnNavigatedTo` and `OnNavigatedFrom` methods from its overridden `OnNavigatedTo` and `OnNavigatedFrom` methods, respectively, and handle `NavigationHelper`'s `LoadState` and `SaveState` events for restoring/persisting state. `LoadState` handlers are passed the "navigation parameter" object (the second parameter passed to the call to `Navigate`, otherwise `null`) as well as the session state `Dictionary`. `SaveState` handlers are passed only the session state `Dictionary`.

When you create a Hub App project, all these changes are applied automatically. Internally, this works in part thanks to a pair of methods exposed by `Frame`—`GetNavigationState` and `SetNavigationState`—that conveniently provide and accept a serialized string representation of navigation history.



If your app does any navigation, you should use `NavigationHelper` (or copy its code) to automatically handle the hardware back button and the standard keyboard/mouse shortcuts.

## Other Ways to Use `Frame`

Not every app needs to follow the pattern of a `Window` hosting a `Frame` that hosts `Page(s)`. A `Window`'s content doesn't have to be a `Frame`, and you can embed `Frames` anywhere `UIElements` can go. We can demonstrate this by modifying a Hub App project to set the `Window`'s `Content` to a custom `Grid` subclass that we create. Imagine this is called `RootGrid`, and it must be constructed with a `Frame` that it wants to dynamically add to its `Children` collection. It would be used in `App.xaml.cs` as follows:

```
// Instead of Window.Current.Content = rootFrame;  
Window.Current.Content = new RootGrid(rootFrame);
```

`RootGrid` can be added to the project as a pair of XAML and code-behind, shown in Listings 7.3 and 7.4.

**LISTING 7.3** RootGrid.xaml: A Simple Grid Expecting to Contain a Frame

---

```

<Grid x:Class="Chapter7.RootGrid" Background="Blue"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <!-- A 3x3 Grid -->
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <!-- Two Buttons to interact with a Frame -->
  <Button Name="BackButton" Grid.Row="1" HorizontalAlignment="Center"
    Click="BackButton_Click">Back</Button>
  <Button Name="ForwardButton" Grid.Row="1" Grid.Column="2"
    HorizontalAlignment="Center" Click="ForwardButton_Click">Forward</Button>
</Grid>

```

---

**LISTING 7.4** RootGrid.xaml.cs: The Code-Behind That Places the Frame and Interacts with It

---

```

using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

namespace Chapter7
{
  public sealed partial class RootGrid : Grid
  {
    Frame frame;

    public RootGrid(Frame f)
    {
      InitializeComponent();
      this.frame = f;

      // Add the Frame to the middle cell of the Grid
      Grid.SetRow(this.frame, 1);
      Grid.SetColumn(this.frame, 1);
      this.Children.Add(this.frame);
    }
  }

```

```
    this.frame.Navigated += Frame_Navigated;
}

void Frame_Navigated(object sender, NavigationEventArgs e)
{
    if (this.frame != null)
    {
        // Keep the enabled/disabled state of the buttons relevant
        this.BackButton.IsEnabled = this.frame.CanGoBack;
        this.ForwardButton.IsEnabled = this.frame.CanGoForward;
    }
}

void BackButton_Click(object sender, RoutedEventArgs e)
{
    if (this.frame != null && this.frame.CanGoBack)
        this.frame.GoBack();
}

void ForwardButton_Click(object sender, RoutedEventArgs e)
{
    if (this.frame != null && this.frame.CanGoForward)
        this.frame.GoForward();
}
}
```

---

By placing the `Frame` in its middle cell, `RootGrid` is effectively applying a thick blue border to the `Frame` that persists even as navigation happens within the `Frame`. (When used this way, `Frame` seems more like an `iframe` in HTML.) The simple back and forward `Buttons` in `RootGrid` are able to control the navigation (and enable/disable when appropriate) thanks to the APIs exposed on `Frame`. This unconventional window is shown in Figure 7.3, after navigating to the second page.

Although this specific use of `Frame` doesn't seem practical, you can do some neat things with a similar approach. One example would be to have a `Page` that always stays on screen containing a fullscreen `Frame` that navigates to various `Pages`. The reason this is compelling is that the outer `Page` can have app bars that are accessible regardless of what the current inner `Page` is. (App bars are discussed in Chapter 9, "Content Controls.")

If you decide you want your `Page` to truly be the root content in your app's `Window`, you can change the code in `App.xaml.cs` to eliminate the hosting `Frame`. This can work fine, but with no `Frame`, you don't get the navigation features.

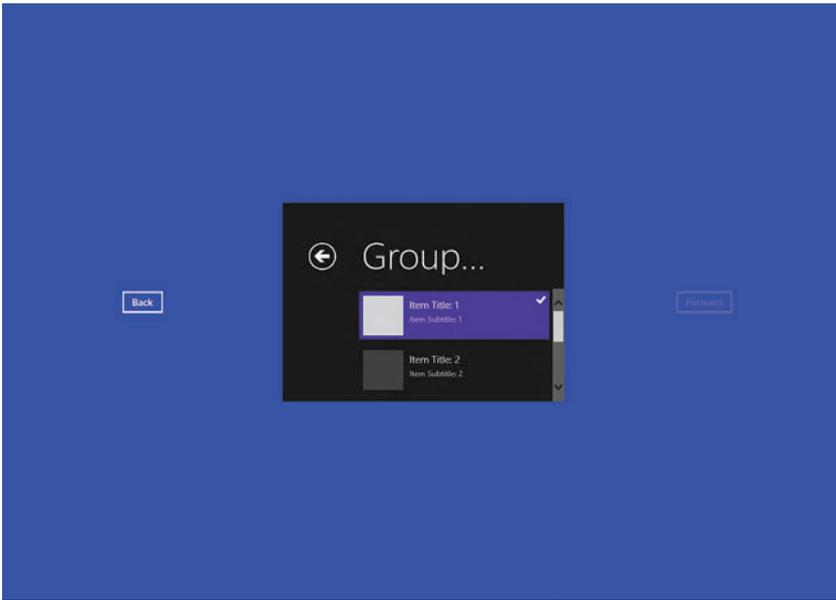


FIGURE 7.3 A `Frame` doesn't have to occupy all the space in an app's window.

## Summary

The design of the Windows Runtime, combined with slick C# language support, could lead one to think, “Threading model? I didn’t realize universal apps *had* a threading model.” In C#, you get to enjoy the benefits of writing an app that largely feels single-threaded, but has all the power of asynchronous code (and a number of parallelism mechanisms employed internally by the Windows Runtime).

Supporting multiple windows within a single app is one area where the code you write can become awkward, but the model of having each window run on a separate UI thread maximizes an app’s responsiveness.

Finally, the features enabled by `Frame` and `Page` support the navigation paradigm common for universal apps. Almost all apps use multiple pages.

*This page intentionally left blank*

# INDEX

## Symbols

### 2D transforms, 72

- combining, 77

- CompositeTransform, 77-78

- MatrixTransform, 78-79

- RotateTransform, 73-74

- ScaleTransform, 74-75

- SkewTransform, 76

- TransformGroup, 78

- TranslateTransform, 77

### 3D transforms, 79-81

- / (slash), 318

## A

- absolute sizing, 96

- accelerometer, 647-649

- shake detection, 650

- tossing motion, 649-650

- Accelerometer.GetDefault method, 648

- AcceptsReturn versus TextWrapping, 302

- accessibility, 35

- high contrast themes, 38-39

- improving screen reading, 36-37

- accessing known folders, 570

- Account Picture Provider, 615-616

- Accuracy, 655

- activating versus launching, 172

- activating apps, 171-172

- activation contracts. *See* contracts

- ActivationKind, 169

- ActualHeight, 65

**ActualWidth, 65****AdControl, 203**

- advanced behaviors, 204-205

**AddDeleteThemeTransition, 462-464****AddFile, 626****AddHandler, 123****adding**

- PC ads to pages, 202-203
- phone ads, 204

**AdEventArgs, 205****adjusting camera settings, MediaCapture, 374-375****admitting logos, 14****ad units, defining in pubCenter, 200-202****advanced behaviors, AdControl, 204-205****advertisements, 200**

- adding PC ads to pages, 202-203
- adding phone ads, 204
- defining ad units in pubCenter, 200-202

**AgeRating, 210****alarm apps, toast notifications, 708-710****Alignment, 69****AllowedFileTypes, 626****altform-XXX, 328****Altitude, 655****AltitudeAccuracy, 655****AltitudeReferenceSystem, 655****Alt key, 155-157****animation, 453**

- custom animation, controlling duration, 475
- custom animations, 472-473, 478
  - From and To, 475-477
- independent versus dependent, 473-474
- property paths, 481-483
- storyboards with multiple animations, 480-481
- tweaking. *See* tweaking animations
- tweaking Storyboard with Timeline, 484-485
- custom keyframe animations, 485
  - discrete keyframes, 488-490
  - easing keyframes, 490
  - linear keyframes, 485-486
  - spline keyframes, 487-488

- dependency properties, 454-455

- easing functions, 490-491
  - built-in easing functions, 493
  - power easing functions, 491-493

- manual animations, 495-497

- theme animations, 466, 469-471
  - Storyboard, 466-468
  - tweaking, 471

- theme transitions, 455-456
  - AddDeleteThemeTransition, 462-464
  - applying to elements, 456-457
  - EdgeUIThemeTransition, 460
  - EntranceThemeTransition, 457-459
  - PaneThemeTransition, 461-462
  - PopupThemeTransition, 459
  - ReorderThemeTransition, 466
  - RepositionThemeTransition, 464-465

- Timeline, 480

- triggering, 468-469

**AppBar, 251-253****AppBarButton, content controls, 234-236**

- BitmapIcon, 240-241

- FontIcon, 238

- PathIcon, 239

- SymbolIcon, 236-237

**app bars, content controls, 249-250**

- AppBar, 251-253

- CommandBar, 254-255

**AppBarToggleButton, 243-244****app commands, 583**

- Play command, 604-605

- Print command, 596-601
  - adding custom options, 603
  - changing default options, 601-602
  - configuring displayed options, 602

- Project command, 606

- Search command, 584
  - customizing Search pane, 586-588
  - programmatically showing Search pane, 588-589

