

Forewords by **Chris Anderson** and **Rey Bango**



Building Windows 8 Apps *with* **JavaScript**

Windows
Development
Series

Chris **Sells** and Brandon **Satrom**
with Don Box

 **telerik**

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Praise for Building Windows 8 Apps with JavaScript

“This is going to be the Windows 8 app book YOU MUST have in your library! It’s well written and expertly covers every aspect of how to build an HTML/JS app for Windows 8!”

—*Jonathan Antoine, Infinite Square and Microsoft MVP*

“Great introduction to app development for Windows 8. After so many years in the XAML space, this book made me want to consider the JavaScript/HTML route.”

—*Shawn Wildermuth, Microsoft MVP (Data), author, trainer, and speaker, www.wilderminds.com*

“What you hold in your hands right now is an excellent walkthrough of how to build, ship, and profit from building apps using HTML and JavaScript for Windows 8. While I’ve been working on Windows 8 for the last two years, I can honestly say that I have learned about new parts of the platform from this book and can’t wait to build an app that uses them.”

—*From the Foreword by Chris Anderson, Distinguished Engineer, Windows Libraries for JavaScript, Microsoft Corp.*

“Chris and Brandon have gone to the heart of Windows 8 programming and produced a clear, concise, and easily understood tutorial that should be on every Windows 8 programmer’s bookshelf. If you are programming Windows 8 with HTML and JavaScript, this is *the* book you need.”

—*Jesse Liberty, Windows 8 technical evangelist, Telerik*

“I feel that this book will be the must-read reference for anyone who is dedicated to building a great Windows 8 app, and will be the book by which all others are compared. Yes, that’s a pretty bold statement, but considering that both of the authors have been deeply involved in Windows 8 app development for a LONG time, especially while they were at Microsoft, I feel confident in that statement.”

—*From the Foreword by Rey Bango, Developer Relations, Microsoft Corp.*

“This is easily the most well-written book on building Windows 8 apps with JavaScript that I have read. It has been an invaluable resource for helping me to transfer my experience with building large JavaScript applications and thick-client applications into a Windows 8 environment. Chris and Brandon do a masterful job of explaining that this is just HTML, JavaScript, and CSS, while at the same time distilling all of the intricate details and subtleties of running web technologies in a native Windows application environment, with the full power of WinRT and the JavaScript extensions for it.”

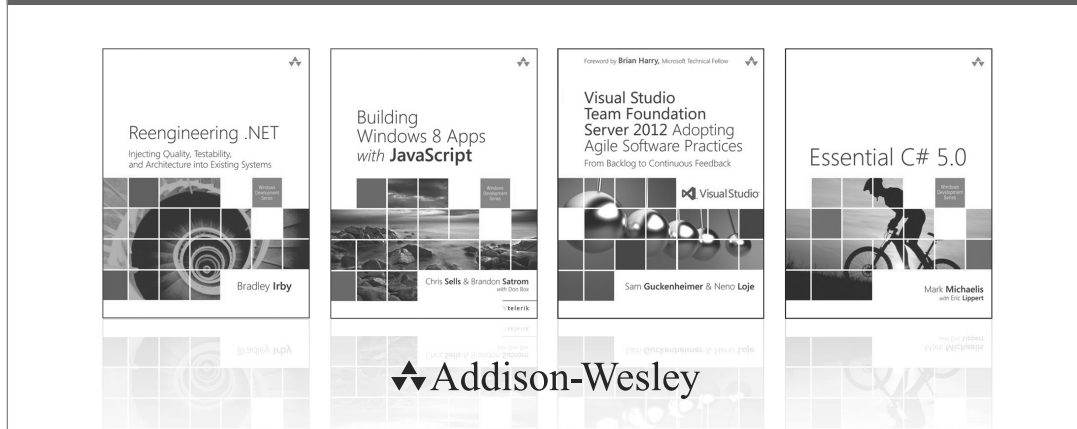
—*Derick Bailey, independent consultant, screencaster, speaker, and author, <http://mutedsolutions.com> and <http://watchmecode.net>*

“Chris and Brandon do a truly excellent job explaining how to create great Windows 8 applications. While reading this book I learned things about the platform.”

—*Josh Williams, Principal Development Lead for WinJS, Microsoft Corp.*

Building Windows 8 Apps with JavaScript

Microsoft Windows Development Series



Visit informit.com/mswinseries for a complete list of available publications.

The Windows Development Series grew out of the award-winning Microsoft .NET Development Series established in 2002 to provide professional developers with the most comprehensive and practical coverage of the latest Windows developer technologies. The original series has been expanded to include not just .NET, but all major Windows platform technologies and tools. It is supported and developed by the leaders and experts of Microsoft development technologies, including Microsoft architects, MVPs and RDs, and leading industry luminaries. Titles and resources in this series provide a core resource of information and understanding every developer needs to write effective applications for Windows and related Microsoft developer technologies.

“This is a great resource for developers targeting Microsoft platforms. It covers all bases, from expert perspective to reference and how-to. Books in this series are essential reading for those who want to judiciously expand their knowledge and expertise.”

– JOHN MONTGOMERY, Principal Director of Program Management, Microsoft

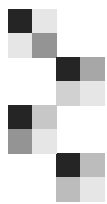
“This series is always where I go first for the best way to get up to speed on new technologies. With its expanded charter to go beyond .NET into the entire Windows platform, this series just keeps getting better and more relevant to the modern Windows developer.”

– CHRIS SELLS, Vice President, Developer Tools Division, Telerik



Make sure to connect with us!
informit.com/socialconnect





Building Windows 8 Apps with JavaScript

- **Chris Sells**
- **Brandon Satrom**
with Don Box

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The .NET logo is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries and is used under license from Microsoft.

Microsoft, Windows, Visual Basic, Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries/regions.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2012953216

Copyright © 2013 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-86128-3

ISBN-10: 0-321-86128-0

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, December 2012

Chris would like to dedicate this book to his mbelle.



*Brandon would like to dedicate this book to his three constants:
Sarah, Benjamin, and Jack.*



This page intentionally left blank



Contents

Foreword by Chris Anderson xvii

Foreword by Rey Bango xix

Preface xxi

Acknowledgments xxvii

About the Authors xxxiii

1 Hello, Windows 8! 1

Your First Windows Store App 2

Getting Started in Visual Studio 2012 6

Controls, Binding, and Styling in Blend 16

Navigation 24

Networking in WinJS and WinRT 29

Split App Template 34

The Rest 40

Where Are We? 40

2 Binding and Controls 41

Binding 41

Binding Objects 42

Initializers 51

Binding List 53

Sorting and Filtering 55

Grouping 58

Templates 60

Controls 63

HTML Elements 63

WinRT Controls 64

WinJS Controls 66

Custom Controls 70

Where Are We? 78

3 Layout 79

Layouts: Taming the Device Matrix 79

Windows 8: Consumer Choice without the Tyranny of Devices 81

Layouts in Windows 8 81

Working with Screen Sizes 84

Orientations 93

View States 95

Using CSS Layout Capabilities to Adapt Your App 99

The CSS3 Grid Layout Specification 100

Adaptive Layouts for Application Content 103

Creating Adaptive UIs with CSS and WinJS 104

Using the CSS Flexbox for Adaptive UIs 104

Using CSS Multi-Column Layout for Adaptive Content 107

Creating Adaptive Collections with the ListView 111

Responding to Layout Changes in JavaScript 113

Where Are We? 116

4 Typography 119

Typography in Windows Store Apps 119

Segoe UI 120

Cambria 122

Calibri 123

CSS3 Web Fonts 124

Using CSS to Tweak Your Typography 129

Working with Platform Iconography 136

Using and Manipulating Icon Fonts in a Windows Store App 147

Where Are We? 153

5 Media 155

- Working with Audio and Video 155
- Getting Started with Media in Windows 8 156
- Styling Media and Creating Custom Controls 159
- Adding Subtitles to Video 163
- Adding Video Effects 167
- Working with Audio in Windows Store Apps 170
 - Creating Background Audio* 171
- Working with User Media Libraries via a File Picker 175
- Selecting Multiple Files 180
- Other File Picker Types 182
- Working with Captured Media 185
- Making Your App Connectable with Play To 189
- Where Are We? 192

6 Drawing and Animation 193

- HTML5 Graphics with SVG and Canvas 193
 - Introducing SVG* 194
 - Introducing Canvas* 199
 - Choosing between Canvas and SVG* 204
- Manipulating Pixels 206
 - Pixel Manipulation with Canvas* 206
 - Pixel Manipulation with `Windows.Graphics.Imaging`* 209
- Animation in Windows Store Apps 212
 - Animations in Windows Store Apps: Fast and Fluid* 213
 - Transforming and Animating with CSS* 213
 - Working with the WinJS Animation Library* 220
- Where Are We? 224

7 App State 225

- Settings 226
 - The Settings Charm* 228
- Lifetime 238
 - WinJS Lifetime Event Helpers* 241
 - Sessions* 242
 - Debugging Sessions* 246
 - WinJS Session Helpers* 250

Files	252
<i>WinJS File Helpers</i>	255
Libraries	256
<i>File Activation</i>	259
<i>File Pickers</i>	261
Where Are We?	266

8 Networking 267

Network Capabilities	267
Mobile Networking	269
XMLHttpRequest	273
<i>Parsing XML Results</i>	274
<i>Progress and Errors</i>	274
<i>Parsing JSON Results</i>	275
Syndication	277
Background Data Transfer	280
Web Content	284
<i>HTML Content</i>	285
<i>iframe Hosting</i>	286
<i>The Web Context</i>	287
Where Are We?	292

9 Shell Contracts 293

The Windows 8 Shell	294
Contracts	295
Search Contract	297
<i>Implementing Search</i>	298
<i>Search Suggestions</i>	303
Share Contract	305
<i>Share Target</i>	310
<i>Accessing Shared Data</i>	316
<i>Reporting Sharing Progress</i>	321
Contacts Contract	322
<i>Contact Picker</i>	322
<i>Contact Providers</i>	325
Debugging Contract Providers	331
Where Are We?	332

10 Shell Integration 333

Live Tiles 333

Your App's Tile 335*Tile Updates* 335*Small and Large Tile Updates* 338*Tile Images* 340*Tile Peeking* 340*Scheduled Tile Updates* 342*Secondary Tiles* 343

Badges 348

Background Tasks 350

Triggering a Background Task 351*Creating a Background Task* 352*Lock-Screen Apps* 355*Avoiding Task Duplication* 357

Toast Notifications 358

App Activation from Toast 361*Scheduled Toast* 362

Where Are We? 363

11 Device Interaction 365

An Introduction to Touch 366

Touch-Friendly HTML Controls 367*Touch-Friendly WinJS Controls* 369*Building Touch-Friendly Apps with Screen Edges* 370*Creating Touch-Friendly Interactions with SemanticZoom* 374*Supporting Mouse and Keyboard Interactions* 379

Working with Device Capabilities 380

Declaring Device Capabilities 380*Working with Recording Devices* 381*Adding In-App Print Capabilities* 384

Working with Location Data 387

Using the Geolocator Object 387*Watching for Location Changes* 390*Using Location Data with Bing Maps* 391*Simulating Location Information* 393

Working with Sensors	394
<i>Working with the Light Sensor</i>	396
<i>Working with the Accelerometer</i>	398
<i>Working with the Compass</i>	400
<i>Working with the Simple Orientation Sensor</i>	402
<i>Working with Other Sensors</i>	403
Where Are We?	403

12 Native Extensibility 405

Multiple Languages, One App	406
Getting Started	407
WinRT and the JavaScript Environment	411
WinRT Classes	413
<i>Classes and Methods</i>	414
<i>Methods and Exceptions</i>	416
<i>Classes and Properties</i>	420
WinRT Objects	421
<i>Objects and Handles</i>	422
WinRT Types in C++/CX and JavaScript	424
<i>Strings</i>	429
<i>Arrays</i>	431
<i>WinRT Value Types</i>	433
Delegates and Functions	435
<i>C++11 Lambdas</i>	436
<i>Creating WinRT Delegates from C++11 Lambdas</i>	439
Events	440
Concurrency and Asynchrony	443
Where Are We?	451

13 Making Money 453

Preparing for Submission	454
<i>Setting Up a Developer Account</i>	454
<i>Reserving Your App Name</i>	454
<i>Preparing Your App for Local Testing</i>	457
<i>Running the Windows App Certification Kit (WACK)</i>	458



Submitting Your App to the Windows Store	463
<i>Completing the Windows Store Submission Process</i>	464
<i>The Certification Waiting Game</i>	471
<i>Dealing with Rejection</i>	473
<i>Submitting an Update</i>	474
Working with Ads	476
<i>Rules for Ads in Windows 8–Style Apps</i>	476
<i>Working with the Windows 8 Ads SDK</i>	477
<i>Working with Media-Based Ads</i>	477
<i>Working with Text-Based Ads</i>	480
Enabling Trial Mode in Your App	483
<i>Introducing the Windows Store API and Simulator</i>	483
<i>Simulating and Testing Trial Functionality</i>	485
Working with In-App Purchases	488
<i>Creating In-App Purchase Functionality</i>	489
<i>Defining In-App Offers in the Windows Store Submission Process</i>	494
Design for Monetization	495
Marketing and Managing Your App	496
<i>Tracking Your App from the Windows Store Dashboard</i>	496
<i>Getting Your App Featured in the Windows Store</i>	496
<i>Getting Paid</i>	498
Where Are We?	499
A JavaScript for C-Family Programmers	501
Hello, World	502
<i>Separation of Concerns</i>	503
<i>Using the id As an object</i>	505
<i>WinJS Activation</i>	505
Values and Types	507
Operators	508
Objects	510
Dates	511
Regular Expressions	511
Arrays	512



Object Prototypes (“Classes”)	514
<i>Constructors</i>	514
<i>Prototypes</i>	515
<i>Prototypical Inheritance</i>	518
<i>Static Members</i>	519
<i>Class Definitions via WinJS</i>	519
Functions	520
<i>Function Arguments</i>	522
<i>Call and Bind</i>	523
<i>Closures</i>	525
Debug Output	525
Scoping	526
<i>Hoisting</i>	526
<i>Modules</i>	527
<i>Namespaces</i>	528
<i>WinJS Namespaces</i>	528
Strict	529
Serialization	531
B Presentation and Style at a Glance	533
Using HTML for App Content and Structure	534
<i>What Is HTML?</i>	534
<i>What’s New in HTML5?</i>	535
Using CSS for App Layout and Style	541
<i>What Is CSS?</i>	541
<i>Where Should I Define My CSS?</i>	553
<i>How CSS Rules Cascade</i>	555
CSS in Windows Store Apps	558
<i>Overriding Default Windows Store App Styles</i>	560
Index	565



Foreword by Chris Anderson

Windows 8 represents a significant change in the Microsoft developer ecosystem. While the consumer-oriented changes in Windows 8 get much of the spotlight (new user experience, touch, tablet computers, etc.), there is a tectonic shift at the core of Windows 8. In Windows 8, developers are presented with a choice of programming environments to build their craft; DirectX, XAML, and HTML. Beyond this, there is now a built-in platform for monetizing their products.

For the past two years I have been working as a development lead and architect for the Windows Library for JavaScript, or WinJS. When we first thought about having HTML and JavaScript be a first-class platform for building native Windows applications, one of the biggest challenges we faced was how to balance the standards-based world of HTML/JS with the native platform features of Windows. With the advent of the Windows Runtime (WinRT) we had the technical tool to simply integrate new features into the web platform, but there was a constant tension about where to stick to the standards and where to innovate.

Our mantra on the WinJS team was to “code to the standard,” and we used WinRT functionality in the implementation of WinJS very sparingly. We felt that it was better to let the app developers decide how much platform dependency they should take.



JavaScript is also a world of heterogeneous toolkits. People often blend together jQuery, require, Modernizer, and Backbone to accomplish their job. When building WinJS we attempted to build a suite of toolkits, which can be mixed and matched with other existing toolkits. You can trivially use Knockout as your binding solution in the WinJS `Listview`; you can plug jQuery UI controls easily into the WinJS declarative control processing. There are places where we built large toolkits (`Listview` being the most obvious example), and others where we have very small toolkits (e.g., the CommonJS Promise/A implementation contained in `base.js`).

As we built WinJS we continually adjusted our design patterns to try to mesh more seamlessly into the existing conventions of the JavaScript community. I remember one of our earlier patterns was to attempt to freeze and seal the prototype definitions for many of our constructor functions. We quickly got feedback that wasn't how JavaScript developers did it. We switched to mutable prototypes and have learned the power of being able to monkey-patch definitions at runtime.

I had the privilege of working with Chris Sells for several years at Microsoft. I never got to work with Brandon Satrom, but judging from the quality of this book I'm guessing he has the same passion for developers, the love of programming, and the art of writing that Chris Sells has.

What you hold in your hands right now is an excellent walk-through of how to build, ship, and profit from building apps using HTML and JavaScript for Windows 8. While I've been working on Windows 8 for the last two years, I can honestly say that I have learned about new parts of the platform from this book and can't wait to build an app that uses them.

I hope you enjoy reading it, and happy coding!

—Chris Anderson

Distinguished Engineer, Windows Libraries for JavaScript, Microsoft Corp.

August 2012



Foreword by Rey Bango

There's never been a better time to be a developer. The Internet has opened up more avenues of opportunities for programmers than at any time I can recall in my 20+ years in information technology. I cringe (with some fondness) at how learning a new technology, especially a programming language, was a fairly arduous task involving terse manuals, glowing monochrome screens, and CRTs as big as some small TVs. But the Internet has changed all of that, allowing us to learn anywhere we want via numerous types of connected devices and have a wealth of information presented to us in the blink of an eye.

And while developers are reaping the benefits of this, the ones truly benefitting are the typical, everyday computer users who no longer need to rely on the White or Yellow Pages to find a local plumber, or a dedicated GPS to get directions, or a DVD player to watch a great movie. The Internet has allowed us all to stay better connected today than at any point in history and it's been made possible by the ingenuity of us: developers.

As consumers become savvier, so do their needs and demands. They want information faster, and they want it in a clear and concise fashion across their devices so that they don't have to relearn how to do things every time they pick up a phone or switch on a computer.



That's the beauty of Windows 8. It aims to bring uniformity and cohesion across all experiences so that consumers can enjoy themselves, easily finding the information they want in a consistent and fluid fashion. And the best part of this is the opportunity for developers to help build those great experiences through the Windows 8 app ecosystem.

Windows 8 users are already accustomed to being connected, and apps play an important role in providing unique experiences for the information and services they depend on. With an install base of several hundred million Windows users, Windows 8 app developers are in an amazing position to not only enhance the experience for consumers but also to leverage a new platform that offers the technological and monetization frameworks to build a successful and profitable business. And with Microsoft including the ability to use JavaScript, HTML, and CSS to build these apps, it opens the platform to savvy web developers who also want to jump on this opportunity.

I've had the pleasure of working with both Chris Sells and Brandon Satrom here at Microsoft. I was very flattered when they asked if I'd write a foreword for their book and can honestly say that I accepted without hesitation. I knew that they would produce something that would bring immediate value to developers who are serious about leveraging the Windows 8 platform. Having spent time reviewing the manuscript, I feel that this book will be the must-read reference for anyone who is dedicated to building a great Windows 8 app and will be the book by which all others are compared. Yes, that's a pretty bold statement, but considering that both of the authors have been deeply involved in Windows 8 app development for a LONG time, especially while they were at Microsoft, I feel confident in that statement.

—Rey Bango

Developer Relations, Microsoft Corp.

October 2012



Preface

There is a certain beauty in man-made things. In some ways, that beauty cannot match nature. In other ways, that beauty is unparalleled because it exhibits the best qualities men and women are able to achieve with their own hands.

Art and architecture are oft-cited examples of man-made beauty, and we recognize this, conveying social and financial reward upon those creations that best exhibit the creative spirit—or even boundless will—of humanity.

But there is a simpler, yet more pervasive proof in the beauty of man-made things: that is the beauty of comfort and familiarity, of feeling like you belong, no matter where you may find yourself. This beauty is all the more meaningful because it speaks to the deep-seated social needs we all possess. It often feels less essential because it is so subtle, and also because we're only acutely aware of this need when it's not being met.

Imagine yourself in a foreign airport or the transit station of an unfamiliar city for the first time. Now suppose that you just stepped off a train or plane with a limited amount of time to get from your gate to the next one. Where do you go? How do you find information?

If you have a firsthand memory of such a place and experience, recalling that memory might even evoke a physical response: dilated pupils, an increased heart rate, clammy hands, and shortness of breath. Stress. Anxiety.



The unfamiliar evokes a need for familiarity, so what do you do? If you are traveling with friends or family, your stress is lessened, but the task remains. What do you do?

You look for familiarity—for anchoring clues: signs, numbers, letters, and text. Anything to help you find your way. A sign with the text “B Gates” and an arrow leading in a specific direction may provide instant relief. A bank of monitors might do the same. Perhaps all you require is the appearance of a stick figure pointing to the closest restroom. Whatever it may be, you would look for, and gravitate to, anything familiar that helps you accomplish the most important task at hand. And once found, those familiar things would anchor you, and provide comfort.

There is beauty in this! When numbers and letters and symbols can anchor us to a deeper reality and point us home, that is sublime art that nothing else can equal.

This, then, is *metro*—creating experiences that anchor us in reality, even in the face of the unfamiliar and the artificial. Further, these experiences do more than simply try to transpose and replicate our comfort from one medium to another. I don’t need the subway signs in Chicago to look exactly like those in Manhattan or Munich. As long as there is just enough present to evoke familiarity, I am comfortable. Better still, I don’t need the sign for the men’s restroom in Beijing to be a life-like photograph of a six-foot-tall, white male. To paraphrase Scott McCloud,¹ iconography is powerful not just because it is abstract, but because its abstract nature makes it identifiable, and we connect best with that which we identify. A stick figure is sufficient because I see myself therein, and that provides familiarity and comfort.