- Search Contract, 584-586

- Settings command, 606
  - handling links with SettingsPane, 608-610
  - showing links, 606-608
- Share command, 589
  - being a share source, 589-592
  - being a share target, 592-595
- app containers, 9**
- app data, 555-557**
  - app files, 560-563
  - app settings, 558-560
    - local settings, 558
    - roaming settings, 559
- AppendResultSuggestion, 396**
- app files, app data, 560-563**
- application definition, updating XAML, 24-29**
- ApplicationId, 204**
- Application tab, package manifest, 11**
- Appointments Provider Contract, 635**
  - source, 635-636
  - targets, 636-637
- App.OnLaunched, 29**
- app packages, 10-11**
- apps**
  - compiling, 9
  - creating, 7-10
  - lifecycle. *See* lifecycle
  - Mail app, 591
  - not running apps, 161
  - running apps, 161
  - sideloaded apps, 220
  - submitting to Windows Store, 40-41
  - suspended apps, 161
- App Specific Hardware ID (ASHWID), 220**
- app themes, 228-229**
  - Flyouts, 234
- AppX, 10**
- App.xaml, 228**
- App.xaml.cs, 28**
- AreTransportControlEnabled, 363**
- Arguments, 169**
- ASHWID (App Specific Hardware ID), 220**
- AssemblyFileVersion, 29**
- AssemblyVersion, 29**

- AsTask, 183**
- ASTA threads, 182**
- async, 182-184**
- asynchronous operations, threading model, 182-184**
- attached properties, 88-89**
- attached property providers, 88**
- attributes, XAML, 44-45**
- audio**
  - background audio, 637
    - adding Background Tasks declaration, 638
    - AudioCategory, 637-638
    - SystemMediaTransportControls, 638-641
  - capturing, MediaCapture, 377
- AudioCategory, 637-638**
- automated testing, Windows Store (testing features), 224-225**
- automation ID, 268**
- AutoPlay Content, contracts, 617-620**
- AutoPlay Device, contracts, 617-620**
- AutoReverse, 479**
- autosizing, 97**
- await, 182-185**

## B

- back button, 193**
- BackEase, 494**
- background audio, 637**
  - adding Background Tasks declaration, 638
  - AudioCategory, 637-638
  - SystemMediaTransportControls, 638-641
- BackgroundColor, 350**
- BackgroundDownloader, 577**
- Background property, 127**
- BackgroundTaskBuilder, 643**
- background tasks, 637**
  - background audio, 637-638
    - adding Background Tasks declaration, 638
    - SystemMediaTransportControls, 638-641

- custom background tasks, 641
  - conditions, 645-646
  - IBackgroundTask, 641-642
  - leveraging background work from the foreground, 644
  - registering, 642-643
  - triggers, 644-645
- background threads, 181**
- background transfers, networking, 576-579**
- backgrounds, logos, 15**
- badges, 701-703**
- BarcodeScanner, 669**
- barcode scanners, 668**
  - BarcodeScanner, 669
  - claiming devices, 669-670
  - enabling, 670
  - retrieving data, 670-671
- BaseTargetType, 503**
- BaseTextBlockStyle, 504**
- BeginTime, 478**
- Bézier curves, 429**
- Binding, 529-530**
  - C#, 531-532
  - collections, 534-536
  - customizing data flow, 532-533
  - improving display, 537
  - RelativeSource, 531
  - sharing source with DataContext, 533-534
  - to plain properties, 532
- BitmapDecoder, 330-331**
  - pixel data, 332
- BitmapEncoder, 340, 343**
- BitmapEncoderId, 350**
- BitmapFrame, 333**
- BitmapIcon, 240-241**
- BitmapImage, 317**
- BitmapProperties, reading from decoders, 337**
- BitmapTransform, 334**
- BitmapTypedValues, 341**
- Blocks, 291**
- Bluetooth Low Energy (Bluetooth LE), 676**
- Bluetooth Smart, 676**
- Bold element, 293**

- BounceEase, 494**
- branding="logo," 690**
- Brushes, 436**
  - color brushes
    - LinearGradientBrush, 437-443
    - SolidColorBrush, 436-437
  - tile brushes, 443-444
    - ImageBrush, 444-445
    - WebViewBrush, 446-450
- Buffering Progress, 360**
- Build Action, 318**
- built-in easing functions, 493**
- bulk transfers, custom USB devices, 681**
- ButtonBase, 234**
- buttons, content controls, 230-231**
  - ButtonBase, 234
  - Flyout, 231-233

## C

- C#**
  - Binding, 531-532
  - GridLength, 98
  - markup extensions, 51
  - XAML and, 44-45
    - loading and parsing XAML at runtime, 56-57
    - naming XAML elements, 57
    - support for, 57-58
- cached composition, 451**
- caching, controlling, 575**
- CameraCaptureUI, 367-368**
  - capturing photos, 368-370
  - capturing video, 370-371
- camera settings, adjusting (MediaCapture), 374-375**
- CanAddFile, 626**
- CancelCommandIndex property, 399**
- CanGoBack, 192**
- CanGoForward, 192**
- Canvas, 88-89**
  - child layout properties, 89

**Capabilities tab, package manifests, 16-17**

- device capabilities, 18
- file capabilities, 17-18
- identity capabilities, 19
- network capabilities, 19

**capture**

- CameraCaptureUI, 367-368
  - capturing photos, 368-370
  - capturing video, 370-371
- CaptureElement, 371
  - capturing photos, 374
  - showing previews, 371-373
- MediaCapture
  - adjusting camera settings, 374-375
  - capturing audio only, 377
  - capturing video, 376

**CaptureElement, 367, 371**

- capturing photos, 374
- showing previews, 371-373

**CaptureFileAsync, 369****capturing audio only, MediaCapture, 377****capturing photos**

- CameraCaptureUI, 368-370
- CaptureElement, 374

**capturing pointers, 123-126****capturing video**

- CameraCaptureUI, 370-371
- MediaCapture, 376

**capturing web content, WebView, 418-419****CarouselPanel, 266**

- WrapGrid, 263

**catalogs**

- displaying, 214
- managing custom catalogs, 216-217

**CenterX, 73-74****CenterY, 73-74****certification process, Windows Store, 40-41****Character Map app, 238****CharacterSpacing, 284-286****CheckBox, 244****child layout properties**

- Canvas, 89
- Grid, 96

StackPanel, 91

VariableSizedWrapGrid, 103

**child object elements, 52**

- collection items, 52
  - dictionaries, 54
  - lists, 53
- content property, 52
- processing rules, XAML, 55-56
- type conversion, 54-55

**children, direct child, 229****ChildrenTransitions, 457****ChildTransitions, 457****ChooseSuggestionOnEnter, 393****choosing minimum width, 85-86****CircleEase, 494****ClaimedBarcodeScanner, 669****ClaimedMagneticStripeReader, 672****claiming devices, barcode scanners, 669-670****ClickMode, 234****client-side code, push notifications, 717-719****clipping, 104-105****ClockIdentifier property, 411****Closing, 626****code-behind, 21****collection items, child object elements, 52**

- dictionaries, 54
- lists, 53

**collections**

- Binding, 534-536
- customizing view of, 546
  - grouping, 547-549
  - navigating, 550

**color brushes**

- LinearGradientBrush, 437-443
- SolidColorBrush, 436-437

**ColumnDefinition, 94****columns, sizing in Grid, 96-97****ColumnSpan, 94****combining 2D transforms, 77****ComboBox, items controls, 265-267****CommandBar, 250, 254-257****commands, app commands. See app commands****commas in Geometry strings, 435**

**CommonStates, 520****communicating with**

- custom Bluetooth devices, 675-676
- custom Bluetooth Smart devices, 677-678
- custom HID devices, 683-684
- custom USB devices, 680-681

**compass, 651****compiling apps, 9****composing**

- HTML with XAML, WebView, 416
- Javascript with C#, WebView, 417-418

**CompositeTransform, 77-78****conditions, custom background tasks, 645-646****configuring displayed options, Print command, 602****configuring scan sources, image scanners, 666-667****config-XXX, 328****connecting to**

- custom Bluetooth devices, 674-675
- custom Bluetooth Smart devices, 677
- custom HID devices, 683
- custom USB devices, 680

**connection info, networking, 581-582****constructor, 25****consumable products**

- in-app purchases, 214-218
- initiating and fulfilling purchases, 214-215
- managing custom catalogs, 216-217

**Contact Contract, 631**

- sources, 631-632
- targets, 633-635

**ContactManager, 631-632****Contact Picker, 628**

- sources, 628-629
- targets, 630

**ContainerContentChanging, 552****ContainFile, 626****Content, 230****content alignment, 70****content checks, 40****ContentControl, Content property, 511****ContentControl property, 513****content controls, 229-230**

- AppBarButton, 234-236
- BitmapIcon, 240-241
- FontIcon, 238
- PathIcon, 239
- SymbolIcon, 236-237

**app bars, 249-250**

- AppBar, 251-253
- CommandBar, 254-255

**AppBarToggleButton, 243-244****buttons, 230-231**

- ButtonBase, 234
- Flyout, 231-233

**CheckBox, 244****HyperlinkButton, 241****RadioButton, 245-246****RepeatButton, 242****ToggleButton, 243****ToolTip, 246-249****content overflow, 84, 103****clipping, 104-105****scaling, 111-114****scrolling, 106-110****ContentPresenter element, 513****content property, child object elements, 52****Content property, 48**

- ContentControl, 511
- Frame, 194

**ContentThemeTransition, 460****ContentTransitions, 457****Content URLs tab, package manifests, 20****contract activations, 614****contracts, 613-614**

- Account Picture Provider, 615-616
- Appointments Provider Contract, 635
  - source, 635-636
  - targets, 636-637

**AutoPlay Content, 617-620****AutoPlay Device, 617-620**

- Contact Contract, 631
  - sources, 631-632
  - targets, 633-635
- Contact Picker, 628
  - sources, 628-629
  - targets, 630
- File Open Picker, 624-626
- File Save Picker, 627-628
- File Type Associations, 620-622
- Protocol, 623
- Share contracts, 614
- Control, 61**
- controlling caching, 575**
- controlling duration, custom animation, 475**
- controls, 228**
  - content controls, 229
  - content controls. *See* content controls
  - custom HID devices, 684
  - date and time controls. *See* date and time controls
  - Hub controls. *See* Hub controls
  - items controls. *See* items controls
  - popup controls. *See* popup controls
  - ProgressRing, 411-412
  - range controls. *See* range controls
  - text controls. *See* text controls
  - ToggleSwitch, 412-413
  - WebView, 413
    - capturing web content, 418
    - composing HTML with XAML, 416
    - composing JavaScript with C#, 417
    - navigation, 414-416
  - XAML controls, 236
- control templates, 509-511**
  - tweaking, 518-519
  - versus data templates, 539
- control transfers, custom USB devices, 680**
- Convert, 545**
- ConvertBack, 545**
- cookies, manipulating, 575**
- CoreDispatcher, 185-186**
- CoreWindow.GetAsyncKeyState, 157**
- CoreWindow.GetKeyState, 157**
- CPU Usage, 10**
- CreateForInPlacePropertyEncodincugAsync, 347**
- CreateForTranscodingAysnc, 346**
- CreatePrintTask, 598,-599**
- CreateWindowAsync, 188**
- CreationCollisionOption, 561**
- CrossSliding, 131, 134**
- curly braces, escaping, 50**
- CurrentApp, 221**
- CurrentAppSimulator, 225**
  - Windows Store, testing features, 222-223
- current location, 653-655**
- CurrentMarket, 209**
- current value display, customizing, 390**
- custom animations, 472-473, 478**
  - controlling duration, 475
  - From and To, 475-477
  - independent versus dependent, 473-474
  - property paths, 481-483
  - storyboards with multiple animations, 480-481
  - tweaking Storyboard with Timeline, 484-485
- custom background tasks, 641**
  - conditions, 645-646
  - IBackgroundTask, 641-642
  - leveraging background work from the foreground, 644
  - registering, 642-643
  - triggers, 644-645
- custom Bluetooth devices, 673**
  - communicating with, 675-676
  - connecting to, 674-675
  - declaring the right device, 673-674
- custom Bluetooth Smart devices, 676**
  - communicating with, 677-678
  - connecting to, 677
  - declaring the right device capability, 676-677
- custom HID devices, 682**
  - communicating with, 683-684
  - connecting to, 683
  - declaring the right device capability, 682
- customizing**
  - current value display, 390
  - data flow, Binding, 532-533

- Grid, 98
- launching apps, programmatically, 178-179
- logo images, 13-16
- page rendering, Image element, 350-353
- playback, MediaElement, 358-359
- ScrollView, 106-107
- Search pane, 586-588
- splash screens, 11-12
- text rendering, 284-287
  - CharacterSpacing, 285-286
  - IsColorFontEnabled, 288-289
  - OpticalMarginAlignment, 287-288
  - TextLineBounds, 286-287
  - TextReadingOrder, 288
- view of collections, 546
  - grouping, 547-549
  - navigating, 550
- custom keyframe animations, 485**
  - discrete keyframes, 488-490
  - easing keyframes, 490
  - linear keyframes, 485-486
  - spline keyframes, 487-488
- custom options, Print command, 603**
- CustomResource, 509**
- custom USB devices, 679**
  - communicating with, 680-681
  - connecting to, 680
  - declaring the right device capability, 679
- custom Wi-Fi Direct devices, 684-686**

## D

- data, 555**
  - app data, 555-557
    - app files, 560-563
    - app settings, 558-560
  - files, 555-557
  - folders, 555-557
  - networking, 572
    - background transfers, 576-579
    - connection info, 581-582
    - HTTP requests, 572-576
    - sockets, 579
    - syndication, 579-581
  - sharing, 592
  - user data, 555, 563
    - file picker, 564-567
    - folder picker, 567-570
    - libraries and special folders, 570-571
- data binding, 529**
  - Binding, 529-530
  - customizing view of collections, 546
    - grouping, 547-549
    - navigating, 550
  - high-performance rendering, ListView and GridView, 550-554
  - rendering, 538
    - data templates, 538-542
    - value converters, 543-546
- DataContext, Binding, 533-534**
- data flow, customizing (Binding), 532-533**
- DatagramSocket, 579**
- DataPackage, 592**
- data templates**
  - versus control templates, 539
  - rendering, 538-542
- data virtualization, ListView, 272**
- DataWriter, 325**
- date and time controls, 407**
  - DatePicker, 408-409
  - TimePicker, 410-411
- DatePicker, 408-409**
- DateTime, 336-337**
- DateTimeOffset, 336**
- Declarations tab, package manifests, 19-20**
- declaring the right device, custom Bluetooth devices, 673-674**
- declaring the right device capability**
  - custom Bluetooth Smart devices, 676-677
  - custom HID devices, 682
  - custom USB devices, 679

**decoding images, 330-331**

- pixel data, 331-333

- reading metadata, 335-339

**default options, Print command, 601-602****DefaultPlaybackRate, 359****DefaultSectionIndex property, 405****default themes, 227****deferral, suspending apps, 166****DependencyObject, 61, 447****dependency properties, animation, 454-455****dependency property, 453****DesiredSize, 65****DestinationHeight, 350****DestinationWidth, 350****DetachPixelData, 332****device capabilities, Capabilities tab (package manifests), 18****DeviceCapability, 673****Device Manager, 679****Device option, 8****device protocol APIs, 663****devices**

- barcode scanners, 668

  - BarcodeScanner, 669

  - claiming devices, 669-670

  - enabling, 670

  - retrieving data, 670-671

- custom Bluetooth devices, 673

  - communicating with, 675-676

  - connecting to, 674-675

  - declaring the right device, 673-674

- custom Bluetooth Smart devices, 676

  - communicating with, 677-678

  - connecting to, 677

  - declaring the right device capability, 676-677

- custom HID devices, 682

  - communicating with, 683-684

  - connecting to, 683

  - declaring the right device capability, 682

- custom USB devices, 679

  - communicating with, 680-681

  - connecting to, 680

  - declaring the right device capability, 679

  - custom Wi-Fi Direct devices, 684-686

  - fingerprint readers, 664

  - image scanners, 664-665

    - configuring scan sources, 666-667

    - ImageScanner, 665

    - performing scans, 665-666

    - previews, 667-668

  - magnetic stripe readers, 671-672

    - retrieving data, 672-673

**device scenario APIs, 663****dictionaries, collection items (child object elements), 54****direct child, 229****Disabled case, ScrollViewer, 109****disabling flipped orientations, 11****discrete keyframes, 488-490****Dispatcher, 185****DispatcherTimer, 495****displaying multiple windows, 186-189****displaying catalogs, durable purchases, 213-214****DisplayMemberPath, 537****DoQuadraticInterpolation, 495****DoubleAnimation, 472-473****DownloadProgress, 360****dragging, 143****DragItemThemeAnimation, 470****drop-down, 266****DropTargetItemThemeAnimation, 470****durable products, 211-213**



  - determining if a product has been purchased, 211

  - initiating purchases, 212-213

**durable purchases, displaying catalogs, 213-214****Duration, 475, 657****Duration property, theme animation, 472****DwellTime, 657****dynamic images, generating with WriteableBitmap, 322-325**

## E

- EaselInOut**, 494-495
- EaseOut**, 494
- easing functions**, 490-491
  - built-in easing functions, 493
  - power easing functions, 491-493
- easing keyframes**, custom keyframe animations, 490
- EdgeGesture**, 135
- EdgeUIThemeTransition**, 460
- effects**
  - MediaTranscoder, 383
  - playback, MediaElement, 361
- ElasticEase**, 494
- elements**, XAML, 44-45
- Ellipse**, 423
- EllipseGeometry**, 428
- embedding UIElements**, RichTextBlock, 297-298
- Emulator option**, 8
- EnableDependentAnimation**, 474
- enabling barcode scanners**, 670
- encoding images**, 339-341
  - transcoding, 344-347
  - writing metadata, 343-344
  - writing pixel data, 341-342
- encoding options**, 341
- Energy Consumption**, 10
- EntranceThemeTransition**, 457-459
- escaping curly braces**, 50
- EvenOdd**, 431
- event bubbling**, 120
- event processing**, order of, 45
- events**
  - manipulation events, 137-140
  - playback, MediaElement, 359-360
  - routed events, 120
- event triggers**, 468
- explicit runs versus implicit runs**, 291
- ExponentialEase**, 494

## F

- Facedown**, 652
- Faceup**, 652
- FadeInThemeAnimation**, 469
  - troubleshooting, 471
- FadeOutThemeAnimation**, 469
- failure conditions**, initiating and fulfilling purchases, 216
- FallbackValue**, Binding, 530
- feature-differentiated trials**, 206-207
- feature reports**, custom HID devices, 684
- file activation**, 619
- FileActivatedEventArgs**, 622
- file capabilities**, Capabilities tab (package manifests), 17-18
- FileOpenPicker**, 568
- File Open Picker contract**, 624-626
- FileOpenPickerUI**, 626
- file picker**, user data, 564-567
- FileRemoved**, 626
- files**, 555-557
- File Save Picker**, 627-628
- FileSavePicker**, 567
- File Type Associations**, contracts, 620-622
- FileTypeFilter**, 569
- file variations**
  - loading automatically, Image element, 326-327
  - loading manually, Image element, 329
- Fill**, 127, 423
- FillBehavior**, 480
- FillRule**, 431
- fingerprint readers**, 664
- Flat line cap**, 426
- flipped orientations**, disabling, 11
- FlipView**, 274-276
- FlowDirection**, 71, 91
- FlushAsync**, 341
- Flyouts**
  - app themes, 234
  - buttons, content controls, 231-233
- focus**, keyboard input, 158-159

**FocusStates**, 520  
**folder picker, user data**, 567-570  
**folders**, 555-557  
**FontIcon**, 238  
**footers, ListView**, 270  
**FooterTransitions**, 457  
**format, transcoding (MediaTranscoder)**, 381-382  
**format patterns, date and time**, 409  
**FormattedPrice**, 209  
**Frame**  
     Content property, 194  
     navigating pages, 195-197  
**FrameworkElement**, 61, 75  
**freemium business model**, 199  
**free trials**, 205-206  
     feature-differentiated trials, 206-207  
     helping user purchase full license, 208-210  
     time-based trials, 206  
**From, custom animations**, 475-477  
**From+By**, 477  
**FromHorizontalOffset**, 458  
**FromIdAsync**, 675  
**FromVerticalOffset**, 458  
**full-screen mode**, 358