Metro is not Windows. Or Windows Phone. Or Xbox. It is not live tiles, black backgrounds, Segoe UI, or boxes with straight corners. It’s not HTML5, CSS, or JavaScript. Metro is not even Microsoft. It can live in the browser or in the desktop. It can even live in iOS or Android, because it was never really about a platform at all.

1. Published in 1994, McCloud’s masterwork *Understanding Comics* is just as much about the art and science of visual communication as it is about comics specifically. Buy and read this book now; you’ll thank us for it.

In a time when the metro design language is increasingly being used to instruct developers to find and delete every border-radius rule in their CSS, to follow a design checklist, or even to capture a design in a series of boxes, it's important that we remember that metro—like every great design idea ever conceived—is about building something that is beautiful for others. It is about delivering something that anchors us in reality and helps us find our way. It's about creating something that is beautiful because it's useful and comfortable.

You can't code your way to metro, except perhaps by accident. Even then, what you create will likely seem more artificial than authentic.

You can't even design your way to metro; no tutorial, checklist, or book will deliver a "metro experience" simply because you added colorful tiles, fancy page-flip effects, or a digital representation of a tabletop calendar.

All software is and has always been about the beneficiary of our work, and metro is no different. Never has there been a checklist or process to unlock what is most beneficial for every case because the real value lies in the process of discovery. Once you discover what the person using your application needs, it is up to you to discover how to best meet that need. With the results of this discovery, metro is about placing comfort and familiarity on an even footing with utility.

So, learn metro. Read the design guides and use the checklists. Watch the videos and think more like a designer, no matter what you are building. Before any of that, though, think about comfort and familiarity, and how your application or site can deliver those basic human needs better than any other.

That's what metro really means.

Some Terminology

During the development of Windows 8, the names of features and technologies have changed, so I thought I'd provide an up-to-date guide as of the writing of this book (after the RTM but before the General Availability).

- **Metro and Metro style:** The design language that describes the UI and experience of using Windows 8, Windows Phone 7, Windows



Phone 8, and the latest Xbox dashboard is called *metro*. For a while, that term was used to describe the new kinds of Windows 8 apps that are building in this design language—that is, “metro style apps” (no hyphen). Because of a large grocery store chain in Germany, that’s no longer the case.

- **Windows Store apps:** The replacement term from Microsoft for “metro style apps” is “Windows Store apps.” This refers to the fact that Windows 8 apps in the new style—that is, not the desktop apps that we’ve had since Windows 95—are deployed to consumers via Microsoft’s new Windows Store. However, this isn’t a very accurate term, since enterprise apps built in the Windows Store style aren’t deployed via the Windows Store at all. Oh well.
- **WinRT and WinRT apps:** The Windows Runtime (WinRT) is the core on which all Windows Store apps are built. However, Microsoft also uses the name to refer to the ARM version of Windows 8 and the tablets on which it runs—for example, the WinRT Surface refers to the ARM version of Microsoft’s Surface tablet. This would have been a good, accurate name for Windows 8 apps in the “new” style, but alas, it was not meant to be.
- **Modern apps:** Another name you sometimes hear for “Windows Store apps” is “modern apps,” which is just a slap in the face to anyone building apps of any other kind, including Windows 8 apps that run on the desktop. Hopefully this term won’t stick.

In this book, we mostly use the term “Windows Store app.”

What This Book Is For

The goal of this book is to give you a broad look at the range of capabilities you have in building your Windows Store app. It is not an exhaustive reference, but rather a survey of the tools, libraries, concepts, and techniques you need to go from starting a new app, to adding the features you want it to have, to shipping it into the Windows Store and making money.



Throughout the book we provide links to online resources we recommend you use for more details, but the big ones are these two:

<http://design.windows.com>

<http://dev.windows.com>

These two web sites are for the design and development of Windows Store apps. Further, the design web site is where you'll learn all about the metro design language, although you're unlikely to see that name on the site itself.

Who This Book Is For

This book is for web developers of all kinds—jQuery, PHP, ASP.NET, Rails, and so forth—who want to understand how to bring their web knowledge to the Windows 8 platform to build first-class applications.

This book is for designers who want to gain an understanding of how Windows Store apps are built from web technologies.

This book is for .NET, Win32, MFC, or Visual Basic developers who want to know how the next generation of Windows programs will be written using web technologies.

This book is not for developers who don't already have programming experience. A grasp of the basics of HTML, JavaScript, and CSS is going to help you greatly, but if you're brand new to these technologies, I recommend the appendixes at the end of the book, which are meant to provide a useful foundation of the web platform available to you in building Windows Store apps.

In short, this book is for anyone who's ever written a Windows program or written a web site and is interested in building Windows Store apps for Windows 8.

Sample Code and Errata

The sample code and any errata for this book can be found at <http://sellsbrothers.com/writing/win8jsbook>.

To run this book's sample code, you'll need the Windows 8 RTM, Visual Studio 2012 RTM, and Blend for Visual Studio 2012 (all of which are available at <http://dev.windows.com>).

How to Contact the Authors

Chris Sells is a Vice President for Telerik in the Developer Tools division. He can be reached at

E-mail: csells@sellsbrothers.com, Chris.Sells@telerik.com

Twitter: @csells

Blog: <http://sellsbrothers.com>

Brandon Satrom is a Program Manager for Telerik in the Kendo UI division. He can be reached at

E-mail: bsatrom@gmail.com, Brandon.Satrom@telerik.com

Twitter: @BrandonSatrom

Blog: <http://userinexperience.com>

Don Box is a Distinguished Engineer for Microsoft in the XBOX division. He can be reached at

E-mail: dbox@microsoft.com



Acknowledgments

From Chris Sells

This book has been a very long journey for me. I started it in September 2010, the same month I started on the Visual Studio 2012 team working on the end-to-end Windows 8 story for JavaScript programmers. I sat down the hall from Chris Anderson, Josh Williams, Chris Tavares, Jeff Fisher, David Owens, Vijaye Raji, Andy Sterland, Anson Horton, and Michael Booth, who comprised the bulk of the WinJS team and a big chunk of the Visual Studio 2012 team working on JavaScript for Win8 apps. I remember haggling over WinJS app models with Chris Anderson, building the first full-featured Win8/JavaScript templates with David Owens, re-working them with Michael Booth, and complaining to Josh Williams about the lack of a developer-friendly data source in WinJS. (Josh invented the binding list just to shut me up.) These guys taught me more about the web platform in 12 months than I'd learned in the previous 15 years of running my own web site.

It was Chris Anderson who wrote the initial outline of this book. The Windows division was putting together a “holiday build” of what was to become Windows 8. You see, most ‘softies ended up taking a portion of the month of December off because of the use-it-or-lose-it vacation policy. However, a large number of them are such type A+ personalities that they can’t actually take that much time off without going crazy, so they often



write programs for fun during that time. With that in mind, the holiday build of Win8 was to be prepared before the 1st of December as a stocking stuffer for bored employees from whom we wanted to gather feedback. For this to work, there needed to not only be the right conglomeration of bits but also documentation to get folks started. Mr. Anderson wrote the first draft of those docs, handed them to me, and said, “Finish this up, will you?”

What Chris had written was a 20-page document with sections like “Getting Started,” “Layout,” “Animation,” et cetera, giving a short intro for each topic. I took one look at that and thought to myself, “This is enough for an entire book.” And so, in two months, Kraig Brockschmidt (of *Inside OLE* fame) and I wrote the first book on Win8 for JavaScript programmers. The first chapter of this book was originally published in September 2011 at the first BUILD conference on msdn.com, titled “Create your first Windows Store app using JavaScript (Windows).”¹

I told you all of that so that I could tell you this: I need to thank John Montgomery for hiring me into the middle of the whole mess, and the WinJS and Visual Studio 2012 teams for taking me in, making me feel welcome, and letting me take part in the creation of an entirely new platform for programming Windows. That happens about once every decade or so, and it was an honor to participate. This book is the direct result of that experience, so you guys should consider this your book—I was only the scribe.

Or rather, I was only the scribe for my parts. Brandon wrote half of this book and I’m thankful he did. Except for the time when he informed me that he was in high school in 1995 when I wrote my first book (bastard!), he has been a joy to work with. He comes with all kinds of real-world, web-based JavaScript experience that I was lacking, so he influenced a lot of the thinking on my chapters.

Brandon and I didn’t write the whole book, however. My longtime friend and colleague, and the undisputed King of COM, Don Box, wrote Chapter 12, “Native Extensibility.” Ostensibly, that chapter is about extending your JavaScript apps using C++, but it’s really about the connective tissue between the two languages known as the Windows Runtime

1. <http://msdn.microsoft.com/en-US/library/windows/apps/br211385>
(<http://tinysells.com/282>).

(WinRT). The WinRT is the next version of Microsoft's venerable Component Object Model, so who better than the author of *Essential COM* to write that chapter? Hopefully it will inspire him to write *Essential WinRT* someday. Thanks, Don, not only for writing that chapter but also for dragging me along into the world of writing in the first place and for showing me how to do it with integrity.

I'd like to thank Michael Weinhardt from the bottom of my heart for what seems like a lifetime of coauthoring. Michael was the developmental editor on this project, which means that he regularly kicked all three of our asses to make sure that what we were saying actually made sense. Michael is also a longtime friend; one of the best parts of any writing project is always working with Michael, because I refuse to write without him.

I also need to thank the reviewers: Chris Anderson, Josh Williams, Jonathan Antoine, and Shawn Wildermuth. I'd especially like to thank Shawn for helping me with the research into Chapter 9, "Shell Contracts," and Chapter 10, "Shell Integration." My new gig keeps me very busy and he helped lay a lot of the foundation for those chapters. I couldn't have done those without you, Shawn.

I need to thank Joan Murray, my editor at Addison-Wesley. She's suffered through my tardiness on two books now, the aborted *Programming Data*, (I wrote my half—honest!), and now this one. Joan provides an effective mix of "soft" and "hard," with a bit of "grandmother guilt" thrown in. The fact that this book is published so near to the General Availability of Windows 8 is because Joan was "encouraging" me right along.

And finally, I need to thank my family for being so understanding when I had to steal time away from them to spend with this book. After 14 books (although lucky #13 never saw the light of day), I intend for this one to be my last. I've been doing this since the Sells brothers were born, using their names in my example programs because even when I was writing, I was also missing them. Of course, I have to thank my girlfriend, Michelle, for the home-cooked meals she brought while I was writing this book, and her son, Marcus, for lending his name to some of the last samples I wrote. It's been a great run and you've all been very supportive, but I'm not going to ask you to do so again. Oh, maybe I'll noodle on a novel in a cabin by the sea when I'm safely retired and I'm no longer working two jobs (the book

and my real job), but from now on, I'm going to let the next generation document the new technologies that come along.

OK, Mr. Anderson. I'm all finished.

From Brandon Satrom

I love history because history is the backstory that makes us care about the “now” of a narrative. This is true in fiction and in life. Not surprisingly, it's also true of the technical work you're about to read. So let me tell you a story...

In the spring of 2011, while I was still working for Microsoft, I got the itch to write a book. Not just any book, but one about HTML5, specifically the ubiquity of the web platform, and the potential for web technologies to be used beyond the browser and in desktops and devices. Little did I know at the time that the Windows team was in the midst of revolutionizing its platform for customers and developers alike. Fast-forward to September 2011 when, as a BUILD attendee, I was introduced to Windows 8, WinJS, and the WinRT. Over the course of the week I spent in Anaheim, I realized that I didn't just want to write an HTML5 book; I wanted to write a book about building HTML5 apps for Windows 8.

Chris and I actually met during that first BUILD, though it wasn't then that we teamed up to write this book. You see, Chris already had a coauthor, and I merely approached him after his excellent Day One keynote to say “hi,” congratulate him and the team, and ask for mentoring and advice on writing a book myself. He kindly agreed, and we chatted a few times as I began planning a solo Win8 book. (No matter what he says, I remain convinced that Chris does not remember meeting me at BUILD. After all, I'm just some lowly web developer, and he's Chris “COM” Sells!)

In another stroke of interesting coincidence, Chris and I both left Microsoft for Telerik less than one month apart, where we landed in different divisions of the company. It was at this time that Chris was on the hunt for a coauthor for the book you now hold in your hands. By chance, we reconnected, and Chris took me on as his coauthor. The rest, as they say, is infamy. Or something like that.

Now, the backstory of this book is coming to a close as we make our final revisions, pore over hundreds of pages to remove all mention of “Metro,”

retake screenshots for the fiftieth time, and recompile code samples for the hundredth time. And in this moment of reflection, I first want to thank my coauthor, Chris Sells. Chris took a chance on a no-name web developer whose only writing experience was a handful of articles for *MSDN* magazine and a bunch of unfinished fiction, and I am deeply grateful to him for the opportunity. I'm also grateful for his early honesty about the writing process. You were certainly right, Chris: It's not at all fun. And yet, somehow it still seems so worthwhile.

I'm humbled beyond belief to have my name adorn the cover of a book alongside none other than Don Box and Chris Sells. It feels simultaneously amazing and surreal. Somehow, I still feel that my name should be about half its current size and printed in transparent gray. All that's to say: It's been an honor to work alongside these two men and to soak up their many years of expertise.

Thanks as well to Michael Weinhardt, our developmental editor on this book. Michael and Chris were both wonderful mentors to me during the writing process, especially as I made the transition from magazine-length technical writing to book-length technical writing. My first chapter, Chapter 3, "Layout," was an early challenge for me, but Michael and Chris kept on pushing me to rewrite and revise until a story emerged. The process of peer review among the three of us was an amazing developmental experience for me, and I am grateful to Michael and Chris for pushing for my best work and for encouraging me when it poured forth.

I'm also grateful for the work of our reviewers: Chris Anderson, Josh Williams, Jonathan Antoine, and Shawn Wildermuth. Thanks especially to Jonathan, whose attention to detail, fact-checking, and ability to spot platform changes from one prerelease version of Win8 to the next saved my butt more than a few times. Thanks as well to our editor, Joan Murray. Thanks for putting up with us, and keeping us on track, Joan!

Finally, I want to thank my wife, Sarah, and my sons, Benjamin and Jack. Sarah, you are my partner, my friend, and the love of my life. Thank you for recognizing my gifts and for encouraging me to write. Thank you for supporting me as I worked on this book, for celebrating with me during the highs, and for helping any way you could during the lows. Benjamin and Jack, my sons, I love you both, and I am so blessed to be your father.



Thank you both for hugs and kisses when I needed them, as well as for the interruptions and breaks from writing that I needed even more.

To all three of you: Thanks for your patience and understanding during those times when writing meant time apart. It's time I owe you, and it's time I intend to pay back, with interest. Starting now.

From Don Box

Don would like to thank Deon Brewis, Martin Gudgin, Herb Sutter, Zach Johnson, and Logananth Seetharaman for their thoughtful feedback, and encourages Logan to not spend his five dollars in one place.



About the Authors

Chris Sells (@csells) is Vice President of the Developer Tools division at Telerik. He's written several books, including *Programming WPF*, *Windows Forms 2.0 Programming*, and *ATL Internals*. In his free time, Chris makes a pest of himself on Microsoft forums and mailing lists. More information about Chris and his various projects is available at www.sellsbrothers.com.

Brandon Satrom (@BrandonSatrom) is Program Manager for Kendo UI at Telerik and is based in Austin, Texas. A longtime web developer, Brandon loves to talk about HTML, JavaScript, CSS, open source, and whatever new, shiny tool or technology has distracted him from that other thing he was working on. Brandon speaks at events all around the world, and he loves hanging out with and learning from other passionate developers, both online and in person. He also loves writing and has had several articles featured in publications like *MSDN* magazine, *The Architecture Journal*, and *.net* magazine. Brandon can be reached online at his blog, www.UserInExperience.com.

Don Box, contributing author, is a Distinguished Engineer at Microsoft. At Microsoft, Don has worked on platform and developer technologies for .NET, SQL, and most recently, Xbox. Prior to Microsoft, Don roamed the Earth helping developers come to terms with COM, including writing *Essential COM* for Addison-Wesley.



This page intentionally left blank



1

Hello, Windows 8!

WINDOWS 8 brings together a number of ways to develop and think about developing apps. If you want to continue to build Windows desktop apps with WPF/Silverlight, Windows Forms, and/or DirectX, you are free to do so. Likewise, if you'd like to continue to build web sites using ASP.NET, HTML, and JavaScript, you're free to do that, too. Further, if you want to build touch-centric Windows Phone apps with Silverlight or XNA, that's OK.

However, in this book, we're focusing on how to build a new kind of app which is a hybrid of all three of these existing kinds of apps; this hybrid is called a Windows Store app. A Windows Store app is like a desktop app in that it's installed on your computer, unlike a web site. On the other hand, a Windows Store app is like a web site in that you can build it using HTML5, JavaScript, and CSS. However, instead of generating the UI on the server side, you'll see that the JavaScript framework for building Windows Store apps and the underlying Windows Runtime (WinRT) allows you to build apps with client-side state, offline storage, controls, templates and binding, along with a whole host of other services. Further, because Windows 8 is a tablet OS as well as a desktop OS, Windows Store apps are meant to be used via touch like Windows Phone apps as well as with the keyboard and mouse like traditional desktop apps. Of course, the big feature of Windows Store apps is that they can be submitted into the new Windows Store that is available front and center on the new Windows 8 Start screen.

In short, Windows Store apps are meant to work across different devices, taking maximum advantage of each and merging the best parts of desktop, web, and mobile apps into a single user and developer experience, all available from the Windows Store. In this chapter, we're going to dig into both the developer and the user experience, focusing on the former, of course, given that this is a programming book.

And because I like to start my programming books with a bit of programming, let's get right to it.

Your First Windows Store App

A Windows Store app built using HTML, JavaScript, and CSS starts with an HTML file:

```
<!DOCTYPE html>
<html>
<head><title>Hello, Metro/JS</title></head>
<body><h1>Hello and welcome to Windows Store apps for JavaScript!</h1>
</body>
</html>
```

This HTML, if it were loaded in the web browser, would result in the world's most boring web page. Further, a web page (or series of web pages, styles, code, resources, etc.) is not a Windows Store app. A Windows Store app includes these things but also includes the following metadata and resources to define the app for the Windows 8 Start screen:

- A **manifest file** to describe your app, including the name, description, start page, and so on
- A set of large and small **logo images** to be displayed on the Start Screen
- A **store logo** to be displayed by the Windows Store
- A **splash screen** to show when your app starts

The manifest file is an XML file called `appxmanifest.xml`, and a minimal one looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="http://schemas.microsoft.com/appx/2010/manifest">
  <Identity
    Name="a8c906d0-f878-4bd4-b727-5363ce0bfb52"
    Version="1.0.0.0"
    Publisher="CN=csells" />

  <Properties>
    <DisplayName>hello</DisplayName>
    <PublisherDisplayName>csells</PublisherDisplayName>
    <Logo>images\storelogo.png</Logo>
  </Properties>

  <Prerequisites>
    <OSMinVersion>6.2.1</OSMinVersion>
    <OSMaxVersionTested>6.2.1</OSMaxVersionTested>
  </Prerequisites>

  <Resources>
    <Resource Language="en-US" />
  </Resources>

  <Applications>
    <Application Id="App" StartPage="default.html">
      <VisualElements
        DisplayName="hello"
        Logo="images\logo.png"
        SmallLogo="images\smalllogo.png"
        Description="hello"
        ForegroundText="light"
        BackgroundColor="#000000">

        <SplashScreen Image="images\splashscreen.png" />

      </VisualElements>
    </Application>
  </Applications>
</Package>
```

The manifest¹ has things in it like the name and description, references to the logo images, and, most importantly, the name of the HTML file that represents the app's start page (`default.html` in this case).

1. You can read about the `appxmanifest.xml` file format here: <http://msdn.microsoft.com/library/windows/apps/br211474.aspx> (<http://tinysells.com/164>).

With the manifest and supporting files in place, the most basic way to get our super-exciting app registered with the system starts with PowerShell,² which you can access from the Start screen, and its `appx` module. The `appx` module in the Windows 8 PowerShell provides a number of commands that allow you to manage the Windows Store apps installed on your computer.³ The term *appx* is used by Microsoft to refer to packaged Windows Store apps, all of which have an `.appx` extension.⁴

If you're going to package and sign your app for submission to the Windows Store, you may decide to use the `MakeAppx.exe` and `SignTool.exe` command-line tools (which are part of the Windows Store developer tools), but to simply install an app on your own machine, the `Add-AppxPackage` PowerShell command from the `appx` module will do the trick, as Figure 1.1 illustrates.

After a successful execution of `Add-AppxPackage`, the `Get-AppxPackage` command will show you that it has been installed correctly, as Figure 1.1 also shows. Even more exciting, your app is now listed on the Start screen,⁵ as Figure 1.2 shows.

Besides our new hello tile, you'll notice that the Start screen shows tiles of different sizes with both static and dynamic information (I told Brandon that Portland wasn't cloudy *every* day!). For information about tiles, you'll want to read Chapter 10, "Shell Integration."

At this point, you're free to launch the app and see the "Hello and welcome to Windows Store apps for JavaScript!" inspirational message displayed (and which is too boring for a screen shot).

2. PowerShell is the next-gen command-line shell built into Windows.

3. You can see the complete list of commands in the `appx` module here: <http://technet.microsoft.com/en-us/library/hh856045.aspx> (<http://tinysells.com/158>).

4. An `appx` file is a file in the Open Packaging Conventions (OPC) format, which essentially means it's a `.zip` file with a few extras.

5. You can get to the Start screen by pressing the Windows key on your keyboard, by pressing `Ctrl+Esc`; by clicking in the lower left-hand corner of your screen; by moving your mouse to the upper right or lower right of your screen and clicking the Start button; by swiping in from the right-hand side of your screen using your finger and pressing the Start button; or by pressing `Win+C` and clicking on the Start button. Microsoft really wants you to be able to Start things.