## G

**games, pausing**, 165  
**GATT (Generic Attribute Profile)**, 676-678  
**GattCharacteristic**, 678  
**generating dynamic images with**  
     **RenderTargetBitmap**, 325  
**Geocoordinate**, 655  
**Geofence**, 656-658  
**geofencing**, 656-659  
**Geolocator**, 654  
**Geometry**, 428-429  
     FillRule, 431  
     GeometryGroup, 432-433  
     PathFigure, 429  
     PathSegment, 429  
         representing as strings, 433-435  
         string commands, 434  
**GeometryGroup**, 428, 432-433  
**GeometryTransform**, 422  
**Geopoint**, 655  
**GeoshapeType**, 655  
**GestureRecognizer**, 131-134  
     mouse input, 143  
**gestures**, 118, 130-136  
     EdgeGesture, 135-136  
     GestureRecognizer, 131-134  
     UIElement gesture events, 136  
**GetDeferral**, 166  
**GetGeopositionAsync**, 659  
**GetIntermediatePoints**, 122  
**GetParentAsync**, 556  
**GetPosition**, 122  
**GetPreviewPage event**, 598  
**GetPropertiesAsync**, 338, 347  
**GetThumbnailAsync**, 329, 335  
**GetValue**, 455  
**gift cards**, 200  
**globalization**, 29-35  
**GoBack**, 192  
**gradients**, 443  
**Grid**, 91-99  
     child layout properties, 96  
     comparing to other panels, 99  
     customizing, 98  
     mimicking Canvas, 99  
     mimicking StackPanel, 99  
     percentage sizing, 97  
     sizing rows and columns, 96-97  
**GridLength, C#, 98**  
**GridView**, 273-274  
     high-performance rendering, 550-553  
**GridView incremental rendering**, 550-553  
**grouping, customizing view of collections**, 547-549  
**GroupStyle**, 548  
**gyrometer**, 651

## H

- halting bubbling, 122-123
  - handwriting, rendering, 146-149
  - handwriting recognition, 150-153
  - hardware back button, 193
  - HardwareButtons, 193
  - headers, ListView, 270
  - HeaderTransitions, 457
  - Heading, 655
  - height, 64-65
  - HelloRealWorld.Shared, 8
  - HelloRealWorld.Windows, 8
  - HelloRealWorld.WindowsPhone, 8
  - helping user purchase full license, 208-210
  - Hidden case, ScrollViewer, 108
  - hidden managed-to-unmanaged code transitions, 324
  - HID (Human Interface Device), custom. *See* custom HID devices
  - hiding software keyboards, 308
  - HidInputReport, 683
  - HidOutputReport, 683
  - HighContrastChanged event, 39
  - high contrast themes, accessibility, 38-39
  - high-performance rendering, ListView and GridView, 550-554
  - hijacking existing properties for new purposes, 517
  - history suggestions, SearchBox, 392
  - hit testing, 126-127
  - Holding, 131
  - HoldingState, 134
  - homeregion-XXX, 328
  - HorizontalAlignment, 69
  - HorizontalChildrenAlignment, 101
  - HorizontalScrollMode, 109
  - HtmlFormatHelper.CreateHtmlFormat, 590
  - HttpBaseProtocolFilter, 575
  - HttpClient, 572
  - HTTP filters, 574
  - HTTP GET request, 572
  - HTTP prefetching, 576
  - HTTP requests, networking, 572-576
  - HttpResponseMessage, 575
  - Hub controls, 403-405
    - HubSection, 405-406
      - jumping to, 406-407
  - HubSection, 405-406
    - jumping to, 406-407
  - HyperlinkButton, 241
- 
- IAsyncAction, 574
  - IBackgroundTask, 641-642
  - IconElement, 236, 240-241
  - identity capabilities, Capabilities tab (package manifests), 19
  - IlluminanceInLux, 651
  - image-based large templates, 696
  - ImageBrush, 444-445
  - Image element, 316
    - decoding images, 330-331
      - pixel data, 331-333
      - reading metadata, 335-339
    - encoding, writing metadata, 343-344
    - encoding images, 339-341
      - writing pixel data, 341-342
    - generating dynamic images with RenderTargetBitmap, 325
    - generating dynamic images with WriteableBitmap, 322-325
    - loading file variations manually, 329
    - multiple environments, 325-326
      - loading file variations automatically, 326-327
      - loading file variations manually, 329
    - multiple variations, leveraging resource packages, 329-330
    - referencing files with URIs, 316-318

- rendering PDF content as an image, 347-348
    - customizing page rendering, 350-353
    - rendering a PDF page, 348-350
  - scaling, 326
  - stretching with nine-grid feature, 319-322
  - transcoding, 344-347
- ImageOpened, 317**
- ImageProperties, 335-336**
- images**
- decoding, 330-331
    - pixel data, 331-333
    - reading metadata, 335-339
  - encoding, 339-341
    - transcoding, 344-347
    - writing metadata, 343-344
    - writing pixel data, 341-342
- ImageScanner, 665**
- image scanners, 664-665**
- configuring scan sources, 666-667
  - ImageScanner, 665
  - performing scans, 665-666
  - previews, 667-668
- implicit runs versus explicit runs, 291**
- implicit styles, 505**
- TargetType, 506
- improving screen reading, 36-37**
- in-app purchases, 210-211**
- consumable products, 214-218
  - durable products, 211-213
- inclinometer, 651**
- incoming call notifications, toast notifications, 708-710**
- incremental item rendering, 551**
- ListView, 272
- independent versus dependent, custom animations, 473-474**
- index markers, 360**
- inertia, manipulation, 141**
- inheritance, style inheritance, 504-505**
- InitializeAsync, 372, 373**
- InitializeComponent, 24**
- initiating purchases, durable products, 212-213**
- InkManager, 150-152**
- Inlines, 291**
- INotifyCollectionChanged, 536**
- input, 65**
- keyboard input. *See* keyboard input
  - mouse input. *See* mouse input
  - pen input. *See* pen input
  - touch input. *See* touch input
- input reports, custom HID devices, 683**
- InputScope, TextBox, 304-306**
- input scopes, 306**
- restricting what is typed in text boxes, 307
- Intellisense, 505**
- intermediate-mode graphics, 451**
- interrupt transfers, custom USB devices, 681**
- IRandomAccessStream, 318-319**
- IsAvailable, 557**
- IsBarrelButtonPressed, 144**
- IsColorFontEnabled, 285, 288-289**
- IsEditable property, 267**
- IsEraser, 144**
- IsFullWindow, 358**
- IsHorizontalMouseWheel, 142**
- IsHorizontalScrollChainingEnabled, 109**
- IsIgnoringHighContrast, 350**
- IsIndeterminate, 388**
- IsInRange, 145**
- IsInverted, 145**
- IsPressed, 455**
- IsScrollInertiaEnabled, 109**
- IsSpellCheckEnabled, 302**
- IsSwipeEnabled, 270**
- IsTextPredictionEnabled, 303**
- IStorageItem, 555-556**
- IsVerticalScrollChainingEnabled, 109**
- ItemContainerTransitions, 457**
- item controls, 259-261**
- ItemHeight property, 101**
- item rendering, ListView, 272**
- items, reordering (ListView), 272**
- ItemsControl, 260-261, 538**
- items controls**
- ComboBox, 265-267
  - FlipView, 274-276

- GridView, 273-274
- items panel, 262-265
- ListBox, 267-269
- ListView, 269-270
- MenuFlyout, 279-280
- SemanticZoom, 276-279

- items panel, 262-265
- ItemWidth property, 101
- ITextCharacterFormat, 309
- ITextDocument, 309-310
- ITextParagraphFormat, 309
- ITextRange, 309
- ITextSelection, 309

## J

- JavaScript, composing with C# (WebView), 417-418
- jump effect, 477
- jumping to HubSection, 406-407

## K

- KeyboardCapabilities, 154
- keyboard input, 153-154
  - focus, 158-159
  - keyboard modifiers in pointer events, 157-158
  - key states, 155-157
  - UIElement keyboard events, 154-155
- keyboard modifiers in pointer events, 157-158
- keyboards, software keyboards, 308
- keyboard shortcuts, panes, 611
- keyframe animations, custom. *See* custom keyframe animations
- KeyModifiers, 122
- key states, 155-157
- keywords, XAML, 59-60
- killing apps, lifecycle, 163-164
- know folders, accessing, 570
- KnownFolders, 570

## L

- landscape, 652
- landscape-flipped, 652
- language code, 30
- languages, 31-35
  - Pseudo Language, 32-33
- language-XXX, 328
- lang-XXX, 328
- Large 310x310 Logo, 13
- large tile templates, 696
- Latitude, 655
- LaunchActivatedEventArgs, 168-169
- launching
  - versus activating, 172
  - purchasing dialog, 208-209
- launching apps, 168
  - LaunchActivatedEventArgs, 168-169
  - PreviousExecutionState, 169-170
  - programmatically, 176
    - customizing, 178-179
    - URI, 176-177
- LaunchUriAsync, 177
- layout, 63
  - 2D transforms. *See* 2D transforms
  - content overflow, 103
    - clipping, 104-105
    - scaling, 111-115
    - scrolling, 106-110
  - panels
    - Canvas, 88-89
    - Grid, 91-99
    - StackPanel, 90-91
    - VariableSizedWrapGrid, 99-103
  - position, 68
    - Alignment, 69
    - content alignment, 70
    - FlowDirection, 71
  - size
    - height, 64-65
    - Margin, 66-67
    - Padding, 66-67
    - width, 64-65

- window size and location, 84-85
  - choosing minimum width, 85-86
  - discovering orientation, 86-87

**layoutdir-XXX, 328**

**LeftHeader, 109**

**leveraging resource packages, Image element, 329-330**

**libraries**

- managing, 571
- user data, 570-571

**LicenseChanged event, 207**

**LicenseInformation class, 208**

**licenses, helping user purchase full license, 208-210**

**lifecycle, 161-162**

- activating apps, 171-172
- killing apps, 163-164
- launching apps, 168
  - LaunchActivatedEventArgs, 168-169
  - PreviousExecutionState, 169-170
- resuming apps, 166
- suspending apps, 164-165
  - deferral, 166
  - Suspending event, 165-166
- terminating apps, 167
- transitioning from running to killing, 163-164

**LightSensor, 651**

**LightSensorReading, 651**

**Line, 423-424**

**LinearGradientBrush, 437-443**

**linear interpolation, 473**

**linear keyframes, 485-486**

**LinearXXXKeyFrame, 487**

**LineBreak, 291**

**LineGeometry, 428**

**LineHeight, 284**

**LineSegment, 430**

**links**

- Settings command, handling with SettingsPane, 608-610
- showing, Settings command, 606-608

**ListBox, 267-269**

**ListBoxItem, 261**

**listing details, 209**

**listings**

Applying ScaleTransform to Buttons in a StackPanel, 74-75

App.xaml.cs—The Code-Behind for the App Class, 26-28

App.xaml—The Markup for the App Class, 25

Button's Control Template from Its Default Style, 520-522

A Complex ToolTip, 248

The ControlTemplate for Button from Listing 17.1, Enhanced with Visual States, 524-527

CustomSettingsFlyout.xaml.cs: The Code-Behind File for a Custom Settings Pane, 609

CustomSettingsFlyout.xaml: The User Interface for a Custom Settings Panel, 608

insert.js: The Auto-Generated Insert Script, with Demo Code for Sending a Push Toast Notification, 714-715

MainPage.xaml.cs: Creating and Monitoring a Geofence, 657-659

MainPage.xaml.cs: Generating Random Audio for MediaElement to Play, 365-367

MainPage.xaml.cs: Handling Manipulation Events to Enable Panning, Rotating, and Zooming, 139

MainPage.xaml.cs: How to Make GestureRecognizer Raise Its Events, 132-134

MainPage.xaml.cs: Performing Simple Drag and Drop, 124-125

MainPage.xaml.cs: Rendering Ink Strokes on a Canvas, 146-149

MainPage.xaml.cs: Supporting Print and Print Preview, 596-598

MainPage.xaml.cs: Tracking Multiple Pointers, 128-130

MainPage.xaml.cs: Using CreateWindowAsync to Create Then Show a New Window, 188-189

MainPage.xaml—Markup with User-Visible English Text Removed, 30

MainPage.xaml—The Initial Markup for the Main Page, 21

MainPage.xaml—Updated Markup for the HelloRealWorld App, 22

Modifying a User-Selected File In-Place by Inverting Its Pixels and Changing Metadata, 344-346

- push.register.cs: Sending the Current Channel URI to the Service's insert.js Script, 717-718
- RootGrid.xaml: A Simple Grid Expecting to Contain a Frame, 196
- RootGrid.xaml.cs: The Code-Behind That Places the Frame and Interacts with It, 196-197
- Updates to the ControlTemplate for Button That Make It More Reusable, 514-515
- Updating an App's Tile with Content for Three Tile Sizes, 688-689
- WindowsStoreProxy.xml: The Default Windows Store Data Used by CurrentAppSimulator, 222-223
- The Zigzag Animation for Figure 16.9, 486
- lists, collection items (child object elements), 53**
- ListView, 269-270**
  - data virtualization, 272
  - footers, 270
  - headers, 270
  - high-performance rendering, 550-554
  - incremental item rendering, 272
  - reordering items, 272
  - ScrollIntoView, 270
  - selection, 270-271
- ListView incremental rendering, 550-553**
- live regions, 37**
- live tiles, 687**
  - tile templates, 688-690
    - image-based large templates, 696
    - large tile templates, 696
    - medium tile templates, 690
    - peek medium templates, 692
    - peek wide templates, 696
    - static image-based wide templates, 694-696
    - static medium templates, 690
    - static text-only wide templates, 693-694
    - text-only large templates, 696
    - wide tile templates, 693
  - updating options, 699-701
- LoadComponent method, 59**
- loading**
  - file variations automatically, Image element, 326-327
  - file variations manually, Image element, 329
  - XAML at runtime, 56-57

- local content suggestions, SearchBox, 395**
- local files, app data, 561**
- localization, 29**
- local notifications, 690**
- local options, showing toast notifications, 710**
- local settings, app settings (app data), 558**
- local updates, live tiles, 700**
- location, 652**
  - current location, 653-655
  - geofencing, 656-659
- lock screen, 719-721**
  - secondary tiles, 721
- logical pixels, 66**
- logo images, customizing, 13-16**
- logos**
  - admitting, 14
  - backgrounds, 15
  - scaling, 14-16
- Longitude, 655**

## M

- magnetic stripe cards, 672**
- MagneticStripeReader, 672**
- magnetic stripe readers, 671-672**
  - retrieving data, 672-673
- Mail app, 591**
- main page logic, XAML (updating), 23-24**
- main page user interface, XAML (updating), 21-22**
- making money**
  - advertisements. *See* advertisements
  - free trials. *See* free trials
  - in-app purchases, 210-211
  - validating Windows Store receipts, 218-220
- managing custom catalogs, consumable products, 216-217**
- manipulating cookies, 575**
- manipulation, inertia, 141**
- ManipulationDelta, 138-140**
- manipulation events, 137-140**

**manipulations, 118, 136**

manipulation events, 137-140

**ManipulationStarting, 137**

**manual animations, 495-497**

**Margin, 66-67**

**markers, playback (MediaElement,) 360-361**

**Markers, 361**

**markup compatibility, XML namespace, 47**

**markup extensions**

C#, 51

XAML, 49-51

**Matrix3DProjection, 81**

**MatrixTransform, 78-79**

**MaxHeight, 64**

**MaximumRowsOrColumns, 101**

**MaxLines, 284**

**MaxWidth, 64**

**media**

capture. *See* capture

playback, 356-357

speech synthesis, 384

transcoding. *See* transcoding

**MediaCapture, 373-374**

adjusting camera settings, 374-375

capturing audio only, 377

capturing video, 376

terminating, 373

**media content, playback, 357-358**

**MediaElement, 356**

playback, 356-357

adding effects, 361

customizing, 358-359

markers, 360-361

media content, 357-358

MediaPlayer, 363-364

playing custom media formats, 365-367

states and events, 359-360

using as a media player, 362

**MediaEncodingProfile, 381**

**MediaFailed, 360**

**MediaOpened, 360**

**MediaPlayer, 363-364**

**media players, playback (MediaElement), 362**

**MediaStreamSource, 365-367**

**MediaTranscoder**

changing format, 381-382

changing quality, 378-379

trimming, 382-383

**MediaTranscoder class, 378**

**medium tile templates, 690**

**Memory Usage, 10**

**MenuFlyout, 279-280**

**MenuFlyoutItem, 279**

**MenuFlyoutSeparator, 280**

**MessageDialog, 397-399**

**messages, sending/receiving (proximity), 660-661**

**MessageWebSocket, 579**

**metadata**

associating with audio, video, 359

reading

decoding images, 335-339

with metadata query language, 338-339

writing, Image element, 343-344

**metadata query language, reading (raw metadata), 338-339**

**Microsoft Local Language Portal, 35**

**MinHeight, 64**

**minimum width, choosing, 85-86**

**MinWidth, 64**

**MonitoredStates, 656**

**MouseCapabilities, mouse input, 141-142**

**MouseDevice, 141**

**mouse input, 141**

GestureRecognizer, 143

MouseCapabilities, 141-142

MouseDevice, 141

pointer events, 142-143

PointerPointProperties, 142-143

UIElement gesture events, 144

**MouseWheelDelta, 142**

**ms-appx, 318**

**Multilingual App Toolkit, 32**

**multithreading, 181-183**

**MusicProperties, 367**

## N

**named elements, templates, 528**

**namespaces, XAML, 45-47**

**naming XAML elements, 57**

**narrow mode, 86**

**Navigate, 191**

**navigating**

- customizing view of collections, 550

- between pages, 189-191

- pages

- basic navigation and passing data, 190-192

- forward and back, 192-193

- Frame, 195-197

- NavigationHelper, 194-195

- page caching, 194

**navigation, WebView, 414-416**

**NavigationCompleted, 415**

**NavigationHelper, 194-195**

**NeighboringFilesQuery property, 622**

**.NET, XAML, 55**

**network capabilities, Capabilities tab (package manifests), 19**

**networking**

- background transfers, 576-579

- connection info, 581-582

- data, 572

- HTTP requests, 572-576

- sockets, 579

- syndication, 579-581

**NetworkStatusChanged, 581**

**NFC tags, 660**

**nine-grid feature, Image element (stretching), 319-322**

**Nonzero, 431**

**notifications**

- local notifications, 690

- push notifications, 711-712

- client-side code, 717-719

- Push Notification Wizard, 712-713

- server-side code, 713-715

- toast notifications. *See* toast notifications, 710

**NotRotated, 652**

**not running apps, 161**

## O

**Object, 61**

**object elements, 44**

**OneTime binding, 533**

**OneWay binding, 533**

**OnFileOpenPickerActivated, 625**

**OnLaunched method, 26**

**OnSuspending method, 26**

**OnWindowCreated, 172, 189**

**OpticalMarginAlignment, 285-288**

**order of property and event processing, 45**

**orientation, 11**

- discovering, 86-87

- sensors, 652

**Orientation, 145**

**Orientation property, 101**

**OrientationSensor, 652**

**output, 65**

**output reports, custom HID devices, 683**

**overscroll effect, 109**

## P

**packaged files, app data, 561**

**package manifest, 10-11**

- Application tab, 11

**package manifests**

- Capabilities tab, 16-17

- device capabilities, 18

- file capabilities, 17-18

- identity capabilities, 19

- network capabilities, 19

- Content URIs tab, 20

- Declarations tab, 19-20

- Packaging tab, 20
- Requirements tab, 16
- Visual Assets tab, 11
  - customizing logo images, 13-16
  - customizing splash screen, 11-12
- Packaging tab, package manifests, 20**
- Padding, 66-67**
- page bar, 253**
- page caching, 194**
- page rendering, customizing (Image element), 350-353**
- pages**
  - navigating
    - basic navigation and passing data, 190-192
    - forward and back, 192-193
    - Frame, 195-197
    - NavigationHelper, 194-195
    - page caching, 194
  - navigating between, 189-191
- panels, 63, 84, 88**
  - Canvas, 88-89
  - Grid, 91-99
  - StackPanel, 90-91
  - VariableSizedWrapGrid, 99-103
- panes, keyboard shortcuts, 611**
- PaneThemeTransition, 461-462**
- panning, ScrollViewer, 140**
- parent elements, 63**
- parsing XAML at runtime, 56-57**
- PasswordBox, 311-312**
- passwords, 313**
- Path, 425-426**
- PathFigure, 429-430**
- PathGeometry, 428, 433**
- PathIcon, 239**
- PathSegment, 429-431**
- pausing games, 165**
- paying per click, 200**
- paying per impression, 200**
- PC ads, adding to pages, 202-203**
- PC projects, creating apps, 8-9**
- PDFPage, 350**
- PDF pages, rendering (Image element), 348-350**
- peek medium templates, 692**
- peek wide templates, 696**
- peer devices, proximity, 661-662**
- pen input, 144-145**
  - handwriting recognition, 150-153
  - rendering handwriting, 146-149
- PenLineCap, 427**
- pens**
  - PointerMoved events, 145
  - RightTapped, 145
- percentage sizing, 97**
- performance, cached composition, 451**
- Performance and Diagnostics page, 10**
- performance scalability, vector graphics, 451**
- performing scans, image scanners, 665-666**
- perspective transforms, 79**
- phone ads, adding, 204**
- PhoneticName property, 704**
- Photo Loop, 375**
- photos, capturing**
  - CameraCaptureUI, 368-370
  - CaptureElement, 374
- pictures, Account Picture Provider (contracts), 615-616**
- pipe, 681**
- PixelBuffer, 325**
- pixel data**
  - decoding images, 331-333
  - writing, Image element, 341-342
- pixels, 66**
- PlaceholderText property, 266**
- PlaneProjection, 80-81**
- playback, 356-357**
  - MediaElement
    - adding effects, 361
    - customizing, 358-359
    - markers, 360-361
    - media content, 357-358
    - MediaPlayer, 363-364
    - playing custom media formats, 365-367
    - states and events, 359-360
    - using as a media player, 362
- PlaybackRate, 359**