```
Windows PowerShell
PS C:\ch01\01-hello> dir

Directory: C:\ch01\01-hello

Mode                LastWriteTime         Length Name
----                -
d----             9/17/2012   5:43 PM          images
-a---             9/17/2012   6:06 PM          1040 appxmanifest.xml
-a---             3/10/2012   6:34 PM          156 default.html

PS C:\ch01\01-hello> Add-AppxPackage .\appxmanifest.xml -register
PS C:\ch01\01-hello> Get-AppxPackage -name "a8c906d0-f878-4bd4-b727-5363ce0bfb52"

Name                : a8c906d0-f878-4bd4-b727-5363ce0bfb52
Publisher           : CN=cse11s
Architecture        : Neutral
ResourceId           :
Version             : 1.0.0.0
PackageFullName     : a8c906d0-f878-4bd4-b727-5363ce0bfb52_1.0.0.0_neutral__b3z51mp4vbmq4
InstallLocation     : C:\ch01\01-hello
IsFramework         : False
PackageFamilyName   : a8c906d0-f878-4bd4-b727-5363ce0bfb52_b3z51mp4vbmq4
PublisherId        : b3z51mp4vbmq4

PS C:\ch01\01-hello>
```

FIGURE 1.1: Adding an appx file and verifying that it's been added



FIGURE 1.2: Our sample app installed into the Start screen

A Windows Store app will always take up the screen space available to it—there are no overlapping Windows Store app windows. However, your app still needs to be able to run at multiple resolutions for different devices and for different “modes,” such as portrait, landscape, snapped, and filled, all of which you can read about in Chapter 3, “Layout.”

After seeing the minimal set of files, tools, and steps needed to build and install a Windows Store app, you’re probably already hoping for a tool to help you create, edit, package, launch, and debug your apps. For that, we’ve got Microsoft Visual Studio 2012.

Getting Started in Visual Studio 2012

Visual Studio is the premiere tool for Microsoft developers building apps for the Web and Windows, and has been for quite a while. It provides project management for keeping your app’s source files together; integrated build, deployment, and launching support; HTML, CSS, JavaScript, graphics, and Windows Store app manifest editing and debugging; and a whole lot more. There are several editions of Visual Studio, but we’ll use Microsoft Visual Studio 2012 Express for Windows 8 (a.k.a. VS), which is available for free⁶ and includes everything you need to build, package, and deploy your Windows Store apps.

To show you Visual Studio 2012 in action, we’re going to need something more interesting to build than an app with a static message (no matter how inspirational it may be). Developers new to any platform seem to have canonical apps that they build: Computer science students build text editors, compiler writers build Pascal compilers, web programmers build blogs, and, for some reason, mobile platform developers build news readers. So, let’s build ourselves a little Really Simple Syndication (RSS) Reader and start from my favorite template: the Navigation App (as Figure 1.3 shows us doing).

6. You can download Visual Studio 2012 for Windows 8, read the docs, browse the samples, and ask your questions here: <http://dev.windows.com>.

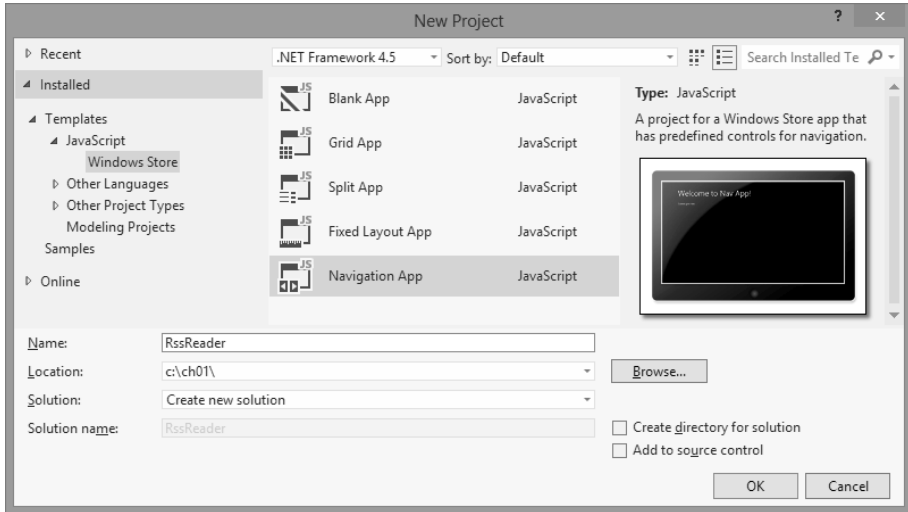


FIGURE 1.3: Creating a Windows Store Navigation App in Visual Studio 2012

The Windows Store app project templates provided with Visual Studio 2012 are as follows:

- **Blank App:** This is pretty much the smallest Windows Store app you can build with the correct manifest and graphics files that includes the Windows Library for JavaScript (a.k.a. WinJS). This is a good template for when you'd like to start from scratch and build up.
- **Grid App:** This is a simple but complete Windows Store app with three pages, navigation support, and the Windows 8 look and feel. This is a good template for starting with a full app that you'd like to modify.
- **Split App:** This is like the Grid App but with two different pages.
- **Fixed Layout App:** This is just like the Blank App template except that it allows you to build an app in a fixed-size area, like a casual game at 1024×768 , and let Windows scale it up or down for you, based on the available space.
- **Navigation App:** This template is the core of both the Grid and Split App templates, except with a single blank page instead of a set of fully functioning pages. This template gives you the navigation support you often want in your apps, but it also lets you build up largely from scratch.

Running the Navigation App template produces a Visual Studio 2012 Windows Store app project file for JavaScript (.jsproj) along with nearly the same set of files used to create our first sample, as Figure 1.4 shows.

The format and the contents of the `package.appxmanifest` file are the same as the `.appxmanifest.xml` file we've already seen, but the `.appxmanifest` extension allows the file to have a custom editor in Visual Studio 2012, as Figure 1.5 shows.

The manifest editor gives you a much easier way to edit the metadata associated with your app than getting all of the angle brackets right in the raw XML file.

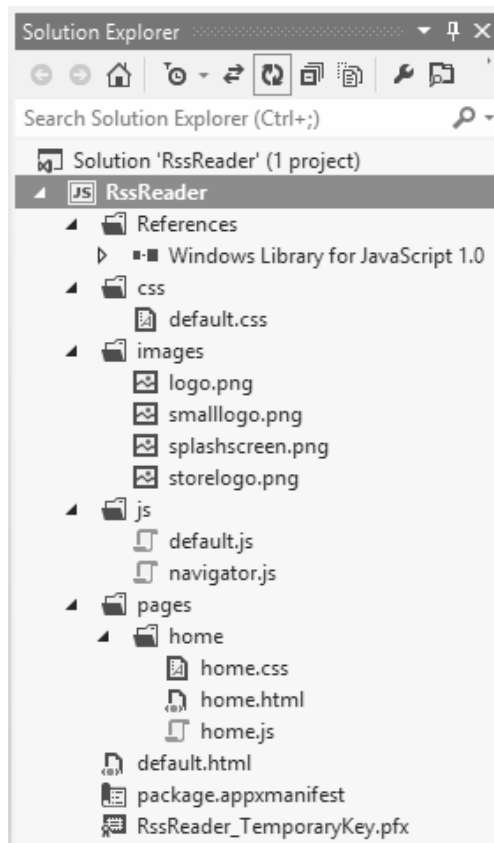


FIGURE 1.4: The files generated by the Visual Studio 2012 Windows Store Navigation App project template

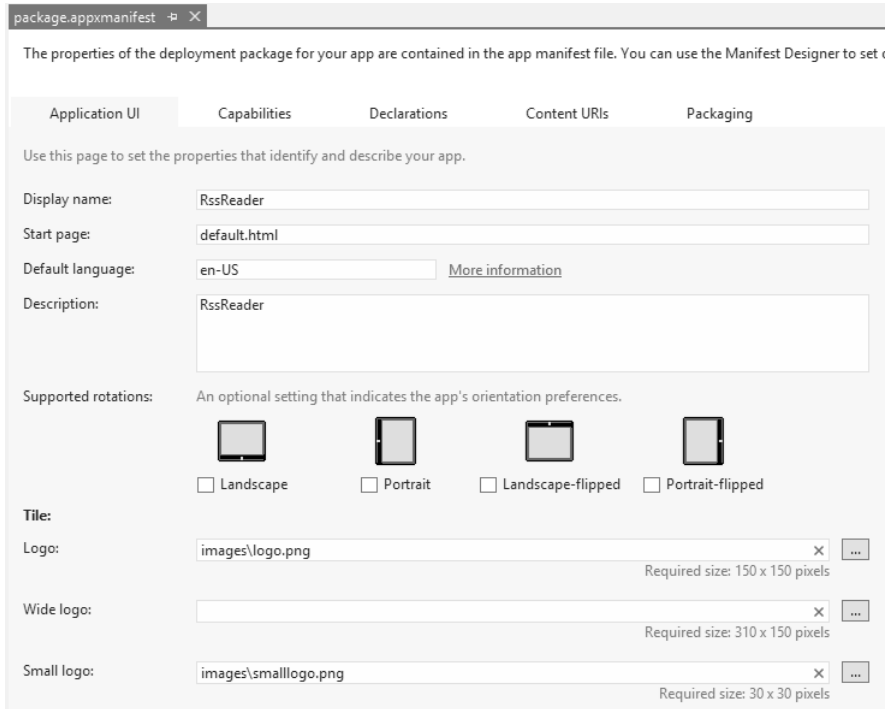


FIGURE 1.5: The Visual Studio 2012 Manifest Designer

The other interesting artifact added to the project is the Windows Library for JavaScript SDK reference. This brings in a reference to WinJS, a set of JS libraries produced by Microsoft to bring together the web platform; that is, HTML5, JavaScript, and CSS, with WinRT to make for a productive app framework for Windows Store apps built with JavaScript. You'll see a lot of both WinJS and WinRT in this book, but to get you started, take a look at the `default.html` file generated by the Navigation App template:

```
<!DOCTYPE html>
<!-- default.html -->
<html>
<head>
  <meta charset="utf-8" />
  <title>RssReader</title>
```

```
<!-- WinJS references -->
<link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
<script src="//Microsoft.WinJS.1.0/js/base.js"></script>
<script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

<!-- RssReader references -->
<link href="/css/default.css" rel="stylesheet" />
<script src="/js/default.js"></script>
<script src="/js/navigator.js"></script>
</head>
<body>
  <div id="contenthost" data-win-control="Application.PageControlNavigator"
    data-win-options="{home: '/pages/home/home.html'}"></div>
</body>
</html>
```

In the head section of the HTML, you'll notice the `link` and `script` elements that reference the styles and JS files that provide the functionality of WinJS. Part of that functionality is parsing the `data-win-control` and `data-win-options` attributes on the `contenthost` div toward the bottom of the file.

The `data-win-control` and `data-win-options`⁷ attributes enable declarative controls in Windows Store apps, essentially turning the HTML div element into an instance of a `PageControlNavigator` control from the `RssReader` namespace defined with this project. The `data-win-options` attribute is a simple JavaScript Object Notation (JSON) object passed to the control at runtime as constructor arguments. This declarative syntax allows programmers to easily lay out their controls using either the text editor built into Visual Studio 2012 or, as we'll soon see, using visual tools.

In the case of the `PageControlNavigator` control, what's happening is that the `default.html` file is really just a host for one or more logical pages that are loaded as your users navigate from one page to another. And, as you can see in the options for the control, the first page to be loaded is `homePage.html`, which the Navigation App template also generates:

7. The HTML5 specification leaves the `data-*` attributes as suggested library-specific and app-specific extensibility points that WinJS takes advantage of along with JavaScript libraries like Kendo UI, jQuery, and KnockoutJS.



```
<!DOCTYPE html>
<!-- homePage.html -->
<html>
<head>
  <meta charset="utf-8" />
  <title>homePage</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.1.0/css/ui-dark.css"
    rel="stylesheet" />
  <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

  <link href="/css/default.css" rel="stylesheet" />
  <link href="/pages/home/home.css" rel="stylesheet" />
  <script src="/pages/home/home.js"></script>
</head>
<body>
  <!-- The content that will be loaded and displayed. -->
  <div class="fragment homepage">
    <header aria-label="Header content" role="banner">
      <button class="win-backbutton" aria-label="Back" disabled
        type="button"></button>
      <h1 class="titlearea win-type-ellipsis">
        <span class="pagetitle">Welcome to RssReader!</span>
      </h1>
    </header>
    <section aria-label="Main content" role="main">
      <p>Content goes here.</p>
    </section>
  </div>
</body>
</html>
```

The HTML in `homePage.html` is a little bit more complicated than in `default.html` because it provides a Back button, a title, and a section making it pretty clear where Microsoft recommends that you put your content. In addition, the generated HTML pulls in the `homePage.js` file, which is where you put the logic that governs how the home page for your app is going to function. The generated skeleton code looks like this:

```
// home.js
(function () {
  "use strict";
```

```
WinJS.UI.Pages.define("/pages/home/home.html", {
    // This function is called whenever a user navigates to this page.
    // It populates the page elements with the app's data.
    ready: function (element, options) {
        // TODO: Initialize the page here.
    }
});
})();
```

The code inside `homePage.js` is wrapped in a self-executing, anonymous function, which is a JavaScript trick to keep everything in the function from leaking into global scope, providing the JavaScript equivalent of a private module. The "use strict" string is the JavaScript way of adding extra error checking at runtime, which is another good practice.⁸

Inside the module, the skeleton code provides a definition of a page control based on the `ready` function and the path to the HTML file associated with the page. A WinJS control is a reusable set of UI and behavior, whereas a page control is a control created around a logical page of HTML. The navigation support in the Windows Store app templates simply loads and unloads page controls as the user navigates between pages.

The `ready` event is fired when the page control is added to the HTML Document Object Model (DOM) and it's an excellent place for us to show a list of feeds for our RSS Reader:

```
// home.js
...
// define the feeds
window.feeds = [
    { title: "Brandon Satrom",
      url: "http://feeds.feedburner.com/userinexperience/tYGT" },
    { title: "Chris Sells",
      url: "http://sellsbrothers.com/posts/?format=rss" },
    { title: "Channel 9",
      url: "http://channel9.msdn.com/Feeds/RSS" },
];
```

8. Specifically, "use strict" is a feature of ECMAScript 5, which is the latest standard version of JavaScript (see <http://ecmascript.org/>). If you have a .NET background but are unfamiliar with the basics of JavaScript, I recommend that you read Appendix A, "JavaScript for C-family Programmers."



```
WinJS.UI.Pages.define("/pages/home/home.html", {
  ready: function (element, options) {
    // show the feeds
    var section = element.querySelector("section");
    section.innerHTML = "";

    feeds.forEach(function (feed) {
      var div = document.createElement("div");
      div.innerText = feed.title;
      section.appendChild(div);
    });
  }
});
...

```

The `ready` function is passed the `div` that presents the page in the HTML DOM via the `element` argument, so it's a good place to do a query for the `section` element to hold our list of feeds. The code inside the `ready` function is standard HTML DOM manipulation code using the global `feeds` data defined above the function.

Running the app provides a full-screen Windows Store app that looks like Figure 1.6.

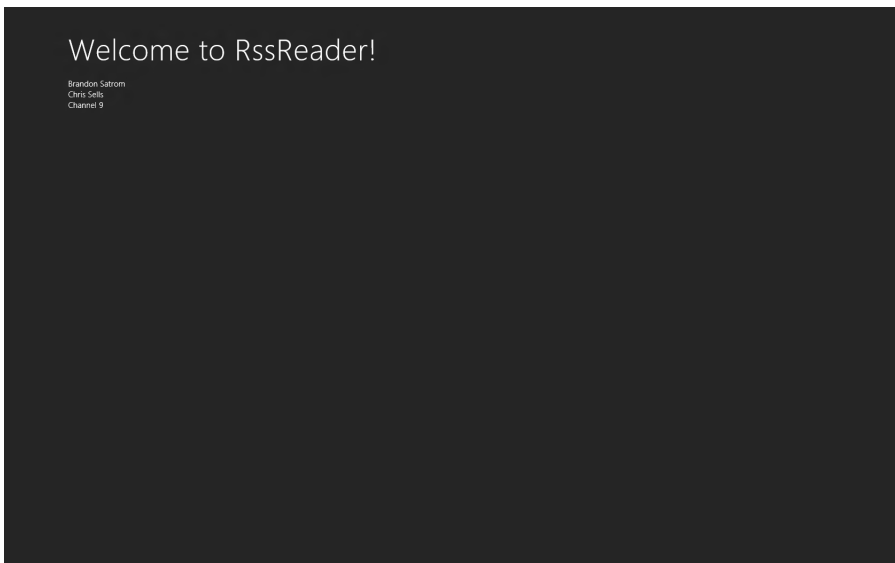


FIGURE 1.6: A list of feed titles in a Navigation App template project

If, in the process of developing this slightly functional app, you find yourself with issues, you can debug your app using Visual Studio 2012 by choosing **Debug | Start Debugging**, which gives you the following debugging tools:

- **Debugger:** Set breakpoints, use the various step debugger commands, and watch JavaScript data and behavior.
- **JavaScript Console:** Interact with JavaScript objects at a command line.
- **DOM Explorer:** Dig through the HTML DOM and see styles by element.
- **Call Stack:** Drill into the current JavaScript call stack.
- **Exceptions dialog:** Turn on the option to break when a JavaScript runtime exception is thrown.

In addition to debugging your app on the local machine (which is the default), you have two other options: remote machine and the simulator. You can change these options by choosing **Project | Properties** and selecting the debugger to launch, as Figure 1.7 shows.

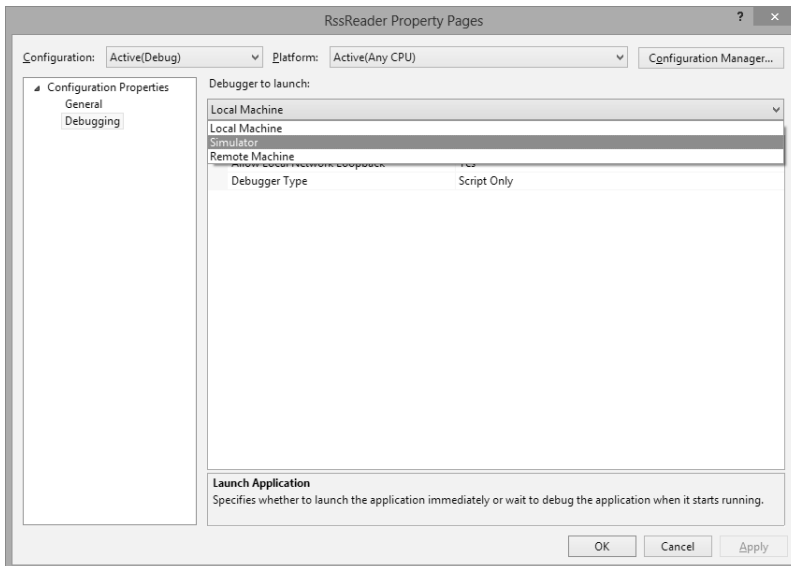


FIGURE 1.7: Choosing to debug against the local machine, the simulator, or a remote machine

The idea of remote machine debugging is that you can develop on a high-powered developer machine but debug on a more modest consumer-grade machine, like a tablet. This is handy to make sure your app works well on the type of machine you're targeting.

The simulator option, on the other hand, creates a remote desktop session back to the machine on which you're already running, providing a frame that lets you simulate various resolutions, landscape/portrait rotations, and touch, even if you're not using a touch-capable device. Figure 1.8 shows our sample app running in the simulator.

And, as if that weren't enough, Visual Studio 2012 is not the only tool you get when you install Visual Studio 2012 Express for Windows 8. If you'd like a WYSIWYG design experience for the visual portion of your app, you've got Microsoft Blend for Visual Studio 2012 (a.k.a. Blend).

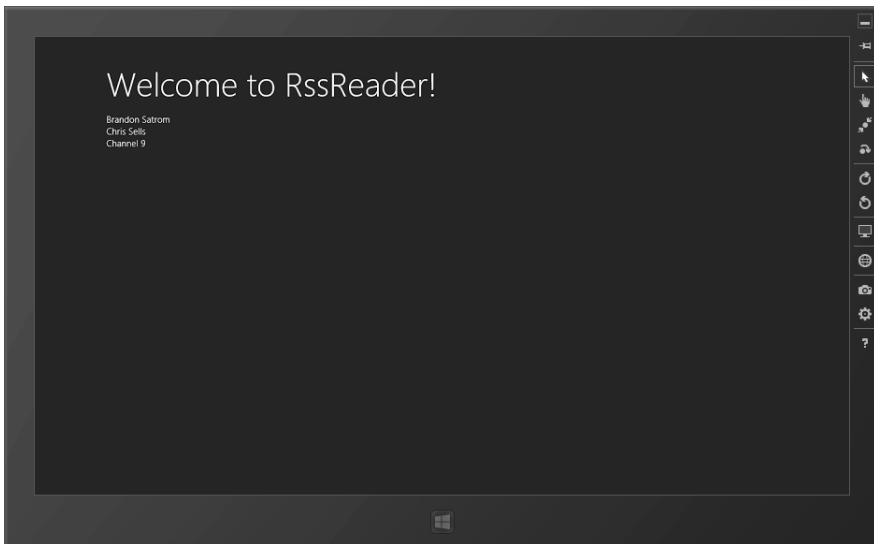


FIGURE 1.8: A Windows Store app running in the simulator

Controls, Binding, and Styling in Blend

Previous versions of Blend focused on the XAML developer. The Windows 8 version of Blend adds support for HTML to enable the design of Windows Store apps specifically with the following features:

- **Integration with Visual Studio:** You can load the same projects in both Visual Studio 2012 and Blend at the same time. In fact, you can load the project you're currently working on in VS by right-clicking on a project in the Solution Explorer and choosing Open in Blend.
- **Project Templates:** Visual Studio 2012 and Blend have the same set of project templates.
- **WYSIWYG Design for HTML:** Each page of your app is laid out as you'd see it when the app is running because Blend is actually running your app to display it accurately as you edit.
- **Interactive Mode:** You can throw a switch in Blend to run your app interactively as you navigate from page to page; then, when you get to a page you'd like to edit, you can flip the switch again and design the elements currently in view.⁹
- **HTML Tool Palettes:** The full set of controls and options are available from a tool palette and property editor.
- **Layout Simulator:** In the same way that VS provides a device simulator, Blend allows your app to be run and edited in one of several sizes and rotations.

Figure 1.9 shows Blend in action on our RSS Reader sample so far.

You'll notice in Figure 1.9 that even though we're inside Blend, our JavaScript code is executing, which is producing the list of feed titles we have. Blend executes your HTML, JavaScript, and CSS as it detects changes to make sure that you're editing the live version of your app. Sometimes it gets a little confused, however, so you can kick it in the pants manually with the Refresh button in the upper right of the design surface.

9. This is one of the most amazing development features of any platform ever. Highly recommended.

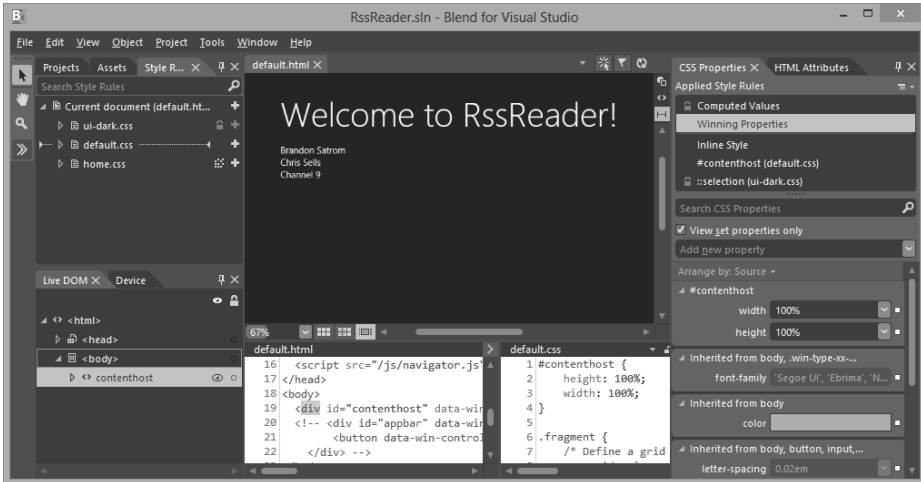


FIGURE 1.9: Microsoft Blend for Visual Studio 2012

To take advantage of that power, let's do a little work in Blend. Right now the JavaScript code is generating a bunch of `div` elements instead of using one of the many controls that comes out of the box for Windows Store development. In particular, we'd like to use a `ListView` control to display those feed titles. Before we do that, however, we want to open the `home.js` file either in Blend (via the Projects tab in the upper left) or in Visual Studio 2012 (making sure to save it and let Blend reload it when it asks) to remove the code in the `ready` function:

```
// home.js
(function () {
    "use strict";

    // define the feeds
    window.feeds = new WinJS.Binding.List([
        { title: "Brandon Satrom", url: ... },
        ...
    ]);

    WinJS.UI.Pages.define("/pages/home/home.html", {
        ready: function (element, options) {
            // let the ListView show the feeds
        }
    });
})();
```

In addition to removing the code in the `ready` function that creates the `div` elements for our feed titles, we've wrapped our feed data in an instance of the `WinJS.Binding.List` object, which will let the `ListView` consume it via data binding.

Once we've updated `home.js`, Blend will show that there are no elements showing the feed data (and if it doesn't, the Refresh button in the upper right above the design surface will put it right). Instead, it will show the paragraph element that says, "Content goes here." You can delete that by clicking on it twice—first to select the `contenthost` element on `default.html`, and then again to select the paragraph in the hosted `home.html`—and then pressing the Delete key.

To see the set of WinJS controls so that you can add a `ListView` control, click on the Assets tab in the top left and choose JavaScript Controls. Figure 1.10 shows the Assets tab and the `ListView` control.

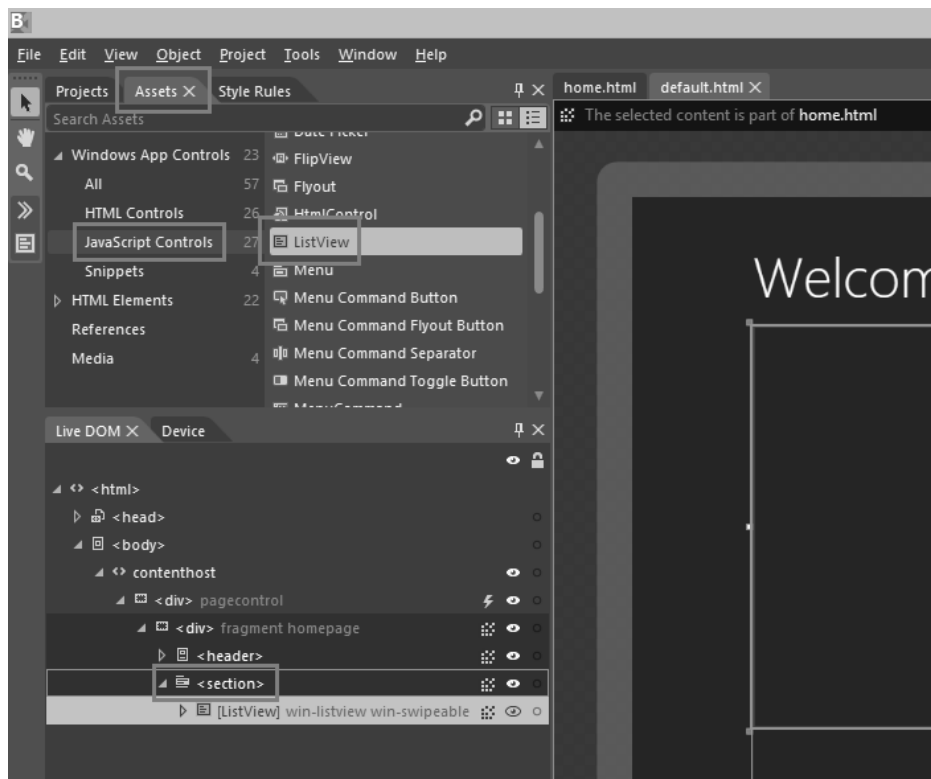


FIGURE 1.10: Using Blend to add a `ListView` control to a Windows Store app

Before adding a `ListView`, make sure you've got the section where we've been putting our content selected so that it makes a big target. The easiest way to do that is to drill into the Live DOM on the lower left until you find it, as Figure 1.10 also shows. Now, you can drag the `ListView` from the Assets tab either onto the section tag in the Live DOM or onto the design surface—it's up to you.

Once you have the `ListView` in the DOM, you can edit the HTML and CSS properties on the right, as Figure 1.11 shows.

The Windows App Controls section of the HTML Attributes tab (as seen in Figure 1.11) is where you get to set all of the options specific to a particular control. For the `ListView`, we want to set the `itemDataSource` property to bind to the feeds data we created earlier in `home.js`. Specifically, we want to set `itemDataSource` to `feeds.dataSource`, which is a property of the `WinJS.Binding.List` object we created earlier, specifically for binding with list controls. Once we've done that, you'll see the `ListView` update itself immediately to show the data, as Figure 1.11 shows, in a jumbled mess.

The problem is that we're no longer separating the data from the feeds list into the specific parts we want to show (the `title`) and the parts we don't want to show (the `url`). To do that, we'll need to provide the `ListView` with a template.

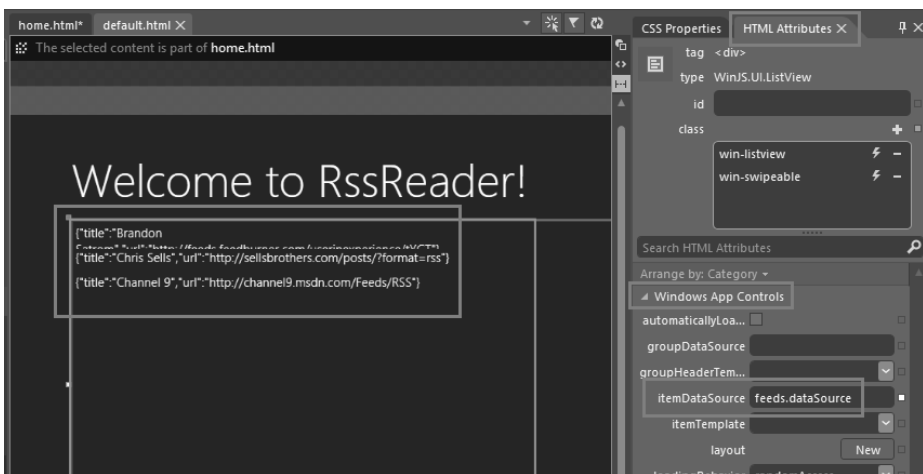


FIGURE 1.11: Using Blend to bind a `ListView` to a list of data

A template is a reusable chunk of HTML that is provided for the purpose of binding items, like what we want to do here. The easiest way to do this is to click on the `itemTemplate` property right underneath the `itemDataSource` property and choose the `<Create new template>` option, which will give you the Create New Template dialog shown in Figure 1.12.

Once you identify your new template, you'll see that the display has updated a little, as Figure 1.13 shows.

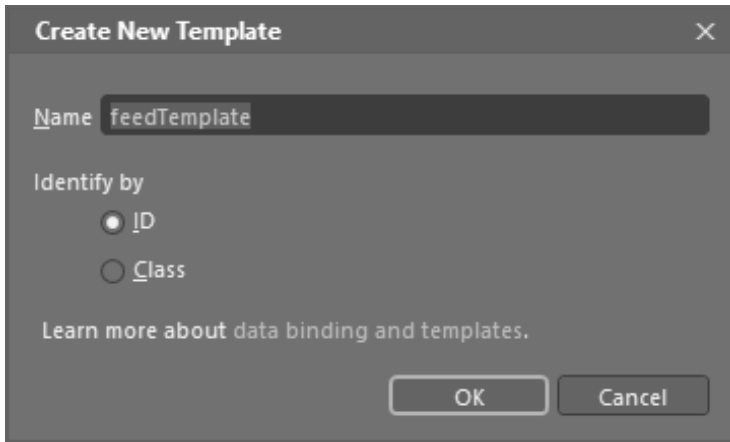


FIGURE 1.12: Using Blend to create a data template

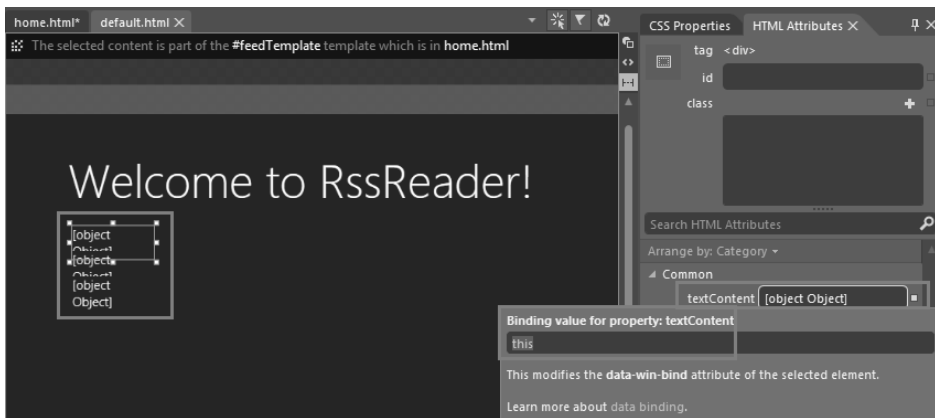


FIGURE 1.13: Using Blend to examine the contents of a data template

By selecting an item from the `ListView`, you'll see that the `textContent` for that item is binding to the entirety of each object, which you can see by clicking on the little square next to the `textContent` field and choosing Edit Data Binding. The "Binding value" dialog shows the binding to the `this` value, but we only want to bind to the `title` property of each object instead of the whole object. To fix this, set the value of the binding to `text` instead of `this`, click the Refresh button, and you'll get exactly what you're after, as Figure 1.14 shows.

Some important stuff is going on under the covers in the HTML with respect to binding and controls that you'll want to read all about in Chapter 2, "Binding and Controls."

In addition to editing HTML—especially HTML5, which works well with WinJS—Blend is also excellent at managing CSS styles. To see the set of styles in our project, click on the Style Rules tab on the upper left (Figure 1.15).

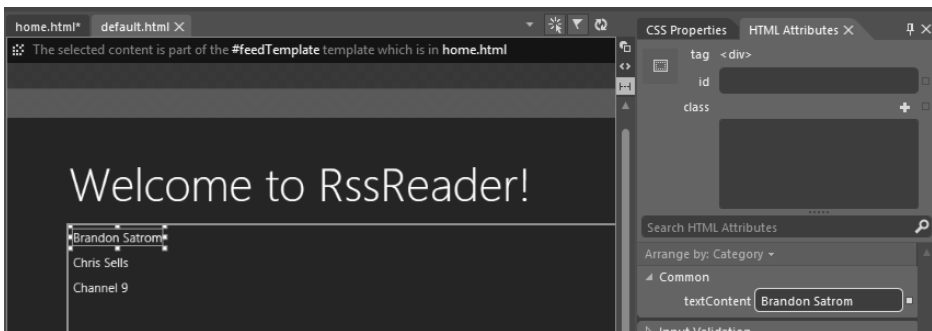


FIGURE 1.14: Using Blend to modify the contents of a data template

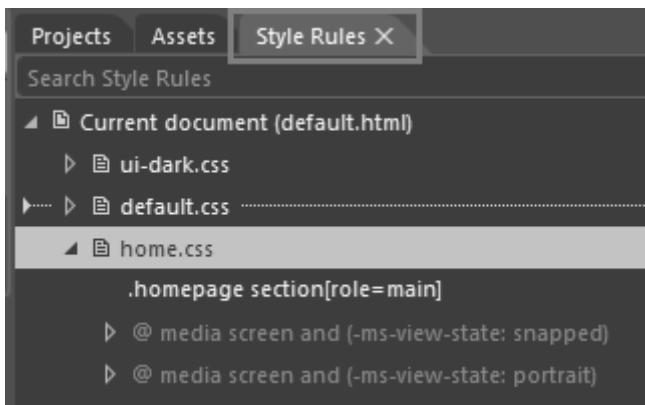


FIGURE 1.15: Using Blend to manage the styles in your project

If we want to increase the size of the feed titles to make them more visible, we'll first want to create a new style, which you can do by right-clicking on `home.css`, choosing **Add Style Rule**, and then entering the selector of your new style; for example, `.feedTitle` (including the leading dot). To associate the new CSS class with your feed titles, select one of the feed titles on the design surface and set the class property in the HTML tab to the new class; for example, `feedTitle` (no leading dot). Figure 1.16 shows what this looks like.

Associating the `feedTitle` class with one of the feed titles in the list actually sets it for all of them because the feed titles come from a repeating template, and Blend knows that. You can see this by selecting `.feedTitle` from the Applied Rules list of the CSS Properties tab, as Figure 1.17 shows.

The boxes around the feed titles in Figure 1.18 make it clear what elements will be affected when you make CSS property changes. Now, it's very easy to set the width and font size for all feed titles at once, as Figure 1.18 shows.

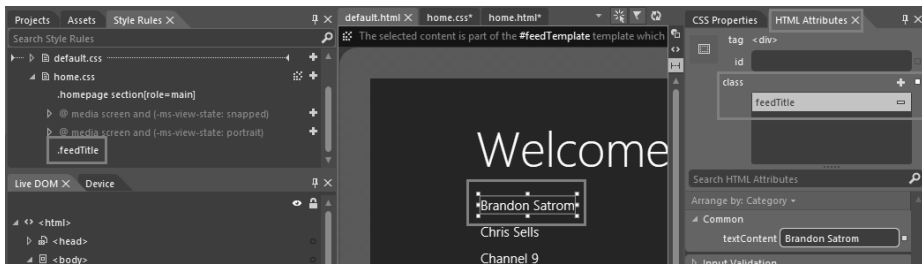


FIGURE 1.16: Using Blend to associate a class with one item from a template associates it with all items from that template.

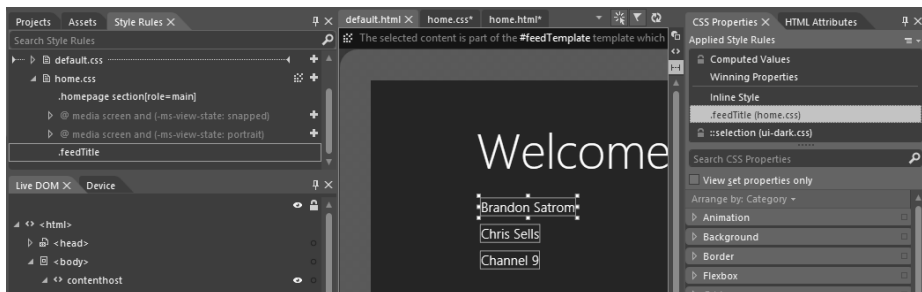


FIGURE 1.17: Selecting a CSS rule in Blend shows all elements to which that rule is applied.

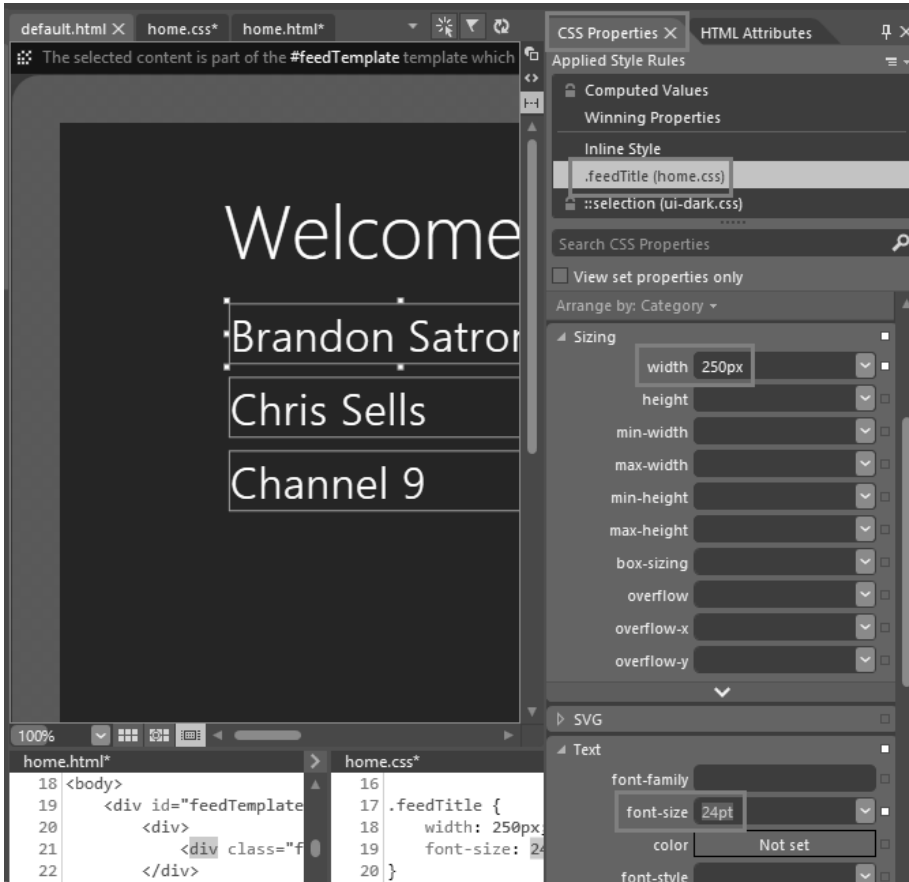


FIGURE 1.18 Using Blend to change CSS properties interactively

Blend allows you to edit an app interactively while it's running, which gives you a very fast turnaround time when you're designing the look and feel of your app. For a much more thorough examination of what you can do with CSS in Windows Store apps, including Blend's support for styling and view modes (e.g., landscape, portrait, etc.), you'll want to read Chapter 3.

So, at this point our feed titles are attractive. However, they're not yet interactive. As the user invokes one of the items—using the keyboard, mouse, or finger—we want to take the user to a page showing the items from that feed. And for that, we'll need navigation.

Navigation

The idea of navigation between pages is not new. In fact, it's the Hyper-text part of the Hypertext Markup Language (HTML). As the user clicks on links (or HTML elements with `onClick` handlers), we often want to bring up a whole new page of data, controls, images, and so forth. In the browser, when this happens, we most often pull down a new page, blanking out the screen and clearing out all of the current state. While we can navigate in a Windows Store app in the same way we can in the browser, we generally prefer to use the navigation service built into WinJS, which gives us much greater control over the UI as we move from page to page and allows us to keep the app state we build up over time, like we can with our list of feeds.

However, before we navigate anywhere, we need somewhere to navigate to. And for that, you'll want to right-click on the pages folder in your project from the Solution Explorer and add a new folder for your page using **Add | New Folder**, calling it `postsPage`. This will hold the files for your new page, which you can add to that folder by right-clicking and choosing **Add | New Item** and then choosing the **Page Control** item from the **JavaScript | Windows Store** category. What you'll see looks like Figure 1.19.

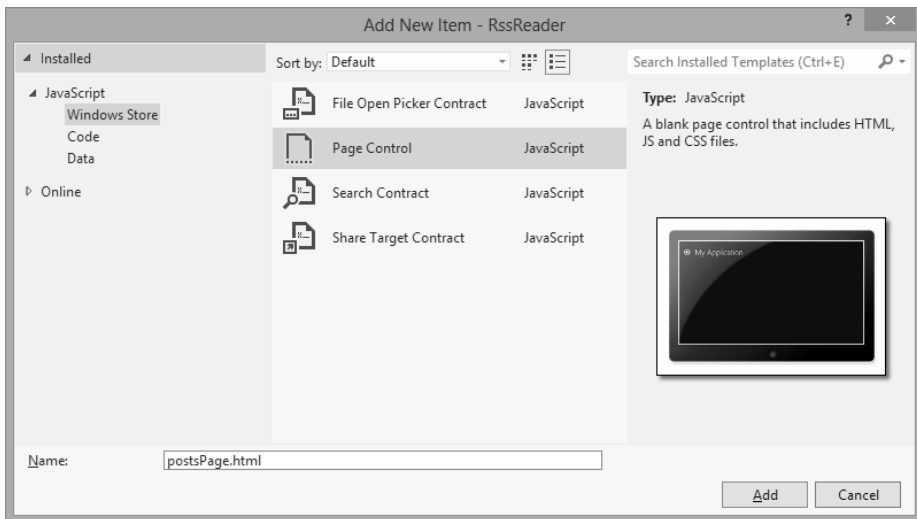


FIGURE 1.19: The Add New Item dialog for Windows Store apps

Each of the item templates in the Windows Store category produces a set of three files—an HTML file, a CSS file, and a JavaScript file—that compose a page control suitable for use in WinJS navigation. The Page Control template creates a blank page control. The other three templates help you implement shell contracts, which you can read all about in Chapter 9, “Shell Contracts.”

Entering the name, such as `postsPage.html`, and pressing Add creates the three new files for our page control, as Figure 1.20 shows.

That’s all we need to do to get a page ready to be a navigation target—the question is, how do we perform the navigation? In the case of the `ListView`, we need to let the `ListView` know we’d like to be notified when an item is invoked, as shown in the code on the following page.

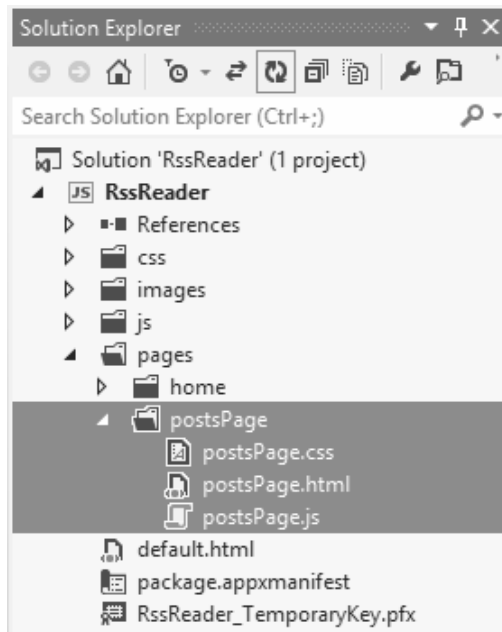


FIGURE 1.20: A new page control added to a Windows Store app