- Play command, 604-605
- playing custom media formats, 365-367
- Playlist class, 363
- Point, 655
- PointCaptures, 126
- PointerCanceled, 121
- PointerCaptureLost, 121
- Pointer class, 119
- PointerDevice class, 118-119
- PointerDownThemeAnimation, 470
- PointerEntered, 121
- pointer events, 120-122
  - mouse input, 142-143
- PointerExited, 121
- PointerMoved, 121
- PointerMoved events, pens, 145
- PointerPoint class, 119-120
- PointerPointProperties, mouse input, 142-143
- PointerPointProperties class, 120
- PointerPressed, 121-122
- PointerReleased, 121-122
- pointers, 118
  - capturing pointers, 123-126
  - hit testing, 126-127
  - Pointer class, 119
  - PointerDevice class, 118-119
  - pointer events, 120-122
  - PointerPoint class, 119-120
  - PointerPointProperties class, 120
  - tracking multiple pointers, 127-130
- PointerUpThemeAnimation, 470
- point of sale devices. *See* barcode scanners
- points, halting bubbling, 122-123
- Polygon, 425
- Polyline, 424-425
- PopInThemeAnimation, 469
- PopOutThemeAnimation, 469
- Popup, 401-403
- popup controls, 397
  - MessageDialog, 397-399
  - Popup, 401-403
  - PopupMenu, 399-401
- PopupMenu, 399-401
- PopupThemeTransition, 459
- portrait, 652
- portrait-flipped, 652
- position, layout, 68
  - Alignment, 69
  - content alignment, 70
  - FlowDirection, 71
- Position, 655
- PositionSource, 655
- power easing functions, 491-493
- PrepareFileTranscodeAsync, 379
- Pressure, 145
- previews
  - CaptureElement, 371-373
  - image scanners, 667-668
- PreviousExecutionState, 169
  - launching apps, 169-170
- primary XML namespace, 47
- Print command, 596-601
  - adding custom options, 603
  - changing default options, 601-602
  - configuring displayed options, 602
- PrintDocument, 596-598
- PrintManager, 596
- PrintTaskRequested, 599
- processing rules, XAML (child object elements), 55-56
- ProductLicenses, 211
- ProductListings, 210
- ProgressBar, 388
- ProgressRing, 411-412
- Project command, 606
- projections, 79
- properties
  - attached properties, 88-89
  - Background property, 127
  - binding to plain properties, 532
  - CancelCommandIndex property, 399
  - ClockIdentifier property, 411
  - content property, child object elements, 52
  - Content property, 48

- DefaultSectionIndex property, 405
- hijacking for new purposes, 517
- HorizontalChildrenAlignment, 101
- IsEditable property, 267
- ItemHeight property, 101
- ItemWidth property, 101
- MaximumRowsOrColumns, 101
- NeighboringFilesQuery property, 622
- Orientation property, 101
- PhoneticName property, 704
- PlaceholderText property, 266
- RenderTransform property, 72
- respecting, 514-516
- SelectedItem, 268
- SelectionMode, 267
- Stretch property, 111
- VerticalChildrenAlignment, 101
- VideoSettings property, 371
- property, order of, 45**
- property elements, XAML, 47-48**
- property paths, custom animations, 481-483**
- proportional sizing, 97**
- ProtocolActivatedEventArgs.Uri, 616**
- Protocol contracts, 623**
- providing suggestions, SearchBox, 391-392**
- proximity, 659-660**
  - peer devices, 661-662
  - sending/receiving messages, 660-661
- Pseudo Language, 32-33**
- pubCenter, 200**
  - defining ad units, 200-202
- publication, 581**
- pull updates, live tiles, 701**
- purchasing dialog, launching, 208-209**
- push notifications, 711-712**
  - client-side code, 717-719
  - live tiles, 701
  - Push Notification Wizard, 712-713
  - server-side code, 713-715
- Push Notification Wizard, 712-713**

## Q

- quality, transcoding (MediaTranscoder), 378-379**
- query suggestions, SearchBox, 393-394**
- quick links, 595**

## R

- radial gradients, 443**
- RadioButton, 245-246**
- RadiusX, 423**
- range controls, 387**
  - ProgressBar, 388
  - SearchBox, 390-391
    - history suggestions, 392
    - local content suggestions, 395
    - providing suggestions, 391-392
    - query suggestions, 393-394
    - result suggestions, 395-396
  - Slider, 389-390
- reading**
  - BitmapProperties from decoders, 337
  - ImageProperties, 335-336
  - metadata with metadata query language, 338-339
- reading metadata, decoding images, 335-339**
- receipts, validating from Windows Store, 218**
- receiving messages, proximity, 660-661**
- Rectangle, 422-423**
- RectangleGeometry, 428**
- red squiggles, 303**
- referencing resource files, 59**
- referencing files with URLs, Image element, 316-318**
- registering custom background tasks, 642-643**
- RelativeSource, Binding, 531**
- Remote Machine option, 8**
- RemoveFile, 626**

**rendering, 538**

- data templates, 538-542
- handwriting, 146-149
- high-performance rendering, ListView and GridView, 550
- value converters, 543-546

**rendering PDF content as an image, Image element, 347-348**

- customizing page rendering, 350-353
- rendering a PDF page, 348-350

**RenderSize, 65****RenderTargetBitmap, generating dynamic images, 325****RenderToStreamAsync, 349****RenderTransform, 422****RenderTransformOrigin, 72****RenderTransform property, 72****reordering items, ListView, 272****ReorderThemeTransition, 466****RepeatBehavior, 479-480****RepeatButton, 242****RepositionThemeAnimation, 470****RepositionThemeTransition, 464-465****representing session state, 173-174****RequestProductPurchaseAsync, 213****RequestUnconstrainedDownloadsAsync, 579****Requirements tab, package manifests, 16****resource, 316****ResourceDictionary, 508****resource files, referencing, 59****resource lookup, styles, 508-509****resource packages, leveraging (Image element), 329-330****resource qualifiers, 328****respecting**

- properties, 514-516
- target control's properties, templates, 511-516

**responding to clicked toast notifications, 708****responding to changes, visual states, 520-522****restricting what is typed in text boxes, 307****result suggestions, SearchBox, 395-396****result TextBlock, 25****resuming apps, 166****retained-mode graphics, 451****RetrieveFeedAsync, 580****retrieving data**

- barcode scanners, 670-671
- magnetic stripe readers, 672-673

**RfcommDeviceService, 676****RichEditBox, 309-311****RichTextBlock, 296-297**

- embedding UIElements, 297-298
- spelling and text predictions, 302-303
- TextBox, 301-302
- text overflow, 298-300

**RightTapped, 131**

- pens, 145

**RightToLeft, 91**

- VariableSizedWrapGrid, 103

**roaming files**

- app data, 561
- app files, 562

**roaming settings, app data (app settings), 559****root object elements, XAML, 45****Rotated90DegreesCounterclockwise, 652****Rotated180DegreesCounterclockwise, 652****Rotated270DegreesCounterclockwise, 652****RotateTransform, 73-74****rotation, orientation, 11****Round line cap, 426****routed events, 120****RowDefinition, 94****rows, sizing in Grid, 96-97****RowSpan, 94****RunAsync method, 182****running apps, 161****S****SatelliteData, 655****ScaleTransform, 74-76, 111, 474****ScaleX, 74****ScaleY, 74**

**scaling, 67**

- content overflow, 111-114
- Image element, 326
  - loading file variations automatically, 326-327
  - loading file variations manually, 329
- logos, 14-16
- Viewbox, 112

**ScanData, 671****ScanDataLabel, 671****ScanDataType, 671****scheduled option, showing toast notifications, 710****ScheduledToastNotification, 711****scheduled updates, live tiles, 700****screen readers, 37****screens, improving screen reading, 36-37****screensavers, turning off during media, 363****ScriptNotify, 418****ScrollBar, 106****scrolling**

- content overflow, 106-111
- snap points, 109-111

**scrolling placeholders, 551****ScrollIntoView, ListView, 270****ScrollView, 106**

- customizing, 106-107
- panning and zooming, 140
- SemanticZoom, 279
- zooming, 114

**SearchBox, 390-391**

- history suggestions, 392
- local content suggestions, 395
- providing suggestions, 391-392
- query suggestions, 393-394
- result suggestions, 395-396

**Search command, 584**

- customizing Search Pane, 586-588
- programmatically showing Search Pane, 588-589
- Search Contract, 584-586

**Search Contract, Search command, 584-586****Search Pane**

- customizing, 586-588
- showing, 588-589

**secondary tiles, 703-705**

- lock screen, 721

**security checks, 40****SelectedIndex, 259****SelectedItem, 259****SelectedItems, 268****SelectedValue, 259****selection, ListView, 270-271****SelectionChanged, 267****SelectionMode, 267, 626****Selector, 259-260****Self, RelativeSource, 531****SemanticZoom, 276-279**

- ScrollView, 279

**sending messages, proximity, 660-661****sensors, 647**

- accelerometer, 647-649
  - shake detection, 650
  - tossing motion, 649-650
- compass, 651
- gyrometer, 651
- inclinometer, 651
- light sensor, 651
- location, 652
  - current location, 655-656
  - geofencing, 656-659
- orientation, 652
- proximity, 659-660
  - peer devices, 661-662
  - sending/receiving messages, 660-661

**server-side code, push notifications, 713-715****session state**

- representing, 173-174
- SuspensionManager, 173

**SetAccountPictureAsync, 616****SetAccountPictureFromStreamAsync, 616****SetDefaultCharacterFormat, 310****SetPixelData, 342****SetPreviewPage, 600**