```
<!DOCTYPE html>
<!-- home.html -->
<html>
<body>
  ...
  <section aria-label="Main content" role="main">
    <div data-win-control="WinJS.UI.ListView"
        data-win-options="{
          itemDataSource: feeds.dataSource,
          itemTemplate: select('#feedTemplate'),
          selectionMode: 'none',
          oniteminvoked: feedInvoked}">
      </div>
    </section>
  </div>
</body>
</html>
```

Remember when we added the `ListView` control to the design surface in Blend? All that did was add the `div` and set the `data-win-control` and `data-win-options` attributes. The `div` represents the WinJS control in the HTML DOM, and `WinJS.UI.ListView` is the name of the JS constructor function (which you can learn all about in Chapter 2). You may also recognize the `itemDataSource` and `itemTemplate` settings we set in the Blend properties panel.

You don't have to use Blend to edit those properties; your favorite text editor will do. In this case, we need to set the selection mode to `none` (we don't want selection—we want invocation), and we set the name of the handler we want to call when the user invokes an item. The handler is implemented in the corresponding JavaScript file:

```
// home.js
(function () {
  "use strict";

  window.feeds = new WinJS.Binding.List([
    { title: "Brandon Satrom", url: ... },
    ...
  ]);

  // mark the event handler as safe for declarative use
  window.feedInvoked = WinJS.UI.eventHandler(function (e) {
    // navigate to the page to show the feed's posts
    var feed = feeds.getAt(e.detail.itemIndex);
```

```
WinJS.Navigation.navigate(  
    "/pages/postsPage/postsPage.html", { feed: feed });  
});  
  
...  
})();
```

The `feedInvoked` handler is wrapped in the `eventHandler` function, which marks it as safe for use from the `data-win-options` in the `home.html` file. This is a security measure to make sure that HTML downloaded from the Internet doesn't get to hijack your apps.

The implementation of the `feedInvoked` handler reaches into the `detail` property of the `event` object to find the index of the item that was invoked. This `feed` object is passed to the `postsPage` using the `navigate` method. The navigation services of WinJS then loads the `postsPage` and passes the `feed` object to the `ready` function via the `options` parameter:

```
// postsPage.js  
(function () {  
    "use strict";  
  
    WinJS.UI.Pages.define("/pages/postsPost/postsPage.html", {  
        ready: function (element, options) {  
            // TODO: do something with the feed object the user invoked  
            var feed = options.feed;  
        },  
    });  
})();
```

Now that we have an `invoke` handler set up on the `ListView`, clicking on a feed title on the home page (Figure 1.21) brings us to the page we've built to show the feed's posts (Figure 1.22).

By now, you may have noticed that while the `Back` button element is present in `home.html`, it's not showing in Figure 1.21, even though it is showing in the `postsPage.html` shown in Figure 1.22. That's because the navigation support in the templates is smart enough to know that there is no history before the home page to go back to, which is why it only shows the `Back` button where there is a "back" to go back to. Further, you can't see this, but the templates also support the `Back` and `Forward` keystrokes that the browser supports (like `Alt+Left Arrow` and `Alt+Right Arrow`).

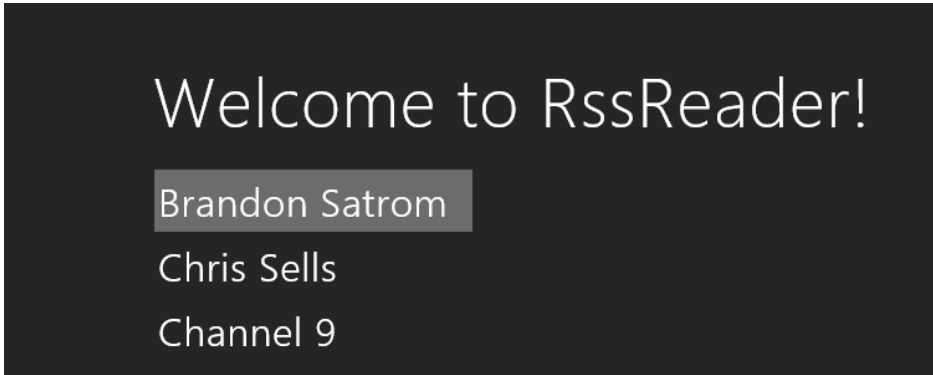


FIGURE 1.21: Triggering the invoke event on a `ListView` control

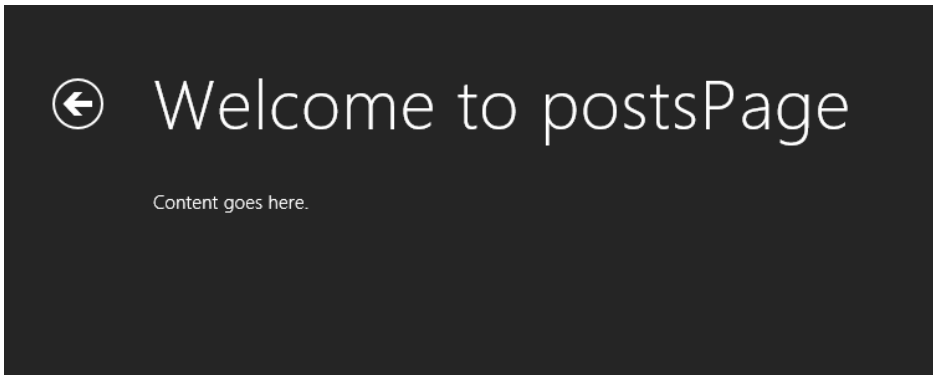


FIGURE 1.22: Navigating to a page control using the WinJS navigation service

All of this means that you can write your pages as page controls and pass objects around, letting the navigation support in the templates do the heavy lifting.

Of course, we're not done with even the basic functionality of our RSS Reader yet because we still haven't downloaded the posts from the selected feed. To do that, we've got to write a little networking code.

Networking in WinJS and WinRT

A large number of client-side apps require access to data provided over the Web, among them e-mail, photo browsing, social networks, music playback, document syncing, and multiplayer games. If you can name a popular app built in the past decade, chances are that it makes use of data accessed over a network. Toward that end, Windows Store apps have several ways to access data over the network, including the most basic: the XMLHttpRequest object.

XMLHttpRequest (XHR) is the name of the object that sparked the AJAX¹⁰/Web 2.0 revolution in 2005 (although the object has been part of Internet Explorer since version 5.0, released in 1999).¹¹ It provides for downloading data using HTTP. The xhr function provided with WinJS is an XMLHttpRequest wrapper that only requires the URL from which to retrieve data:

```
// postsPage.js
...
WinJS.UI.Pages.define("/pages/postsPage/postsPage.html", {
  ready: function (element, options) {
    // download the feed
    this.feed = options.feed;
    var pageTitle = element.querySelector(".pagetitle");
    pageTitle.innerText = this.feed.title;

    this.section = document.querySelector("section[role=main]");
    this.section.innerHTML = "<p>downloading...</p>";

    // download using XMLHttpRequest by creating a promise and
    // telling it what to do when it's done

    // the long way
    var xhrPromise = WinJS.xhr({ url: this.feed.url });
    xhrPromise.done(
      processPosts.bind(this), downloadError.bind(this));

    // the short way (recommended)
    WinJS.xhr({ url: this.feed.url }).
      done(processPosts.bind(this), downloadError.bind(this));
  },
});
```

10. Asynchronous JavaScript And XML, as coined by Jesse James Garrett in 2005.

11. Brandon likes to claim that he was doing AJAX "before it was cool," as far back as 2001 using XHR, ASP 3.0, VB6, and COM, though we haven't found anyone to corroborate his story.

Before downloading the feed data, we stash the feed object into a property associated with this instance of the `postsPage` page control, set the page title using the feed's title, and set a progress indicator for download. The reason we let the user know that we're downloading is because there's no telling how long it's going to take to do the actual download. Further, when we call the `xhr` function, passing in the URL for the feed, the result is not returned to us synchronously, blocking UI updates until the data winds its way back from some random server on the Internet. Instead, the return from `xhr` is a promise.

In fact, all asynchronous functions in WinJS (and in the WinRT) return an instance of `WinJS.Promise`, which represents results to be provided at some time in the future. The `Promise` object exposes the `done` method, which takes three functions as optional arguments: one for success, one for failure, and one for progress.

Upon success, our `processPosts` method is called:

```
// process using XMLHttpRequest
function processPosts(request) {
    // clear the progress indicator
    this.section.innerHTML = "";

    // parse the RSS
    var items = request.responseXML.querySelectorAll("item");
    for (var i = 0, len = items.length; i < len; i++) {
        var item = items[i];
        var parent = document.createElement("div");
        appendDiv(parent,
            item.querySelector("title").textContent, "postTitle");
        appendDiv(parent,
            item.querySelector("pubDate").textContent, "postDate");
        appendDiv(parent,
            item.querySelector("description").textContent, "postContent");
        this.section.appendChild(parent);
    }
}

function appendDiv(parent, html, className) {
    var div = document.createElement("div");
    div.innerHTML = toStaticHTML(html);
    div.className = className;
    parent.appendChild(div);
}
```

This code is pretty standard HTML DOM manipulation and XML processing code familiar to any experienced JavaScript programmer, creating `div` elements as we did earlier in the chapter. The only thing that's unique to Windows Store apps is the call to the `toStaticHTML` method. This call is specifically for when we have random HTML from an unknown source. By default, when setting the HTML of an element, the HTML engine will throw an exception if it finds a piece of dynamic HTML such as a script tag. The `toStaticHTML` call strips out any dynamic HTML it finds, rendering the content unable to take over your app.¹²

In the event that there's an error, we let the user know:

```
function downloadError(feed) {
    this.section.innerHTML = "<p>error</p>";
}
```

With this code in place as well as some styling in `postsPage.css`, our app is finally starting to rock, as you can see in Figure 1.23.

At this point, there are a few nits in our networking code that we might like to work through. For example, Brandon puts a summary of his posts in his feed's description field, whereas I put my entire set of content in there (both approaches are valid). Also, the XML parsing code we've written is specific to RSS,¹³ whereas most blogs these days support Atom.¹⁴ Luckily, because RSS and Atom are so prevalent on the Internet, the WinRT library provides a set of types for dealing with feeds of both syndication formats:

```
WinJS.UI.Pages.define("/pages/postsPage/postsPage.html", {
    ready: function (element, options) {
        ...
        // download using WinRT
        var syn = new Windows.Web.Syndication.SyndicationClient();
        var url = new Windows.Foundation.Uri(this.feed.url);
        syn.retrieveFeedAsync(url).done(
            processPosts.bind(this), downloadError.bind(this));
    },
});
```

12. If you'd like to know more about your options for bringing external HTML into your app safely, see Chapter 8, "Networking."

13. The RSS format is an XML language for publishing updates to content-oriented data.

14. The Atom syndication format is the successor to RSS.

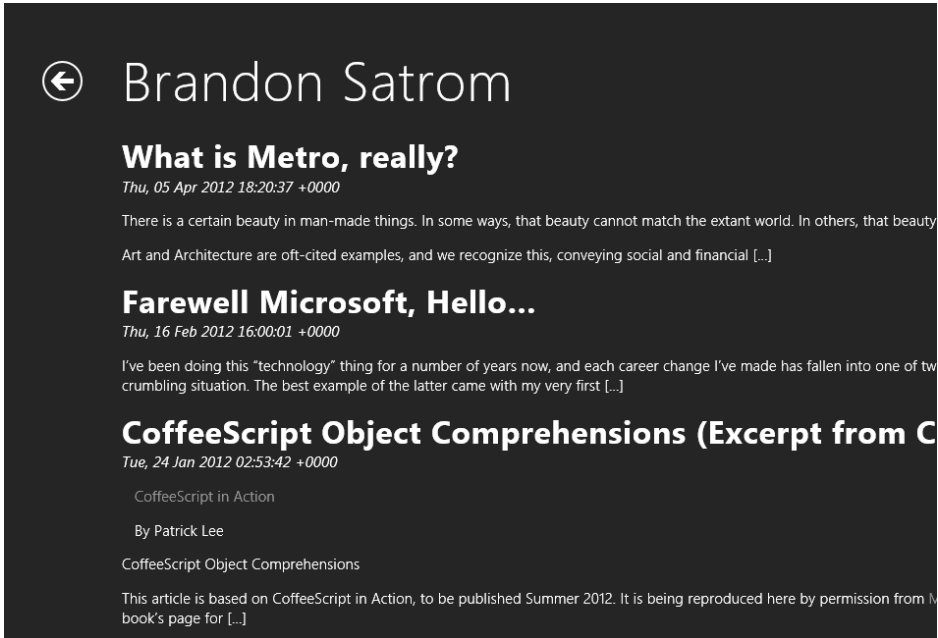


FIGURE 1.23: Showing the contents of an RSS feed using WinJS.xhr

In this code, we've replaced the use of the `xhr` function with the WinRT `SyndicationClient` and `Uri` types from the `Windows.Web.Syndication` and `Windows.Foundation` namespaces, respectively. Like the `xhr` function, the `retrieveFeedAsync` function is asynchronous, returning a promise that works exactly like every other async function in WinJS or WinRT. In our success handler, we handle a list of posts instead of raw XML:

```
// process using WinRT
function processPosts(request) {
    // clear the progress indicator
    this.section.innerHTML = "";

    // iterate over the items
    for (var i = 0, len = request.items.length; i < len; i++) {
        var item = request.items[i];
        var parent = document.createElement("div");
        appendDiv(parent, item.title.text, "postTitle");
        appendDiv(parent, item.publishedDate, "postDate");
        appendDiv(parent, item.summary.text, "postContent");
        this.section.appendChild(parent);
    }
}
```

The updated networking code is a little smarter about where Brandon keeps all of his content, as Figure 1.24 shows.

And not only is the WinRT smart about Brandon's feed and RSS versus Atom, but you'll notice that Visual Studio 2012 is smart about the WinRT. At no time did I need to add a WinRT reference or do anything else special to access a WinRT type or namespace. In fact, if you start typing "Windows." inside Visual Studio 2012, you'll see that it knows all about it (see Figure 1.25).

You'll see a great deal more of WinRT throughout this book, but I encourage you to dig around the Windows namespace on your own; there's a lot of good stuff in there.

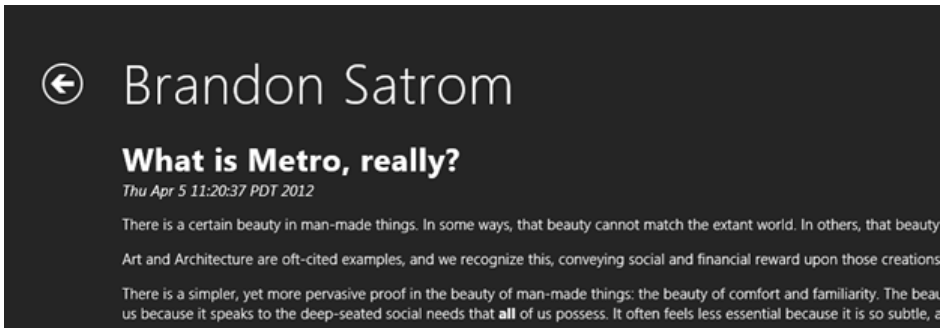


FIGURE 1.24: Showing the contents of an RSS feed using the WinRT SyndicationClient

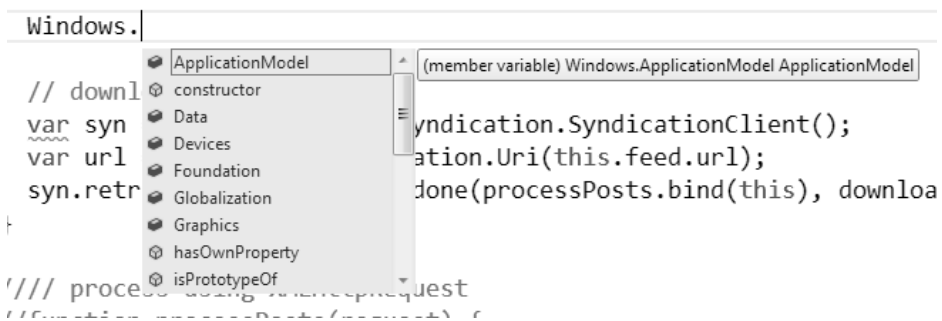


FIGURE 1.25: Visual Studio 2012 knows WinRT!

Split App Template

Further, not only does Visual Studio 2012 know about the WinRT namespaces and types, but also it has been built to know about the Windows 8 user experience style guidelines themselves. As I mentioned, the Blank App, Fixed Layout App, and Navigation App project templates all produce apps that are essentially blank, making for a good base from which to build up. However, the Grid and Split App project templates are meant to be living, breathing Windows Store apps that follow the UX guidelines to the letter, helping you make sure that you'll build great Windows 8 apps as easily as possible.

For example, if you run the Split App project template and run the app without any changes, you'll have an app with two pages, as shown in Figure 1.26 and Figure 1.27.

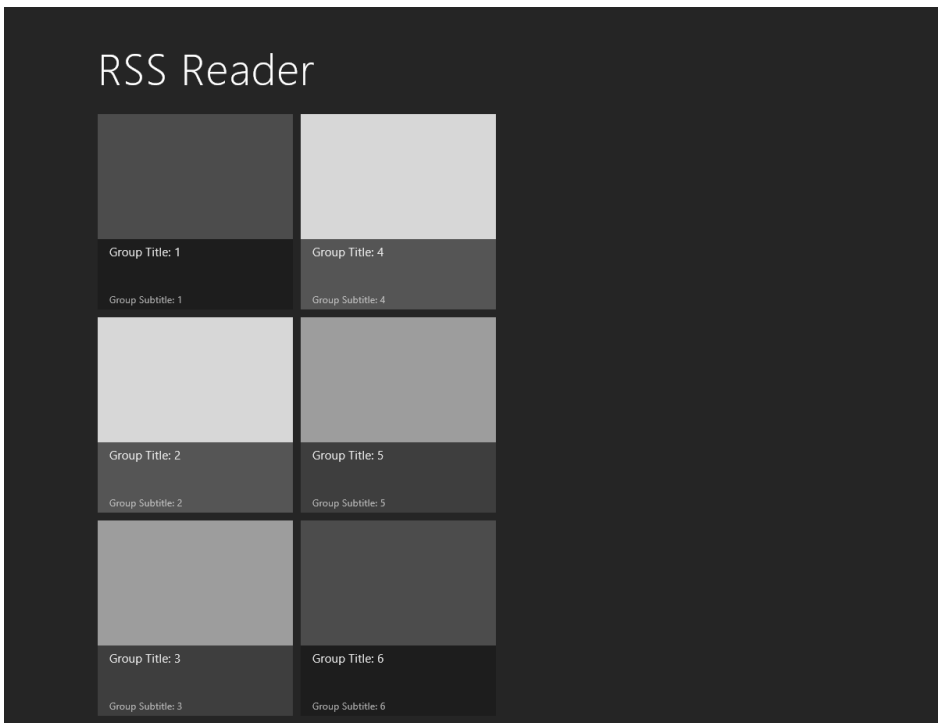


FIGURE 1.26: The `itemsPage.html` page from the Split App project template, showing groups of things



FIGURE 1.27: The `itemsPage.html` page from the Split App project template, showing a list of items

The home page shown in Figure 1.26 is meant to act as a group of things, such as teams of players, groups of people, or, as in our case, feeds of news items. The page you get when you click on one of the groups is shown in Figure 1.27. It represents a list of items in a group; for example, players in a team, people in a group, or news items from a particular feed. In short, the Split App is perfect for our RSS Reader app. The data is all static sample data hardcoded in `data.js`, but replacing the static data with dynamic data is a pretty easy thing to do:

```
// data.js
...
var list = new WinJS.Binding.List();
...
// TODO: Replace the data with your real data.
// You can add data from asynchronous sources whenever it becomes available.
```

```
//generateSampleData().forEach(function (item) {
//    list.push(item);
//});

var feeds = [
{ key: "feed1", title: "Brandon Satrom", subtitle: "blog",
  backgroundImage: darkGray,
  description: "blog",
  url: "http://feeds.feedburner.com/userinexperience/tYGT"
},
{ key: "feed2", title: "Chris Sells", subtitle: "blog",
  backgroundImage: lightGray,
  description: "blog",
  url: "http://sellsbrothers.com/posts/?format=rss"
},
{ key: "feed3", title: "Channel 9", subtitle: "blog",
  backgroundImage: mediumGray,
  description: "blog",
  url: "http://channel9.msdn.com/Feeds/RSS"
}
];

feeds.forEach(function (feed) {
    // download the feed
    var syn = new Windows.Web.Syndication.SyndicationClient();
    var url = new Windows.Foundation.Uri(feed.url);
    syn.retrieveFeedAsync(url).done(processPosts.bind(feed));
});

function processPosts(request) {
    var feed = this;
    request.items.forEach(function (item) {
        // create a post for each item
        var post = {
            group: feed,
            title: item.title.text,
            subtitle: item.publishedDate,
            description: "post",
            content: toStaticHTML(item.summary.text),
            backgroundImage: feed.backgroundImage,
        };