- SetPropertiesAsync**, 347
- SetSource**, 325
- Settings command**, 606
  - handling links with SettingsPane, 608-610
  - showing links, 606-608
- SettingsFlyout**, 611
- SettingsIdentifier**, 626
- SettingsPane**, handling links, 608-610
- shake detection**, accelerometer, 650
- Shape**, 421-422
  - Ellipse, 423
  - Line, 423-424
  - Path, 425-426
  - Polygon, 425
  - Polyline, 424-425
  - Rectangle, 422-423
  - strokes, 426-428
- Share command**, 589
  - being a share source, 589-592
  - being a share target, 592-595
- Share contracts**, 614
- ShareOperation**, 595
- share source**, Share command, 589-592
- share target**, Share command, 592-595
- sharing data**, 592
- sharing source with DataContext**, Binding, 533-534
- Show**, 609
- ShowAsync**, 183
- ShowError**, 388
- ShowIndependent**, 610
- showing**
  - Search pane, 588-589
  - toast notifications, 710-711
- showing/hiding software keyboards**, 308
- showing links**, Settings command, 606-608
- showing previews**, CaptureElement, 371-373
- ShowPaused**, 388
- ShowScrollingPlaceholders**, 551
- shrinking**, 105
- sideloaded apps**, 220
- SimpleOrientationSensor**, 652
- Simulator option**, 8
- simulators**, 8
- SineEase**, 494
- SingleUse**, 657
- size**, layout
  - height, 64-65
  - Margin, 66-67
  - Padding, 66-67
  - width, 64-65
- SizeChanged event handler**, 84
- sizing**
  - absolute sizing, 96
  - autosizing, 97
  - percentage sizing, 97
  - proportional sizing, 97
  - rows and columns in Grid, 96-97
- SkewTransform**, 76
- slashes (/)**, 318
- Slider**, 389-390
- snap points**, 109
  - scrolling, 110
- sockets**, networking, 579
- software keyboard**, TextBox, 304-306
- software keyboards**, 308
- SolidColorBrush**, 436-437
- source**, Appointments Provider Contract, 635-636
- source property**, Binding, 530
- SourceRoot**, 351
- sources**
  - Contact Contract, 631-632
  - Contact Picker, 628-629
  - image scanners, 666-667
- Sources**, 316
- spaces in Geometry strings**, 435
- Span**, 292-293
- SpatialReferenceld**, 655
- special folders**, user data, 570-571
- speech synthesis**, 383-384
- Speech Synthesis Markup Language (SSML)**, 384-385
- SpeechSynthesizer**, 384
- Speed**, 655
- SpeedRatio**, 478-479

spelling and text predictions, RichTextBlocks, 302-303  
 spelling errors, 303  
 SplashScreen, 169  
 splash screens, 29  
     customizing, 11-12  
 spline keyframes, 487-488  
 SplineXXXKeyframe, 487  
 SplitCloseThemeAnimation, 471  
 SplitOpenThemeAnimation, 471  
 Square 30x30 Logo, 13  
 Square 70x70 Logo, 13  
 Square 150x150 Logo, 13  
 SSML (Speech Synthesis Markup Language), 384-385  
 StackPanel, 90-91  
     child layout properties, 91  
 star sizing, 97  
 star syntax, 97  
 StartPeriodicUpdate, 701  
 StartTime, 657  
 states, playback (MediaElement), 359-360  
 static image-based wide templates, 694-696  
 static medium templates, 690  
 StaticResource, 508  
 static text-only wide templates, 693-694  
 static VisualStateManager, 523  
 StorageFile, 556  
 StorageFolder, 556, 571  
 Store Logo, 13  
 Storyboard, 466-468  
     tweaking with Timeline, 484-485  
 storyboards with multiple animations, 480-481  
 StreamSocket, 579  
 StreamWebSocket, 579  
 Stretch, 69  
 stretching with nine-grid feature, Image element, 319-322  
 Stretch property, 111  
 strings, Geometry, 433-435  
 Stroke, 423  
 StrokeDashArray, 426-427  
 StrokeDashCap, 426  
 StrokeEndLineCap, 426

StrokeLineJoin, 426  
 StrokeMiterLimit, 426  
 strokes, Shape, 426-428  
 StrokeStartLineCap, 426  
 Style, 502  
 style inheritance, 504-505  
 styles, 499-503  
     BaseTargetType, 503  
     implicit styles, 505  
     resource lookup, 508-509  
     setting templates int, 517-518  
     style inheritance, 504-505  
     theme resources, 506-508  
 stylus. *See pens*  
 submitting apps to Windows Store, 40-41  
 subscriptions, 200  
 suspended apps, 161  
 suspending apps  
     deferral, 166  
     lifecycle, 164-165  
     Suspending event, 165-166  
 Suspending event, suspending apps, 165-166  
 SuspensionManager, 163, 174-175  
     session state, 173  
 SVG-to-XAML converters, 435  
 SwipeBackThemeAnimation, 470  
 SwipeHintThemeAnimation, 470  
 SymbolIcon, 236-237, 237  
 syndication, networking, 579-581  
 SystemConditionType, 646  
 System.GPS.XXX, 338  
 System.Image.XXX, 338  
 SystemMediaTransportControls, background audio, 638-641  
 System.Photo.XXX, 338  
 System.XXX, 338

## T

tap and send, 659  
 Tapped, 131  
 target action, SemanticZoom, 276

**target control's properties, templates, 511-516**

**target property, Binding, 530**

**TargetProperty, 482-483, 487**

**targets**

Appointments Provider Contract, 636-637

Contact Contract, 633-635

Contact Picker, 630

**targetsize-XXX, 328**

**TargetType, implicit styles, 506**

**Task, 183**

**technical checks, 40**

**TemplateBindingBackground, 515**

**templates, 499, 509**

control templates, 510-511

tweaking, 518-519

data templates, rendering, 538-542

data versus control, 539

named elements, 528

setting inside styles, 517-518

target control's properties, 511-516

tile templates, 688

image-based large templates, 696

large tile templates, 696

medium tile templates, 690

peek medium templates, 692

peek wide templates, 696

static image-based wide templates, 694-696

static medium templates, 690

static text-only wide templates, 693-694

text-only large templates, 696

wide tile templates, 693

toast templates, 706-707

**template selectors, 543**

**temporary files, app files, 562**

**terminating MediaCapture, 373**

**terminating apps, 167**

**testing Windows Store features, 220-221**

automated testing, 224-225

CurrentAppSimulator, 222-223

**TextAlignment, 284**

**TextBlock, 37, 283**

text controls

customizing text rendering, 284-287

text content, 289-291

text elements, 291-294

text selection, 294-295

whitespace, 291

**TextBox, 301-302**

InputScope, 304-306

RichTextBlocks, 301-302

software keyboard, 304-306

spelling and prediction, 302-303

text selection, 304

**text content, TextBlock, 289-291**

**text controls, 283**

PasswordBox, 311-312

RichEditBox, 309-311

RichTextBlock, 296-297

embedding UIElements, 297-298

text overflow, 298-300

RichTextBlocks

spelling and text predictions, 302-303

TextBox, 301-302

RichTextBlock. *See* RichTextBlock

TextBlock, 283

customizing text rendering, 284-287

text content, 289-291

text elements, 291-294

text selection, 294-295

TextBox, 301-302

InputScope, 304-306

software keyboard, 304-306

spelling and prediction, 302-303

text selection, 304

text elements, 291-294

**text elements, TextBlock, 291-294**

**TextElements, 291**

**TextLineBounds, 284, 286-287**

**text-only large templates, 696-698**

**text overflow, RichTextBlock, 298-300**

**TextPointer, 295**

**TextReadingOrder, 285, 288**

**text rendering, customizing, 284-287**

- CharacterSpacing, 285-286
- IsColorFontEnabled, 288-289
- OpticalMarginAlignment, 287-288
- TextLineBounds, 286-287
- TextReadingOrder, 288

**text selection**

- TextBlock, 294-295
- TextBox, 304

**text-to-speech, 383-384****TextTrimming, 284****TextWrapping, 284**

- versus AcceptsReturn, 302

**theme animations, 466, 469-471**

- Duration property, 472
- Storyboard, 466-468
- tweaking, 471

**ThemeResource, 507-508****theme resources, styles, 506-508****theme transitions, 455-456**

- AddDeleteThemeTransition, 462-464
- applying to elements, 456-457
- ContentThemeTransition, 460
- EdgeUIThemeTransition, 460
- EntranceThemeTransition, 457-459
- PaneThemeTransition, 461-462
- PopupThemeTransition, 459
- ReorderThemeTransition, 466
- RepositionThemeTransition, 464-465

**themes, 228-229**

- default themes, 227

**Thickness, 66****third-party payment systems, 200****threading, 181-183****threading model, 181-182**

- awaiting synchronous operations, 182-184
- transitioning between threads, 185-186

**ThreadPoolTimer, 495****threads**

- ASTA threads, 182
- transitioning between, 185-186

**Thumbnail property, 542****thumbnails, 557****TickPlacement, 389****tile brushes, 443-444**

- ImageBrush, 444-445
- WebViewBrush, 446-450

**TileID, 169****tile templates, live tiles, 688-690**

- image-based large templates, 696
- large tile templates, 696
- medium tile templates, 690
- peek medium templates, 692
- peek wide templates, 696
- static image-based wide templates, 694-696
- static medium templates, 690
- static text-only wide templates, 693-694
- text-only large templates, 696
- wide tile templates, 693

**time-based trials, 206****Timeline**

- animation, 480
- tweaking Storyboard, 484-485

**TimePicker, 410-411****TimeSpan, 475****Timestamp, 655****Title, 626****To, custom animations, 475-477****toast notifications, 705**

- alarm apps, 708-710
- incoming call notifications, 708-710
- options for showing, 710-711
- responding to clicked, 708
- toast templates, 706-707

**toast templates, toast notifications, 706-707****ToggleButton, 243****ToggleMenuFlyoutItem, 280****ToggleSwitch, 412-413****ToolTip, 246-249****ToolTipService, 247****TopHeader, 109****tossing motion, accelerometer, 649-650****touch events, 118**

**touch input, 118**

- gestures, 130-136
  - EdgeGesture, 135-136
  - GestureRecognizer, 131-134
  - UIElement gesture events, 136
- manipulations, 136-137
  - inertia, 141
  - manipulation events, 137-140
- pointers, 118
  - capturing pointers, 123-126
  - hit testing, 126-127
  - Pointer class, 119
  - PointerDevice class, 118-119
  - pointer events, 120-122
  - PointerPoint class, 119-120
  - PointerPointProperties class, 120
  - tracking multiple pointers, 127-130
- points, halting bubbling, 122-123

**tracking multiple pointers, 127-130****transcoding**

- Image element, 344-347
- media, 378
- MediaTranscoder
  - adding effects, 383
  - changing format, 381-382
  - changing quality, 378-379
  - trimming, 382-383