        // let the list know about each post
        list.push(post);
    });
}
...

```

Toward the top of `data.js` is a comment that begs us to replace the use of the sample data with our real data. Here we've dropped in our array of feeds to iterate over, pulling in our posts asynchronously, just as we did earlier in the chapter. The code to pull in our data and matching it to the shape of the group and item data assumed in the rest of the app is all that's required to build the complete RSS Reader built up manually throughout this chapter (and shown in Figure 1.28 and Figure 1.29).

As you can see in Figure 1.29, the second page of the Split App (the `splitPage` page control) is fancier than what we built: It uses the CSS Grid for layout, changing the content the user is viewing on the right based on the item he chose on the left. The other major feature that the built-in Split

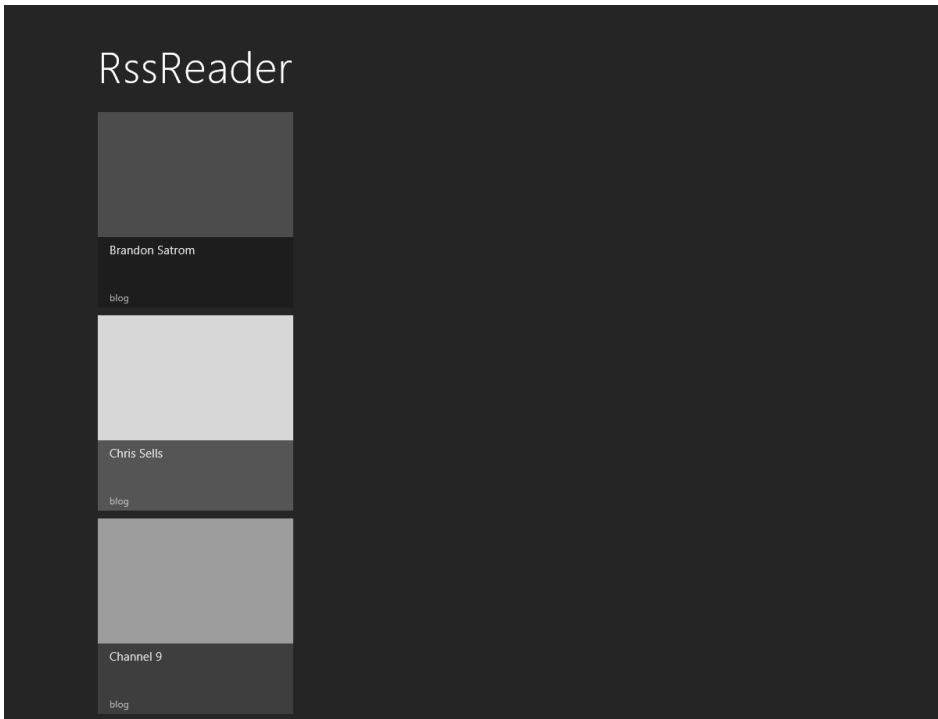


FIGURE 1.28: The `itemsPage.html` page from the Split App project template, showing real data

and Grid App project templates have is support for view state changes as the user moves between landscape, portrait, filled, and snapped modes. Figure 1.30 shows our shiny new RSS Reader in snapped mode (which you can get to most easily by pressing Win+period).

You can read all about the view states in Chapter 3, “Layout.”

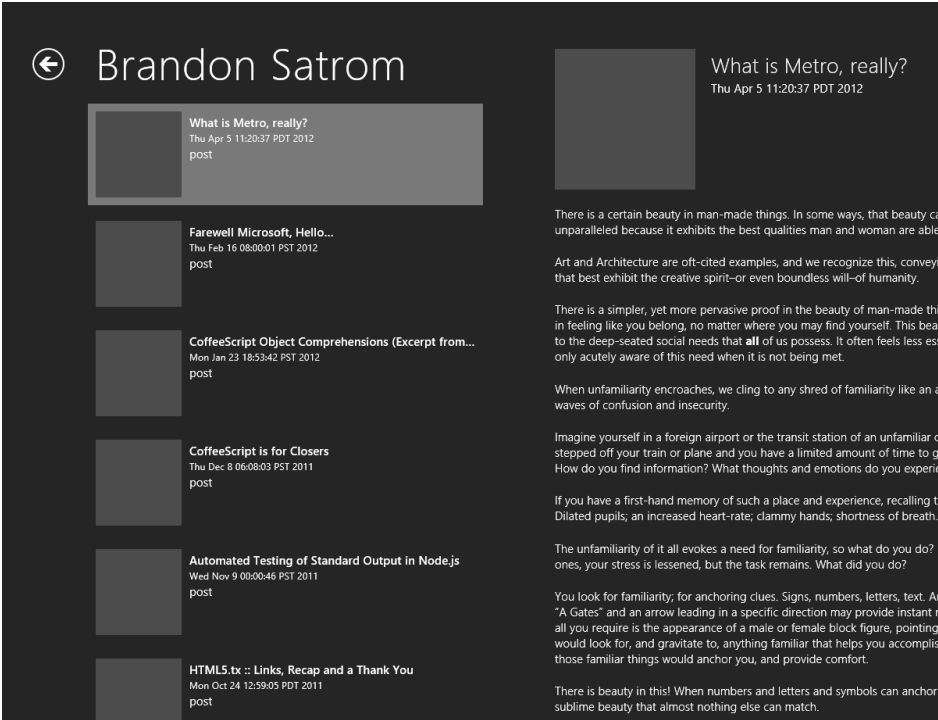


FIGURE 1.29: The splitPage.html page from the Split App project template, showing real data

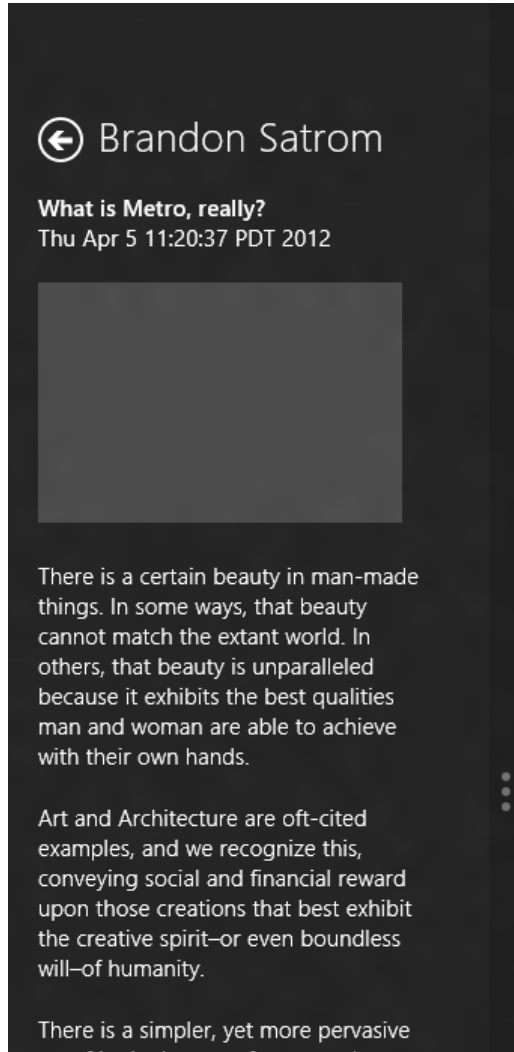


FIGURE 1.30: The snapped mode support built into the Split App project template

The Rest

But wait! There's more!

- If you wanted to add support for the media enclosures that the Channel 9 feed provides, you can learn how to do that in Chapter 5, "Media."
- To add cool animation support as the user navigates between pages, read Chapter 6, "Drawing and Animation."
- For keeping track of the posts that users have previously read between sessions of the app or to let them choose which feeds they'd like to read, check out Chapter 7, "App State."
- To let users create posts tagged with their current GPS coordinates or to refresh the feed data when users shake their tablets, read Chapter 11, "Device Interaction."
- If you need to extend your Windows Store app with native code written in C++ to do something heavy-duty, like calculating the 10,000th digit of pi, you'll want to dig into Chapter 12, "Native Extensibility."
- And finally, to learn how to deploy your app, handle trial mode, or stick advertisements at the bottom of each post, you'll want to read Chapter 13, "Making Money."

Where Are We?

This chapter has been a whirlwind tour through the tools, techniques, and technologies associated with Windows Store apps built using JavaScript for Windows 8 via Visual Studio 2012 and Blend. It may seem like a lot, but we've really only scratched the surface. Keep reading; we're just getting started!



Index

Symbols

- \$ (dollar sign) in substring attribute selectors, 552
- & (ampersand) in C++ references, 423
- > (arrow) operator, 423
- * (asterisk)
 - in C++ pointers, 423
 - in substring attribute selectors, 552
 - universal selector, 544
- ^ (caret)
 - in substring attribute selectors, 552
 - type modifier in C++/CX, 422
- > (child selector), 546
- : (colon) in pseudo-class and pseudo-element selectors, 549
- . (dot)
 - class selector, 545
 - dot operator, 413
- = (equals sign)
 - === and !== operators, using for object identify, 424
 - == (equal to) operator in JavaScript, 509
 - === (identity) operator in JavaScript, 509
- ! (exclamation mark) as warning icon, 149
- @font-face rule, 125, 148
 - referencing in font-family property, 126
- # (hash symbol), id selector, 545
- [] (square brackets), array operator, 55, 432
- _ (underscore), prefixing property and method names, 73

A

- ABI (application binary interface), 406
- Accelerated Massive Parallelism (AMP), 405
- AccelerationX, Y, and Z values, 399
- accelerometer, 398
- accessibility
 - HTML5 video integrated into applications, 166
 - information on your app, 464
- activation, 240
 - activated event, WinRT, 245
 - activate event, 239
 - app activation from toast, 361
 - checking for toast activation, 362
 - file, 259–261
 - primary and secondary tiles, 347
 - search, 300
 - Share target, 311, 314
 - WinJS, 505
 - WinJS activated helper event, 252
- ActivationKind
 - file, 260
 - search, 300
- Add-AppxPackage command, 4
- addEventListener method, 76, 442, 443
- Add New Item dialog, 24
- adjustVolume function, 162
- ad platforms, 453
- ads in your app, 476–482
 - media-based ads, 477–480

- ads in your app (*contd.*)
 - rules for Windows 8–style apps, 476
 - text-based ads, working with, 480–483
 - working with Windows 8 Ads SDK, 477
- adUnitId value, 478, 481
- advanced features, specifying for your app, 466
- ::after pseudo-element, 138
- age rating and rating certification, 466
- AJAX (Asynchronous JavaScript and XML), 273
- AMP (Accelerated Massive Parallelism), 405
- animatable properties, 217
- animation, 193
 - activation and dismissal, Win8 touch apps, 373
 - circle in Canvas, 203
 - fade effects, for ad control, 482
 - in Windows Store apps, 212–224
 - fast and fluid, 213
 - transforming and animating with CSS, 213–219
 - using WinJS Animation library, 220–224
 - SVG animation with JavaScript, 198
- animation-direction property, 219
- animation-duration property, 219
- animation-iteration-count property, 219
- animation-name property, 219
- animation-timing-function property, 219
- anonymous functions, 521
- AppBarCommands, 140
 - referencing AppBarIcon enum values, 146
 - Windows 8 touch-style control, 373
- AppBar control, 140
 - building for Windows 8–style apps, 372
 - Pin to Start Screen button, 344, 346
 - touch, mouse, and keyboard interactions with, 379
 - WinJS animation, 221
- AppBarIcon enumeration, 140
 - enumeration values and icons, 142–146
- appConStreamReference helper, 304
- application binary interface (ABI), 406
- ApplicationData class, 227, 252–254
 - current.localFolder property, 253
- Application object
 - activated event, 241, 506
 - local, roaming, and temp file folders, 255
 - settings event, 230
- ApplicationView class, 113
- ApplicationViewState enumeration, 113
- app name, reserving, 454, 456
- AppPackages folder, 467
- AppSimulator object, 486
- Apps search provider, 297
- app state, 225–266
 - files, 252–256
 - libraries, 256–265
 - lifetime, 238–252
 - settings, 226–238
- .appx files, 4
 - adding and verifying addition of, 5
- appxmanifest.xml file, 3
 - example, 3
- appx module, 4
- .appxupload file, 467
- arcs, drawing in Canvas, 201
- arguments (functions in JavaScript), 522
- Array object, 433, 514
- ArrayReference class, 432
- arrays
 - JavaScript, 54, 512–514
 - WinRT types in C++/CX and JavaScript, 427, 431
- arrow (->) operator, 423
- article element, 538
- as method, 46
- assignment versus binding, 42
- Asynchronous JavaScript and XML (AJAX), 273
- asynchrony
 - asynchronous data model, 62
 - asynchronous functions, 32
 - concurrency and, 443–451
 - show Async method, MessageDialog class, 64
- AsyncStatus::Error, 445
- Atom syndication format, 31, 278
- attribute selectors, 548
- attributes (HTML), adding to controls via Blend, 19
- audio and video, 155–156
 - adding subtitles to video, 163–170
 - adding video effects, 167–170
 - audio formats supported in Windows Store apps, 159
 - audio in Windows Store apps, 170
 - creating background audio, 171–192
- audio bar, playing HTML5 audio, 159

- audio element, 156, 539
 - assigning `audioFile` objectUrl to, 384
 - attributes, 158
 - recording device, 381
- `audioFile` object, 384

B

- back buttons, 136
 - icon for, 137
- background audio, creating, 171–192
- `BackgroundCapableMedia` property, audio, 172
- background data transfer, 280–283
- `BackgroundDownloader` object, 280
- `BackgroundTaskBuilder` object, 352
- background tasks, 350–358
 - app with, declaration in manifest, 351
 - avoiding task duplication, 357
 - creating, 352–354
 - lock screen apps, 355–357
 - triggering, 351–352
- `backgroundtasks.js` file, 351
- badges, 348–350
 - for lock screen apps, 355
 - pulling in badge data from the Internet, 350
- Badge Update Manager (BUM), 348
- `base.js` file, 506
- `::before` pseudo-element, 138
 - Heydings font for, 150
- `bind` function implemented with the `call` function, 524
- binding. *See* data binding
- `bind` method, 51
- Bing
 - app, front and back with peeking, 334
 - Map control, hosting in Windows Store app, 288–291
 - Maps, using location data with, 391
 - search results, 275
- bitmap data, sharing, 320
- `BitmapTransform` object, 211
- Blank App template, 7
- Blend, 15
 - adding CSS media queries in, 87–90
 - changing display resolution in, 86
 - controlling device orientation, 94
 - controlling view states, 97
 - controls, binding, and styling in, 16–23

- adding `ListView` control via, 18
- binding `ListView` to data source, 19
- managing CSS styles, 21
- Flexbox section in CSS Properties window, 106
- high-resolution view of application, 92
- Multi-Column Layout, 109
- viewing CSS Grid Layout lines, 101
- blockquote element, 130
- body element, Grid layout defined on, 100
- borders, style rules for, 138
- BUM (Badge Update Manager), 348

C

- Cached File Updater contract, 296
- Calibri font, 120, 123–124
 - with font-feature-settings applied, 134
- callback parameters, `UICommand` object, 65
- call function, 523–525
- Call Stack (Visual Studio 2012), 14
- Cambria font, 120, 122
- `camelCase`, 412
- `CameraCaptureUI` API, 185–189
- `CameraCaptureUIMode`, 188
 - video, 188
- canceling ongoing downloads, 272
- cancelled event, background tasks, 353
- Canvas, 199–204
 - animating a circle, 203
 - choosing between SVG and, 204–206
 - drawing context for, 200
 - fill and stroke functions, 201
 - HTML5 graphics with, 193
 - pixel manipulation with, 206–209
- canvas element, 75, 199, 538
- capabilities, device, 380–387
 - adding in-app print capabilities, 384–387
 - recording devices, 381–384
- captions
 - adding subtitles to video, 163–170
 - caption element, removing float on, 98
- `captions.vtt` file, 164
- captured media, working with, 185–189
- Cascading Style Sheets. *See* CSS
- case
 - setting with `text-transform` property, 131
 - WinRT declarations projected into JavaScript, 412
- category and subcategory for your app, 464

C/C++

- C++ 11 lambdas, 436–439
- C++ for high-performance games, 205
- JavaScript programs interacting with, 405
- projecting WinRT constructs into C++, 407
- C++/CX (C++ Component Extensions), 407
 - adding C++ component to JavaScript program, 407–409
 - concurrency and asynchrony, 443–451
 - defining WinRT types, 411
 - delegates and functions, 435–440
 - WinRT classes, 413–421
 - WinRT objects, 421–424
 - WinRT types in C++/CX and JavaScript, 424–435
 - arrays, 431–433
 - strings, 429–431
 - value types, 433–435
- certification by Windows Store, 471
- certification failure report, 473
- Character Map application, 138
 - Segoe UI Symbol characters viewed in, 139
 - visualizing custom fonts in, 149
- Charms bar, 294
 - initiating contract via, 295
 - Search charm, 297
- checkboxes, adapting to touch, 368
- checkpoint event, 242, 251
- child selector, (>), 546
- circle element, 195
- circles
 - animation in Canvas, 203
 - drawing with Canvas, 199
 - drawing with SVG, 195
 - styling SVG circle, 196
- Class.define method, 519
- Class.derive and Class.mix methods, 520
- classes
 - class keyword in WinRT type declarations, 414
 - definitions via WinJS, 519
 - WinRT, 413–421
 - methods, 414
 - WinRT types in C++/CX and JavaScript, 425
- class selector (.), 545
- clipboard, 293
- closures, 435, 525
- code examples for this book, 75

colors

- adding background color to svg container, 196
- CSS background-color rule applied to svg element, 197
- defining for gradients, 195
- defining for radial gradient in Canvas, 200
- defining for radial gradient in JavaScript, 197
- column-count property, 108, 109
- column-fill property, 109, 110
- column-gap property, 109, 110
- column-rule-color property, 109, 110
- column-rule-style property, 109, 110
- COM (Component Object Model), 406
 - HRESULTS, 417
 - LPCWSTR-based APIs, 429
 - Windows Runtime Library (WRL), 407
- commandsrequested event, SettingsPane object, 229
- Common Language Infrastructure (CLI) metadata format, 406
- Common Language Runtime (CLR), 406
- Communications value,
 - msAudioDeviceType attribute, 171
- compass, 400
 - readings, 401
- concurrency and asynchrony, 443–451
- concurrency::concurrent_vector<T>, 451
- conditions for background triggers, 352
- Console value, msAudioDeviceType attribute, 171
- const modifier, 432
- constructors
 - JavaScript, 514–515
 - WinRT classes, 413
 - WinRT, defining and invoking from JavaScript, 421
- const wchar_t* values, 429
- contactnremoved event, 329
- ContactPicker object, 322–325
 - contactPickerUI argument, 326
- contact providers, 324, 325–332
 - debugging in Visual Studio 2012, 332
- Contacts contract, 322–330
 - Contact Picker, 296, 322–325
 - filtering contacts, 324
 - contact providers, 325–330
- contacts.html file, 327

- content property, 138
- contracts, 295–297. *See also* shell contracts
 - list of Windows 8 contracts, 296
- control attribute, 478
- Control Panel, 294
- controls, 63–78
 - adding via Blend, 17
 - ads from Advertising SDK, 478
 - caption bar on video controls, 165
 - custom, 70–78
 - control class, 72
 - events, 75
 - properties and methods, 73
 - custom video controls with JavaScript, 161
 - families of, 63
 - HTML5, 63
 - MediaControl object, for background audio, 174
 - page control for WinJS navigation, 25
 - playback controls for audio and video elements, 158
 - settings panel, 234
 - templates, 60–63
 - touch-friendly HTML controls, 367–369
 - touch, mouse, and keyboard interactions with, 379
 - WinJS, 66–69
 - WinJS animations, 221
 - WinJS touch-friendly controls, 369–370
 - WinRT, 64
- controls attribute, video element, 162
- converter function, 52
- costType function, 270
- counters
 - C++/CX implementation, 440
 - consuming C++/CX counter in JavaScript, 440
 - JavaScript implementation, 439
- crashes and hangs test, 460
- Create App Package wizard, 460, 474
- createEventProperties method, WinJS.
 - Utilities, 76
- createFileAsync function, 253
- createFiltered function, 56
- createGrouped function, 59
- Create New Template dialog, 20
- createSorted function, 56
- createToastNotifier method, 361
- cryptography, 466

- CSS, 9, 533
 - changing properties using Blend, 23
 - CSS3 Web Fonts, 124–128
 - execution by Blend, 16
 - layout capabilities, using to adapt your app, 99–104
 - adaptive layouts for app content, 103
 - CSS3 Grid Layout specification, 100–103
 - managing styles with Blend, 21–23
 - media queries
 - checking app orientation, 95
 - high-definition resolutions support, 91
 - styles for Snapped and Filled views, 97
 - using to adapt to resolutions, 86–91
 - styled HTML elements as control containers, 63
 - styling icon fonts, 150–154
 - styling media, 159
 - HTML5 video elements, 160
 - styling SVG elements and their children, 197
 - transforming and animating with, 213–219
 - animation, 218
 - transitions, 215
 - WinJS wrapper functions for, 223
 - transforms, 151
 - using Flexbox for adaptive UIs, 104–107
 - using for app layout and style, 541–558
 - CSS and Windows Store apps, 558–564
 - how CSS rules cascade, 555–560
 - where to define for Windows Store apps, 553–555
 - using Multi-Column Layout for adaptive content, 107–110
 - using to tweak typography, 129–135
- CSS3 Media Queries module, 87
- CurrentApp object, 494
- currentApp property, Store namespace, 483
- CurrentAppSimulator object, 483, 486, 494
- cursive fonts, 127

D

- DaleIpsium.com, 214
- data-* attributes (HTML5), 10, 539–541
- data/begin/end/size, 429
- data binding, 41–63
 - data list to ListView control for file picker, 182
 - grouping of binding lists, 58–59

- data binding (*contd.*)
 - initializers, 51–53
 - lists, 53
 - ListView, using Blend, 19
 - objects, 42–51
 - change in data updating bound elements, 47
 - listening for value to change, 50
 - rebinding different objects to same elements, 49
 - view model, 48
 - sorting and filtering binding lists, 55–57
 - tile updates, 339
 - types of, 42
 - using templates, 60–63
 - WinJS.Binding.List object, 18
- data context, setting in binding operation, 45, 48
- data conversion, 52
 - numeric age into words, 52
- data.js file, 62, 236, 247
- Data namespace, 235
 - appIconStreamReference helper, 304
 - resolveItemResource method, 305
- DataPackage object, 306, 310
- DataPackageView class, 319
- datarequested event, 307
- dataSource property, 54
- data templates, creating in Blend, 20
- data transfer, background, 280–283
- DataTransferManager object, datarequested event, 305
- data types. *See* types
- data-win-bind attribute, 44, 60
 - bind handler processing of, 51
- data-win-control attribute, 10, 60
 - control creation via, 68
- data-win-control declarations, 370
 - WinJS.UI.AppBar, 373
 - WinJS.UI.AppBarCommand, 373
- data-win-options attribute, 10
 - ad initialization, 478
 - custom clock control, 74
 - DatePicker control, 68
 - groupHeaderTemplate and itemTemplate properties, 61
- Date object, 68, 511
- DatePicker control, 66, 369
- debug configuration test, 460
- Debugger (Visual Studio 2012), 14
- debugging
 - Debug Output, 525
 - sessions, 246–249
 - using VS2012, 14
- decoders and encoders for images, 211
- default attribute, track element, 165
- default.css page, 89
- default.html file, 3, 9
- default.js file, 506
 - Share contract support, handling datarequested event, 307
- [DefaultOverload] metadata attribute, 416
- deferral, 240
 - getting and completing in WinJS, 241
- define function, 72
- defineProperties method, Object class, 73
- delay property, 217
- delegate keyword, 436
- delegates, 427, 435–440
 - creating WinRT delegates from C++ 11 lambdas, 439–451
 - using with events, 440–443
- descendant selector, 547
- description field, WinRT exceptions, 419
- description of your app, 468–471
 - promotional images, 496
- Desktop, 294
- details object, events, 77
- DevCenter, 454
- Developer account, setting up, 454
- developer low-res, 84
- developer skills, Canvas versus SVG, 204
- device interaction, 365–404
 - touch, 366–380
 - working with device capabilities, 380–387
 - working with location data, 387–394
 - working with sensors, 394–403
- Digi-Key sensor boards, 395
- digital signatures or DRM, 466
- dispatchEvent method, 77
- display property, 101
 - setting for Flexbox on container element, 104
 - setting for grid container element, 100
- div element
 - AppBar control, 140
 - data-win-bind attribute, 60
 - Grid layout defined on, 100
 - host for WinJS control content, 66
 - HTML5 semantic markup and, 538
 - id attribute, using as JavaScript object, 505
 - SemanticZoom control in, 377
- DLL files, 407, 409
- DLNA-certified devices, 189

- documentLibrary, PickerLocationID enum, 177
- document object, 504
- DOMContentLoaded event, 504
- DOM (Document Object Model), 504
 - adding ListView control via Blend, 19
 - page control added to, 12
- DOMEventMixin class, 76
- DOM Explorer (VS2012), 14
- Dosis web font, 126
- dot (.) operator, 413
- download, pausing, 282
- download progress, 274, 282
- DPI. *See* resolutions
- drawing and animation, 193–224
 - animation in Windows Store apps, 212–224
 - HTML5 graphics with SVG and Canvas, 193–206
 - manipulating pixels, 206–212
- duration property, 217

E

- E_BOUNDS HRESULT, 417
- ECCN (Export Commodity Classification Number), 467
- ECMA, 501
 - Common Language Infrastructure (CLI) metadata format, 406
- ECMA-262 version 5.1, 501
- ECMAScript, 501
 - ECMAScript 5 specification, 73
- element animations with WinJS, 221
- element property
 - custom clock control, 71, 74
 - DatePicker control, 67
- ellipse element, 195
- embed tags, SVG file in, 194
- Encoder object, 211
- enterPage and exitPage functions, 223
- enumerations, WinRT enum type, 412
- ErrorCode property, 445
- error codes, WinRT ABI, 417
- errors
 - app resources validation, 461
 - download, 282
- European Computer Manufacturers Association (ECMA), 501
- even and odd keywords, 553
- EventArgs class, 442
- event handlers
 - feedInvoked handler (example), 27
 - for slider change event, video element, and mute button clicks, 163
 - sourceRequested event, Play To, 191
- events, 440–443
 - custom control, 75
 - methods for event subscriptions and dispatching events, 76
 - JavaScript event handling, 506
 - resize event listener, 115
 - WinRT application events, 239
 - WinRT classes, 413
- exceptions
 - WinRT class methods and, 416–419
 - WinRT exceptions projected into JavaScript, 419
- Exceptions dialog (VS2012), 14
- execUnsafeLocalFunction, 286
- exists function, 256
- Export Commodity Classification Number (ECCN), 467
- external style sheets for CSS, 554

F

- Facebook, 295
- fadeIn effects, 222
- fast and fluid animations in Windows Store apps, 213
- feedInvoked handler (example), 27
- feeds, RSS and Atom, 31
- field access with dot (.) operator, 413
- figcaption element, 82
- figure element, 82
 - resizing, 98
- File Explorer, 294
- FileIO functions, 261
- FileOpenPicker object, 176, 261
 - with thumbnail view of user's image library, 177
- File Picker contract, 296
- file picker, working with media libraries, 175–180
 - other file picker types, 182
 - selecting multiple files, 180–192
- files, 252–256
 - activation, 259–261
 - file encoding test, 460
 - WinJS file helpers, 255–256
- FileSavePicker object, 183
- Files search provider, 297
- File Type Association declaration, 260

fileTypeFilter property, FileOpenPicker object, 177

Filled view state, 95–98

fill effect, creating for video, 170

fills

- in Canvas, 201
- in SVG, 195

Fixed Layout App template, 7

Flexbox (CSS), 104–107

Flickr, 295

flip effect, video element, 168

flipping images. *See also* transforms

- using BitmapTransform in Windows. Graphics.Imaging, 211

fluid animations, 213

FolderPicker object, 182, 263–265

fontDisplay class, 150

font-family property, 120, 126, 138

font property, 126

fonts

- animating icon fonts, 219
- Calibri, 123–124
- Cambria, 122
- CSS3 Web Fonts, 124–128
- for use in Windows Store apps, 120
- icon fonts in Windows Store app, 147–153
- licensing, 126
- Segoe UI, 120

font-size property, 151, 542

font-style property, 126

font-weight property, 126

footer element, 538

forEach method, Array object, 513

for-in loop (JavaScript), iterating over arrays, 513

for loop (JavaScript), iterating over arrays, 513

forms, creating, 43

frac Opentype feature, 134

fr sizing construct in CSS, 100

fullscreen-portrait media query, 103

Full Screen view state, 95

functions

- delegates and, 435–440
- creating WinRT delegates from C++ 11 lambdas, 439–451
- JavaScript, 520–525
- arguments, 522
- call and bind, 523
- closures, 525

function scope in JavaScript, 526

Future Access List Service, 264

G

games, ratings board certificates for, 466

geolocation support in Win8-style apps, 387

Geolocator object, 387

- positionchanged event, 391

get and set methods

- creating properties from, 73
- for WinRT class properties, 420

Get-AppxPackage command, 4

getBitmapAsync method, 320

getCostType function, 271

getFileAsync function, 254, 259

getInternetConnectionProfile method, NetworkInformation, 269

GET, PUT, POST HTTP methods, 273

getStorageItemsAsync method, 320

global positioning system (GPS) data, 387

global scope in JavaScript, 526

glyphs, 136

- badge, 349
- character mappings to, in Heydings font, 148
- in icon fonts, 147

Google Web Fonts project, 126

gradients

- creating radial gradient with Canvas, 200
- creating radial gradient with JavaScript, 197
- radial gradient defined in svg element, 195

graphics

- HTML5, with SVG and Canvas, 193–206
- pixel manipulation with Canvas, 206–209
- pixel manipulation with Windows. Graphics.Imaging, 209–212

Grid App template, 7

- data.js file, asynchronous data model, 62

Grid Layout (CSS3), 100, 115

- adaptive layouts for application context, 103

groupHeaderTemplate property, 61

grouping

- binding list data, 58–59
- creating grouped ListView, 376
- group headers for ListView control, 60

gyrometer sensor, 403

H

handles

- Platform::Array, 432

- Platform::String, 430
- WinRT objects and, 422–424
- handle-to-object (^) type modifier, 423, 436
- hangs test, 460
- hasKey function, 227
- head element, style block in, 554
- header element, 538
- headers, selectors for, 127
- Heydings, 148
- high-definition resolutions, supporting with
 - media queries, 91
- history, navigation, 244
- hoisting, 526
- homePage.html file, 10
- homePage.js file, 11
- hover transition with rotation, 217
- HRESULTS, 417
 - C++/CX exception types encapsulating, 418
 - E_BOUNDS HRESULT, 417
- HTML
 - binding object to set of HTML elements, 43
 - HTML Tool Palettes (in Blend), 16
 - navigation, 24
 - separation of JavaScript code from, 503
 - shareTarget.html file, 312
 - sharing HTML data, 306
 - stripping out dynamic HTML returned by XMLHttpRequest, 31
 - using for app content and structure, 534–541
 - web content, 285–286
 - WYSIWYG Design for HTML (in Blend), 16
- HTML5, 9
 - data-* attributes, 10
 - DOCTYPE, 502
 - documentation, 63
 - elements as controls, 63
 - Geolocation API, 387
 - graphics with SVG and Canvas, 193–206
 - media elements, 156
 - new features in, 535
 - data attributes, 539–541
 - media markup, 538
 - semantic markup, 536
 - state management facilities, 225
- IAsyncAction interface, 444
- IAsyncInfo interface, 444
- IAsyncOperation interface, 444
- IAsyncOperation<T> interface, 445
- icon option property, 146
 - AppBar control, 140
- icons
 - animating icon fonts, 219
 - icon fonts in Windows Store app, 147–153
 - working with platform iconography, 136–146
- IDL (Interface Definition Language), 444
- id parameters, ICommand object, 65
- id selector (#), 545
- iframe element
 - hosting remote HTML content, 286
 - web context and, 287–291
- IDataAdapter interface, 54
- IDataSource interface, 54
- illuminanceInLux reading, light sensor, 397
- images
 - Canvas image inverter, 207
 - file picker for, 175–178
 - on live tiles, 340
- Imaging API, 209–213
- img tags, 155
 - WinJS animation of, 221
- !important CSS rule, 557
- importScripts function, WorkerGlobalScope object, 354
- in-app purchases, 488–495
 - creating in-app purchase functionality, 489–493
 - defining offers in store submission process, 494
- inclinometer sensor, 403
- IndexedDB, 225
- initializers, 51–53
- inline styles, 553
- input element
 - new types in HTML5, 535
 - settings for video, 161
- instance properties and methods, 73
- instant commit, settings panels, 234
- Interactive Mode (Blend), 16
- Interface Definition Language (IDL), 444
- Internet (Client) capability, 268, 340
- Internet connection, information on, 269
- isAutoCollapseEnabled property, ad controls, 479
- isAutoRefreshEnabled property, ad controls, 479
- itemDataSource property, setting for ListView, 19

_itemInvoked handler, 302
 itemsPage.html page (Split App template),
 34
 showing real data, 37
 itemTemplate property, 61
 IUnknown interface, 406

J

JavaScript, 9, 501–532, 533
 adding C++ component to JavaScript
 application, 407–409
 arrays, 54, 512–514
 calling WinRT class method from, 415
 Canvas API, 200
 concurrency, 451
 Debug output, 525–526
 execution by Blend, 16
 functions, 520–525
 interaction with audio and video
 elements, 161
 interaction with programs in C/C++,
 405
 object prototypes (classes), 514–520
 objects, 510
 operators, 508–509
 projecting WinRT constructs into, 407
 delegates and functions, 435–440
 WinRT classes, 413–421
 WinRT objects, 421–424
 scoping, 526–529
 selecting and manipulating SVG via, 197
 separation of code from HTML, 503–504
 serialization, 531
 shareTarget.js file, 313
 showing search results, 301
 struct mode, 529–530
 updating tiles for apps, 336
 values and types, 507–508