**TransformGroup, 78****transitioning between threads, threading model, 185-186****transitioning from running to killing, lifecycle, 163-164****transitions**

- hidden managed-to-unmanaged code transitions, 324
- theme transitions. *See* theme transitions

**TranslateTransform, 77, 474****translating languages, 33****TriggerConnectState, 662****triggering animation, 468-469****triggers, custom background tasks, 644-645****trimming, MediaTranscoder, 382-383****troubleshooting FadeInThemeAnimation, 471****TrySetFileName, 628****tweaking**

- control templates, 518-519
- theme animation, 471

**tweaking animations, 478**

- AutoReverse, 479
- BeginTime, 478
- FillBehavior, 480
- RepeatBehavior, 479-480
- SpeedRatio, 478-479

**tweaking Storyboard with Timeline, custom animations, 484-485****Twist, 145****TwoWay binding, 533****type conversion, child object elements, 54-55****type converters, XAML, 49****U****UI Accessibility Checker, 36****UIElement, 61, 65****UIElement gesture events, 136**

- mouse input, 144

**UIElement keyboard events, keyboard input, 154-155****UIElements, RichTextBlock, 297-298****UI objects, 182****UI threads, 181****UI virtualization, 262****Update, 700****UpdateTile, 689****updating**

- live tiles, options for, 699-701
- XAML
  - application definition, 24-29
  - main page logic, 23-24
  - main page user interface, 21-22
- XAML code, 21

**URIs**

- launching apps, programmatically, 176-177
- referencing files, Image element, 316-318

**USB devices, custom USB devices. *See* custom USB devices**

**UserConsentVerifier**, 664

**user data**, 555, 563

file picker, 564-567

folder picker, 567-570

libraries and special folders, 570-571

**UserInformation**, 616

**user themes**, 228-229

## V

**validating Windows Store receipts**, 218-220

**value converters**, 546

rendering, 543-546

**VariableSizedWrapGrid**, 99-103

child layout properties, 103

RightToLeft, 103

**vector graphics**, 421, 451

Brushes. *See* Brushes

Geometry, 428-429

FillRule, 431

GeometryGroup, 432-433

PathFigure, 429

PathSegment, 429

representing as strings, 433-435

performance scalability, 451

Shape, 421-422

Ellipse, 423

Line, 423-424

Path, 425-426

Polygon, 425

Polyline, 424-425

Rectangle, 422-423

strokes, 426-428

**VerticalAlignment**, 69

**VerticalChildrenAlignment**, 101

**VerticalScrollMode**, 109

**video**, capturing

CameraCaptureUI, 370-371

MediaCapture, 376

**VideoEncodingQuality**, 380

**VideoSettings property**, 371

**video stabilization**, 361

**Viewbox**, 111-113

**view model**, 540

**views**, 186

**VirtualKey**, 157

**VirtualKey.Menu**, 155

**Visual Assets tab**, package manifests, 11

customizing logo images, 13-16

customizing splash screen, 11-12

**VisualStateGroup**, 523

**Visual State Manager**, 519-521

**VisualStateManager**, 519-521

**visual states**, 499, 519-520

responding to changes, 520-522

visual transitions, 523-527

**Visual Studio**, support for XAML and code-behind, 57

**visual transitions**, visual states, 523-527

**VSM (Visual State Manager)**, 519-521

## W

**Wait method**, 184

**WebAuthenticationBroker**, 313

**WebView**, 413

capturing web content, 418

composing HTML with XAML, 416

composing JavaScript with C#, 417

navigation, 414-416

**WebViewBrush**, 446-450

**whitespace**, TextBlock, 291

**Wide 310x150 Logo**, 13

**wide tile templates**, 693

**width**, 64-65

**Wi-Fi**, custom Wi-Fi Direct devices, 684-686

**Window.Current.Bounds**, 85

**windows**, displaying multiple, 186-189

**Windows.ApplicationModel.Store**, 221

**Windows.Graphics.Display.DisplayProperties**, 87

**window size and location, layout, 84-85**

- choosing minimum width, 85-86
- discovering orientation, 86-87

**Windows Store**

- certification process, 40-41
- submitting to, 40-41
- testing features, 220-221
  - automated testing, 224-225
  - CurrentAppSimulator, 222-223
- validating receipts, 218-220

**wizards, Push Notification Wizard, 712-713****WrapGrid, 263****WritableBitmap, generating dynamic images, 322-325****writing**

- ImageProperties, 335-336
- metadata, Image element, 343-344
- pixel data, Image element, 341-342

**X****XAML, 43-44**

- attributes, 44-45
- C# and, 44-45
  - loading and parsing XAML at runtime, 56-57
  - naming XAML elements, 57
  - support for, 57-58
- child object elements. *See* child object elements
- elements, 44-45
- keywords, 59-60
- markup extensions, 49-51
- namespaces, 45-47
- .NET, 55
- property elements, 47-48
- root object elements, 45
- type converters, 49
- updating
  - application definition, 24-29
  - main page logic, 23-24
  - main page user interface, 21-22

**XAML code, updating, 21****XAML controls, 236****XamlTune, 435****XAML UI Responsiveness, 10****x:Boolean, 60****x:Class, 60****x:Double, 60****x:FieldModifier, 60****X:Int32, 60****x:Key, 60****XLIFF files, 32-33****XML, 10****xml:lang, 59****XML namespace, markup compatibility, 47****xml:space, 59****x:Name, 57, 60****x:Null, 60****x:StaticResource, 60****x:String, 60****x:Subclass, 60****x:TemplateBinding, 60****x:ThemeResource, 60****XTilt, 145****x:Uid, 60****Y****YTilt, 145****Z****ZIndex value, 90****ZoomFactor, 114****zooming, ScrollViewer, 114, 140****ZoomSnapPoints, 114****Z order, 90**

*This page intentionally left blank*

# UNLEASHED

**Unleashed** takes you beyond the basics, providing an exhaustive, technically sophisticated reference for professionals who need to exploit a technology to its fullest potential. It's the best resource for practical advice from the experts, and the most in-depth coverage of the latest technologies.

[informat.com/unleashed](http://informat.com/unleashed)

## OTHER UNLEASHED TITLES

### **C# 5.0 Unleashed**

ISBN-13: 9780672336904

### **ASP.NET Dynamic Data Unleashed**

ISBN-13: 9780672335655

### **Microsoft System Center 2012 Unleashed**

ISBN-13: 9780672336126

### **System Center 2012 Configuration Manager (SCCM) Unleashed**

ISBN-13: 9780672334375

### **Windows Server 2012 Unleashed**

ISBN-13: 9780672336225

### **Microsoft Exchange Server 2013 Unleashed**

ISBN-13: 9780672336119

### **Windows 8.1 Apps with HTML5 and JavaScript Unleashed**

ISBN-13: 9780672337116

### **Microsoft Visual Studio 2012 Unleashed**

ISBN-13: 9780672336256

### **System Center 2012 Operations Manager Unleashed**

ISBN-13: 9780672335914

### **Microsoft Dynamics CRM 2013 Unleashed**

ISBN-13: 9780672337031

### **Microsoft Lync Server 2013 Unleashed**

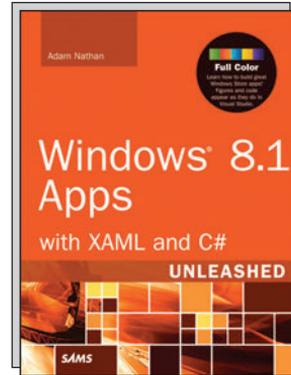
ISBN-13: 9780672336157

### **Visual Basic 2012 Unleashed**

ISBN-13: 9780672336317

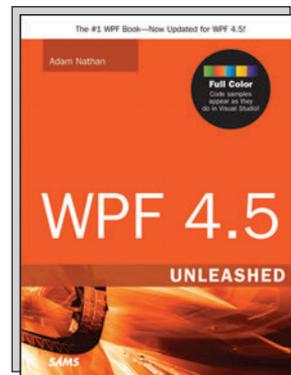
### **Microsoft Visual Studio LightSwitch Unleashed**

ISBN-13: 9780672335532



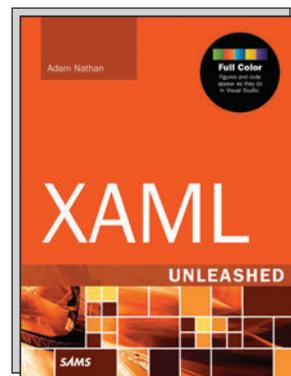
### **Windows 8.1 Apps with XAML and C# Unleashed**

ISBN-13: 9780672337086



### **WPF 4.5 Unleashed**

ISBN-13: 9780672336973



### **XAML Unleashed**

ISBN-13: 9780672337222

**SAMS**

[informat.com/sams](http://informat.com/sams)

# SAMS

# REGISTER



## THIS PRODUCT

[informit.com/register](http://informit.com/register)

Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to **[informit.com/register](http://informit.com/register)** to sign in or create an account.

You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:

- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

### **About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE**

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you're looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.

## **informIT.com**

THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison-Wesley | Cisco Press | Exam Cram  
IBM Press | Que | Prentice Hall | Sams

SAFARI BOOKS ONLINE

**InformIT** is a brand of Pearson and the online presence for the world's leading technology publishers. It's your source for reliable and qualified content and knowledge, providing access to the top brands, authors, and contributors from the tech community.

## LearnIT at InformIT

Looking for a book, eBook, or training video on a new technology? Seeking timely and relevant information and tutorials? Looking for expert opinions, advice, and tips? **InformIT has the solution.**

- Learn about new releases and special promotions by subscribing to a wide variety of newsletters. Visit **[informit.com/newsletters](http://informit.com/newsletters)**.
- Access FREE podcasts from experts at **[informit.com/podcasts](http://informit.com/podcasts)**.
- Read the latest author articles and sample chapters at **[informit.com/articles](http://informit.com/articles)**.
- Access thousands of books and videos in the Safari Books Online digital library at **[safari.informit.com](http://safari.informit.com)**.
- Get tips from expert blogs at **[informit.com/blogs](http://informit.com/blogs)**.

Visit **[informit.com/learn](http://informit.com/learn)** to discover all the ways you can access the hottest technology content.

### Are You Part of the IT Crowd?

Connect with Pearson authors and editors via RSS feeds, Facebook, Twitter, YouTube, and more! Visit **[informit.com/socialconnect](http://informit.com/socialconnect)**.