 Windows Library for JavaScript SDK
 classes, 136
 Windows Store app project file for, 8
 WinJS activation, 505
 WinRT and JavaScript environment,
 411–413
 WinRT asynchronous operations projected
 into, 445
 WinRT events, accessing, 442
 WinRT types in C++/CX and JavaScript,
 424–435
 JavaScript Console (VS2012), 14
 “Josh’s List”, 53

JSON (JavaScript Object Notation), 59, 531
 parsing results from WinJS.xhr, 275
 JSON object, 227
 JSON.parse function, 227, 250
 JSON.stringify function, 227, 250

K

keyboard interaction, 366
 supporting, 379
 keyframes, defining in CSS, 218
 KnownFolder enumeration, 210
 KnownFolders object, 259

L

lambdas, 435, 525
 C++11 lambdas, 436–439
 concurrency and asynchrony, 446
 creating WinRT delegates from C++11
 lambdas, 439–451
 LINQ-oriented C#, 521
 landscape mode, 93
 view states for apps in, 95
 last rule (in CSS), 555
 Latin characters, Heydings font values, 148
 launch attribute, toast element, 361
 layout, 79–118
 creating adaptive UIs with CSS and WinJS,
 104–115
 responding to layout changes in
 JavaScript, 113–115
 using CSS Flexbox, 104–107
 using CSS Multi-Column Layout,
 107–110
 taming the device matrix, 79–98
 layouts in Windows 8, 81
 orientations, 93–95
 view states, 95–98
 Windows 8, choice without device
 tyranny, 81
 working with screen sizes, 84–93
 using CSS layout capabilities to adapt
 your app, 99–104
 Layout Simulator (Blend), 16
 letter-spacing property, 130
 libraries, 256–265
 enabling library-related capabilities in
 manifest file, 257
 enumerating files from, 259
 file activation, 259–261
 file pickers, 261–265

- licenseInfo object, licensechanged event, 488
- LicenseInformation element, 486
 - ExpirationDate child element, 487
- LicenseInformation object, 491
- licensing, fonts, 148
- lifetime, 238–252
 - sessions, 242–246
 - debugging, 246–249
 - WinJS lifetime event helpers, 241–242
 - WinRT app lifetime states and events, 239
- lifetime management (C++), 437
- liga OpenType feature, 135
- light sensors, 396–397
- line element, 196
- link element, 10
- LinkUri element, 484
- ListLayout, 115
- List object, 53
- lists
 - binding, 53
 - grouping binding list data, 58
 - list property, FileOpenPicker object, 176
- Listview control
 - adding via Blend, 18
 - setting itemDataSource property, 19
 - binding to data source, 19
 - bound to dataSource property of items object, 53
 - containing contacts, 328–330
 - creating adaptive collections with, 111–112
 - displaying images selected in multi file picker, 180–192
 - grouped, 376
 - grouping a binding list, 58
 - invoke handler on, 25
 - responding to layout changes in
 - JavaScript, 113–115
 - SemanticZoom and, 375
 - templates for, 60
 - touch interactions, 374
 - touch, mouse, and keyboard interactions with, 379
 - updating as binding list updates, 54
- live tiles, 333–348
 - for lock screen apps, 356
 - scheduled tile updates, 342
 - secondary tiles, 343–348
 - small and large tile updates, 338
 - tile images, 340
 - tile peeking, 340
 - tile updates, 335–338
 - your app's tiles, 335

- local and web contexts, permission differences, 288
- localFolder property, ApplicationData class, 253
- localization, captions for video, 166
- local machine, debugging from, 14
- local, roaming, and temp objects, Application object, 255
- localSettings property, ApplicationData class, 227
- local storage, 225
- local testing, preparing your app for, 457
- location data, working with, 387
 - Bing Maps, 391
 - simulating location information, 393–394
 - using Windows.Devices.Geolocation.
 - Geolocator object, 387–390
 - watching for location changes, 390
- location object, 388
 - WinJS Navigation service, 244
- locationStatus property, 388
- lock screen apps, 355–357
- Lock Screen Settings panel, 356
- logo images, 2
- “Lorem Ipsum” text generators, 214
- LPCWSTR-based APIs, 429
- lux value readings, light sensor, 397

M

- magnetic north, 400
- MakeApp.exe tool, 4
- making money on your app, 453–500
 - ads in your app, 476–482
 - design for monetization, 495
 - in-app purchases, 488–495
 - marketing and managing your app, 496–499
 - preparing for submission, 454–463
 - submitting app to Windows Store, 463–476
 - trial mode, enabling, 483–488
- Manifest Designer (Visual Studio 2012), 9
- manifest file, 2
 - example, 3
 - format and resources test for, 459
- Map control, hosting in Windows Store app, 288–291
- marker element, 196
- marketing and managing your app, 496–499
 - getting paid, 498–499
 - getting your app featured in Windows Store, 496

- marketing and managing your app (*contd.*)
 - tracking your app from Store dashboard, 496
 - Math.PI * 2, 201
 - media, 155–192
 - and pixel manipulation, 193
 - audio and video, 155–156
 - getting started with, in Windows 8, 156–159
 - making application connectable with Play To, 189–191
 - markup in HTML5, 538
 - styling and creating custom controls, 159–163
 - working with captured media, 185–189
 - working with libraries via file picker, 175–180
 - media-based ads, 477–480
 - MediaCapture API, 189
 - MediaCapture object, 185
 - preparing for recording, 382
 - MediaControl object, 173
 - media queries (CSS)
 - checking app orientation with, 95
 - fullscreen-portrait media query, 103
 - supporting high-definition resolutions with, 91
 - tweaking styles for Snapped and Filled views, 97
 - using to adapt to resolutions, 86–91
 - member enumeration with dot (.) operator, 413
 - MessageDialog class, 64
 - metered networks, responsiveness to, 272
 - methods
 - custom control, 73
 - defining for WinRT class, 415
 - WinRT classes, 413, 414
 - and exceptions, 416–419
 - Microsoft Ads SDK, 453
 - Microsoft Advertising, 477
 - Microsoft Blend for Visual Studio 2012. *See* Blend
 - Microsoft.Maps namespace, 290
 - mix method, 76
 - mobile computing, 366
 - mobile networking, 269–272
 - Model-View-ViewModel (MVVM), 48
 - modules (in JavaScript), 527
 - mouse, 366
 - supporting mouse and keyboard interactions, 379
 - ms-appx URL format, 340
 - ms-appx-web prefix to URLs, 289
 - msAudioCategory attribute, audio element, 171
 - BackgroundCapableMedia, 172
 - msAudioDeviceType attribute, audio element, 171
 - MSDN
 - guidelines for using text and typography, 120
 - ms-flexbox display property, 105
 - ms-flex-direction property, 105
 - ms-font-feature-settings property, 134
 - ms-grid-columns property, 101
 - using to lay out container elements, 102
 - ms-grid display property, 100
 - ms-grid-rows property, 101
 - using to lay out container elements, 102
 - msHorizontalMirror attribute, video element, 168
 - msPlayToSource property, 191
 - ms-view-state conditions, 97
 - msZoom attribute, video element, 168
 - Multi-Column Layout (CSS), 107–110
 - multimedia. *See* media
 - Multimedia value, msAudioDeviceType attribute, 171
 - multiple languages, one app, 406–407
 - musicLibrary, PickerLocationId enum, 177
 - mutable keyword, using with lambdas, 438
 - mute button, 161
 - MVVM. *See* Model-View-ViewModel
- ## N
- name, reserving for your app, 454, 456
 - namespaces
 - JavaScript, 528
 - WinJS, 528
 - WinRT declarations projected into
 - JavaScript, 412
 - native extensibility, 405–452
 - adding C++ component to JavaScript application, 407–409
 - concurrency and asynchrony, 443–451
 - delegates and functions, 435–440
 - events, 440–443
 - multiple languages, one app, 406–407
 - WinRT and JavaScript environment, 411–413
 - WinRT classes, 413–421

- WinRT objects, 421–424
- WinRT types in C++/CX and JavaScript, 424–435
- natural user interfaces (or NUIs), 366
- nav element, 538
- navigation, 24–28
 - animation of, 223
 - serializing navigation stack and restoring it in session, 251
- Navigation App template, 7
 - default.html file generated by, 9
 - files generated by, 8
- Navigation object, 244
- NetworkInformation class, 269
- networking, 267–292
 - background data transfer, 280–283
 - in WIN JS and WinRT, 29–33
 - mobile, 269–272
 - network capabilities, 267–268
 - syndication, 277–280
 - web content, 284–291
 - XMLHttpRequest object, 273–276
- networkstatuschanged event,
 - NetworkInformation class, 269
- new operator, 421
- noprint class, 386
- Notes to Testers screen, 471
- NUIs (natural user interfaces), 366
- nullptr keyword, 423
- null values, 507
- Number field, WinRT exceptions, 419
- numbers, badge template, 349

O

- Object.create method, 517
- Object.defineProperties method, 73
- object identity, WinRT and JavaScript, 424
- object prototypes (classes) in JavaScript, 514–520
- object references, 406
 - WinRT, passing between C++/CX and JavaScript, 422
- objects
 - binding, 42–51
 - JavaScript, 510
 - WinRT, 421–424
 - and handles, 422–429
- odd and even keywords, 553
- onactivated event, WebUIApplication object, 240
- one-time binding, 42
- oneTime function, 51
- on<eventname> field, 442
- one-way binding, 42
- onquerysubmitted event, 299
- onresize function, 114
- OpenType layout features, 133–135
- operators, JavaScript, 508–509
- options object, 74
- orientations, 93–95
 - changing Flexbox orientation in portrait mode, 106
 - working with simple orientation sensor, 402
- overflow-y property, 83
- overloading of methods and constructors, WinRT support for, 415

P

- package.appxmanifest file, 8
 - declarations, listed, 298
 - declaring device capabilities, 380
 - enabling library-related capabilities, 257
 - FileSavePicker in Capabilities tab, 183
 - format and resources test for, 459
 - Pictures Library, Capabilities tab, 210
 - properties of your app's tiles, 335
 - setting up background audio in
 - Declarations tab, 172
 - toast notifications, 359
 - Webcam in Capabilities tab, 186
- packages
 - creating for your app, 457
 - uploading your app package, 467
- page animations with WinJS, 223
- Page Control item template, 327
- PageControlNavigator control, 10
- Page Control template, 25
- Parallel Patterns Library (PPL), 405, 446
- PascalCased names, 412
- path element, 196
- pausing a download, 282
- payment for your app, 498
- peeking (tile), 334, 340
- People app, 324
- performance
 - Canvas versus SCG graphics, 204
 - performance test for your app, 460
- PeriodicUpdateRecurrence enumeration, 343
- Permissions flyout, Settings charm, 389
- photo gallery app (example), 81

- photographs
 - built-in Photos app, 295
 - photo mode, CameraCaptureUIMode, 188
 - sharing picture from Photos app, 295, 321
 - PickerLocationId enumeration, 176
 - pickSingleContactAsync function,
 - ContactPicker object, 323
 - pictureLibrary, PickerLocationID enum, 177
 - pinButton object, 347
 - pinch gestures, 379
 - Pin to Start Screen button, 344
 - PixelDataProvider object, 211
 - pixels, manipulating, 206–212
 - using Canvas, 206–209
 - using Windows.Graphics.Imaging, 209–212
 - Platform::Array reference type, 432
 - Platform::ArrayReference type, 432
 - Platform::COMException object, 417, 419
 - Platform::Exception object, 419
 - Platform::Object::ReferenceEquals method, 424
 - Platform::OutOfBoundsException object, 417
 - platform-specific features, apps targeting, 457
 - Platform::String::Data, 431
 - Platform::String reference type, 429
 - Platform::StringReference type, 430
 - Platform::WriteOnlyArray method, 433
 - Play To, 189–191, 296
 - accessing via Devices charm, 190
 - configuring, 190
 - PlayToManager object, 191
 - pointers, C++/CX handles and, 423
 - populateSettings function, SettingsFlyout control, 230
 - PopupMenu class, 65
 - portrait mode, 93
 - positional pseudo-class selector, 552
 - positionchanged event, Geolocator object, 391
 - PositionStatus enum, 388
 - poster attribute, video element, 157
 - invalid URL with, 159
 - POST HTTP method, 273
 - postMessage function, 290
 - posts.html page, adding Print button, 384
 - PPL (Parallel Patterns Library), 405, 446
 - presentation and style, 533–564
 - using CSS for app layout and style, 541–558
 - using for app content and structure, 534–541
 - previousExecutionState property, 245, 246
 - pricing information, 464
 - print capabilities, in-app, 384–387
 - PrintManager object, 384
 - private keyword
 - private members of WinRT types, 428
 - properties and methods in JavaScript, 73
 - processAll function, 45
 - calling for data-win-control property, 68
 - processPosts function, 274
 - productLicenses object, 491
 - programming languages (multiple), one app, 406–407
 - project templates
 - Create New Template dialog, 20
 - Visual Studio 2012 and Blend, 16
 - Windows Runtime Component, 409
 - Windows Store app, 7
 - Promise object, 30, 281
 - cancel method, 272
 - Promotional Images section, Description page, 496
 - properties
 - CSS, 543
 - custom control, 73
 - WinRT classes, 413, 420–422
 - property property, 216
 - prototypal inheritance in JavaScript, 518
 - prototypes in JavaScript, 515–518
 - pseudo-classes, 138
 - pseudo-class selectors, 549
 - pseudo-elements, 138
 - pseudo-element selectors, 549
 - public keyword
 - C++/CX and, 412
 - WinRT class methods, 414
 - purchases, in-app. *See* in-app purchases
- ## Q
- querySelectorAll function, 197
 - querysubmitted handler, 302
 - queryText, containing search string, 300
- ## R
- radial gradients, 195
 - creating with Canvas, 200
 - creating with JavaScript, 197
 - range control, 161
 - rating certificates, 466
 - Rating control, 82, 370



- readingchanged event, light sensor, 396
- readTextAsync function, 254
- readText function, 256
- ready function
 - adding logic for recording devices, 381
 - contacts page control, 328
 - in page controls, 235
- recording devices, 381–384
- rect element, 196
- Reference Manager dialog, 409
- ref keyword, 407
 - in WinRT class declarations, 414
- ref new operator, 423
- ref struct keywords, WinRT class definitions, 414
- regular expressions in JavaScript, 511–512
- rejection by Windows Store, dealing with, 473
- remote machine, debugging from, 14
- removeEventListener method, 76, 442
- render method, Template object, 60
- reportDataRetrieved method, 321
- reportError function, 319
- reportInterval property, sensors, 396
- reportStarted method, 321
- requestCreateForSelectionAsync function, 347
- requestProductPurchaseAsync method, 491
- resize event listener, 115
- resolutions
 - high-definition, supporting with media queries, 91
 - testing apps in Win8 simulator, 84
 - using CSS media queries to adapt to, 86–91
- resolveItemResource method, 305
- resources
 - app resources validation errors, 461
 - test for manifest file, 459
- resultsuggestionchosen handler, 304
- resuming applications, 243
 - debugging resume, 246
 - resume event, 242
 - resuming event, 244
 - sessionState object and, 251
- retrieveFeedAsync function, 32
- roamingFolder property, ApplicationData class, 253
- roaming object, Application object, 255
- roaming settings, 228
- rotation, 214
 - hover transition with clockwise rotation, 217
- RSS (Really Simple Syndication), 31, 278

S

- Scalable Vector Graphics. *See* SVG
- scaling, 214
- ScheduledTileNotification object, 342
- scheduled tile updates, 342
 - updating based on multiple URLs, 343
- ScheduledToastNotification object, 362
- scoping in JavaScript, 526–529
 - hoisting, 526
 - modules, 527
 - namespaces, 528
 - WinJS namespaces, 528
- screen edges, touch-friendly apps with, 370–374
- screen resolutions. *See* resolutions
- screenshots of your application, 468
- screen sizes, 84–93
 - high-definition resolutions, supporting with media queries, 91–93
 - using CSS media queries to adapt to resolutions, 86–91
- script element, 10
 - src attribute, 503
- sealed keyword, 414
- Search contract, 297, 297–305
 - implementing search, 298–303
 - search suggestions, 303
 - Share target, 310, 310–316
 - Windows 8 Search panel, 298
- Search Contract item template, 301
- searchResults.js file, 302
- SecondaryTile constructor, 346
- secondary tiles, 343–348
 - activation on application launch, 347
 - confirmation dialog, placement of, 346
 - creating, 344
- security, Windows security features test, 460
- Segoe UI font, 120
 - contrast between Calibri and, 124
 - OpenType layout features, 134
- Segoe UI Symbol, 121, 370
 - characters viewed in Character Map, 139
 - Unicode values in, 141
- selectionChanged handler, 307
- selectors, 138, 542, 543–553
 - advanced, 548–551
 - CSS3, 551–553
- Selling details screen, submission process, 464
- semantic markup, 536

- SemanticZoom control, 375–379
 - adding to apps, 376
 - creating, 377
 - support of pinch and stretch gestures, 379
 - touch, mouse, and keyboard interactions with, 379
- sensor boards, third-party, 395
- sensors, working with, 394–403
 - accelerometer, 398
 - compass, 400
 - light sensors, 396–397
 - orientation sensor, 402
 - other, 403
- serialization, 531
- sessions, 242–246
 - debugging, 246–249
 - WinJS session helpers, 250–252
- sessionState object, 250–252
- session state, saving, 244
- session storage, 225
- setOptions method, 74
- settings, 226–238
 - local, 227
 - roaming, 228
 - Settings charm, 228–238
 - Microsoft guidelines for settings panels, 234
 - Permissions flyout, 389
- settings event, Application object, 230
- SettingsFlyout control, 230, 233
- SettingsPane object, 229
- Settings search provider, 297
- shaken event for accelerometer, 399
- Share charm, 295
- Share contract, 305–322
 - accessing shared data, 316–321
 - data types supported, 306
 - reporting sharing progress, 321
 - sharing a selected item, 309
- ShareOperation object, 315
 - reportCompleted method, 316
 - reportError function, 319
- Share Target Contract item template, 312
- shareTarget.html file, 312
- shareTarget.js file, 313
- shell contracts, 293–332
 - Contacts contract, 322–330
 - contracts, 295–297
 - Search contract, 297–305
 - Share contract, 305–322
 - Windows 8 shell, 294–295
- shell integration, 333–364
 - background tasks, 350–358
 - badges, 348–350
 - live tiles, 333–348
 - toast notifications, 358–363
- shimmer effect, SVG animation with
 - JavaScript, 198
- shimmer function, 197
- showAsync method, MessageDialog class, 64
- sideloading requirements, 457
- SignTool.exe tool, 4
- SIL Open Font License, 148
- SimpleOrientation enum, 402
- SimpleOrientationSensor object, 402
- simulator
 - debugging from, 14
 - location simulation feature, 393
 - sample app running from, 15
 - using to capture app screenshots, 470
- skewing, 214
- SkyDrive app, 263
 - integration with file pickers, 265
- sliders, setting up for video playback, 163
- small caps (smcp) OpenType feature, 134
- smartphones, 366
- SmtplibClient object, 295
- Snapped view state, 95–98
- sorting binding lists, 56–57
- span element
 - displaying current volume for video, 162
 - using for recording device, 381
- specificity in CSS, 556
- splash screen, 2
 - display during activation of app, 240
- Split App template, 7, 34–38
 - data.js file, asynchronous data model, 62
 - snapped mode support, 39
- split.js file, 307
- SQLite, 226
- square size, 339
- src property, 126
- stack field, WinRT exceptions, 419
- StandardDataFormats enumeration, 319
- Start Screen, 294
 - live tiles, 333
 - sample app installed into, 5
 - secondary tiles, pinning to, 344
 - zoomed-out view of, 375
- state objects, WinJS Navigation service, 244
- static properties and methods, 73
- std::vector, 432

- std::vector<T>, 451
 - std::wstring, 429
 - step attribute, input element, 162
 - stops element, 198
 - changing stop-color attributes, 203
 - StorageFile object, 177, 261
 - StorageFolder object, 264
 - Store API and simulator, 483–485
 - store logo, 2
 - stretch gestures, 379
 - String constructor, 431
 - stringify function, 227
 - StringReference class, 430
 - strings, 422, 427
 - WinRT types in C++/CX and JavaScript, 429–431
 - stroking
 - in SVG, 196
 - modifying stroke properties in SVG, 197
 - stroke effects with Canvas, 201
 - struct mode in JavaScript, 529–530
 - structs
 - struct keyword in WinRT type declarations, 414
 - WinRT types in C++/CX and JavaScript, 425
 - <style> blocks for pages, 553
 - stylistic sets (ssXX tag), 135
 - submission of apps to Windows Store
 - preparing for, 454–463
 - submitting your app, 463–476
 - substring attribute selectors, 551
 - subtitles, adding to video, 163–170
 - suggestions for search, 303
 - suggestionsrequested event, 303
 - Supported Windows 8–style API test, 460
 - suspended applications, 242
 - debugging suspend and resume, 246
 - suspend event, 239, 242
 - WinJS sessionState object and, 251
 - svg element, 194, 538
 - SVG (Scalable Vector Graphics), 193, 194–199
 - choosing between Canvas and, 204
 - compass, 400
 - HTML5 graphics with SVG and Canvas, 193
 - selecting and styling elements, 197
 - syndication, 277–280
 - SyndicationClient class, 32, 277
 - showing contents of RSS feed with, 33
 - System.Graphics.Imaging, 212
 - system-provided search providers, 297
 - %SystemRoot%\WinMetadata winmd files, 407
 - system trigger types and requirements, 355
- ## T
- tablet devices, 366
 - task model, PPL, 446, 449
 - templates
 - badge, 349
 - tile, 336
 - using in data binding, 60–63
 - temp object, Application object, 255
 - temporaryFolder property, ApplicationData class, 253
 - terminated applications, resuming, 243
 - terminate event, 239
 - testing
 - Notes to Testers screen, 471
 - preparing your app for local testing, 457
 - text-based ads, working with, 480
 - text-shadow property, 132
 - adding to icon font, 153
 - text-transform property, 131
 - this keyword
 - in JavaScript, 510
 - lambdas and, 439
 - this-> qualification, lambda member access and, 439
 - this variable, 353, 523–525
 - thumbnails, previewing media in file picker, 176
 - tileId, 348
 - TileNotification object, 338, 342
 - tiles. *See* live tiles
 - tileSquarePeekImageAndText01 template, 341
 - TileTemplateType enumeration, 336
 - Tile Update Manager (TUM), 336
 - notification queue, 343
 - tileWideImageAndText01 template, 336, 338
 - Timed Text Markup Language (TTML), 163
 - TimePicker control, 369
 - timer, selecting for background task, 355
 - TimeTrigger object, 357
 - timing functions, 217
 - animation-timing-function property, 219
 - toast element, 361
 - ToastNotificationManager object, 359
 - toast notifications, 358–363
 - app activation from toast, 361

- toast notifications (*contd.*)
 - scheduled toast, 362
 - ToggleSwitch control, 234, 369
 - Tom8to app, 463
 - touch, 366–380
 - building apps with screen edges, 370–374
 - creating touch-friendly interactions with SemanticZoom, 374–379
 - HTML controls, 367–369
 - supporting mouse and keyboard interactions in Win8 apps, 379
 - WinJS controls, 369–370
 - touch-first, 379
 - track element
 - placing inside video element, 164
 - srclang and label attributes, 166
 - transform functions, 215
 - transforms (CSS), 214
 - animating, 217
 - resources for information, 215
 - SVG compass rose, 401
 - transition property, 217
 - transitions
 - between pages, 223
 - CSS, 215–217
 - CSS animations and, 218
 - SVG compass rose, 401
 - WinJS wrapper functions for, 222
 - translation, 214
 - in CSS animation, 219
 - trial details, 464, 465
 - trial mode, enabling in your app, 483–488
 - simulating and testing trial functionality, 485
 - triggering background tasks, 351–352
 - triggers for background tasks, 355
 - true north, 400
 - truth values in JavaScript, 509
 - TTML (Timed Text Markup Language), 163
 - TUM (Tile Update Manager), 336
 - notification queue, 343
 - Twitter, 295
 - two-way binding, 42
 - type modifier (^) in C++/CX, 422
 - typeof operator, 507
 - types
 - JavaScript, 507–508
 - object prototypes (classes) in JavaScript, 514–520
 - WinRT class method called in JavaScript, 415
 - WinRT declarations projected into JavaScript, 412
 - WinRT types in C++/CX and JavaScript, 424–435
 - arrays, 431–433
 - strings, 429–431
 - value types, 433–435
 - type selector, 545
 - typography, 119–154
 - icon fonts in Windows Store app, 147–153
 - in Windows Store apps, 119–135
 - Calibri font, 123–124
 - Cambria font, 122
 - CSS3 Web Fonts, 124–128
 - Segoe UI font, 120
 - tweaking with CSS, 129–135
 - Microsoft guidelines for, 120
 - working with platform iconography, 136–146
- ## U
- UICommand object, 65
 - ui-dark or ui-light WinJS stylesheets, 369
 - ui.js file, 222, 506
 - unbind method, 51
 - undefined values, 507
 - Unicode
 - values in Heydings icon font, 148
 - values in Segoe UI Symbol font, 138, 141
 - universal selector (*), 544
 - updateLayout method, page object, 115
 - update process for Windows Store apps, 474
 - Uri class, 32
 - URL object, 177
 - URLs
 - ms-appx URL format, 340
 - ms-appx-web prefix, 289
 - objectUrl for audioFile, assigning to audio element, 384
 - using object URLs for file system resources, 177
 - user experience (UX) practices, 301
 - guidelines for Windows Store apps, 220
- ## V
- value keyword, 407
 - values in JavaScript, 507
 - values property, ApplicationData class, 227
 - value types, WinRT types in C++/CX and JavaScript, 433–435

- var keyword, 507
- vertical-align property, 138
- video
 - CameraCaptureUIMode, 188
 - formats supported in Win8 Windows Store apps, 156
 - using file picker for, 178
- video element, 156, 539
 - attributes, 157, 158
 - Microsoft's extension effects, 167
- videoLibrary, PickerLocationID enum, 177
- viewMode property, FileOpenPicker object, 176
- views, application, 113
- view states, 95–98, 115
 - controlling in Blend, 97
- Visual Studio 2012
 - adding SDK references, 392
 - Blend integration with, 16
 - creating instance of Search Contract item template, 301
 - debugging contract providers, 332
 - debugging suspend and resume, 247
 - getting started in, 6–15
 - debugging tools, 14
 - Manifest Designer, 9
 - Windows Store app project templates, 7
 - WinRT and, 33
- volume
 - adjusting and displaying for video, 162
 - adjusting for videos, 161

W

- W3C
 - CSS3 Flexible Box Layout specification, 104
 - CSS3 Grid Layout specification, 100
 - CSS3 specification, 87
 - SVG 1.1 2nd Edition specification, 194
 - TTML and WebVTT, 163
 - video tag, overview of, 157
- WACK (Windows App Certification Kit), running, 458–463
- warning icon, 149
- web content, 284–291
 - HTML, 285
 - web context, 287–291
- web fonts, 124–128
- web platform, 9
- WebUIApplication object, 239
- WebUIBackgroundTaskInstance object, 353

- WebVTT (Web Video Text Tracks), 163–170
- Wi-Fi connections, 270
- win-backbutton class, 136
- winControl property, 60
 - custom clock control, 71, 74
 - DatePicker control, 67
- window object, 526
- window.onresize function, 114
- Windows 8, 79
 - availability on different devices, 81
 - screen resolutions supported, 84
- Windows 8 Ads SDK, 477
- Windows 8 PowerShell, appx module, 4
- Windows 8 shell, 294–295
- Windows App Certification Kit (WACK), running, 458–463
- Windows.ApplicationModel.Background namespace, 352
- Windows.ApplicationModel.Contacts namespace, 323
- Windows.ApplicationModel.DataTransfer namespace, 305
- Windows.ApplicationModel.Store namespace, 483
- Windows.Data.Xml.Dom namespace, 283
- Windows DevCenter, information about
 - keyboard and mouse interactions, 379
- Windows.Devices.Geolocation.Geolocator object, 387
- Windows.Devices.Geolocation.PositionStatus enum, 388
- Windows.Devices.Sensors namespace, 394
- Windows.Foundation namespace, 32
- Windows::Foundation::TypedEventHandler, 442
- Windows.Graphics.Imaging, 206
 - pixel manipulation with, 209–212
- Windows.Graphics.Printing.PrintManager object, 385
- Windows Library for JavaScript SDK classes, 136
- Windows Library for JavaScript SDK reference, 9
- Windows.Media.Capture namespace, 185
- Windows.Media.MediaControl object, 173
- Windows Media Player, 191
- Windows Metadata. *See* winmd files
- Windows.Networking.BackgroundTransfer namespace, 280
- Windows.Networking.Connectivity.NetworkInformation namespace, 269

- Windows Runtime. *See* WinRT
- Windows Runtime Component project
 - template, 409
- Windows Runtime Library (WRL), 407
- Windows Simulator folder, Pictures library, 468
- Windows.Storage.ApplicationData
 - namespace, 227, 252–254
- Windows.Storage.KnownFolder
 - enumeration, 210, 259
- Windows Store, 453
 - getting your app featured, 496
 - Store API and simulator, 483–485
 - submitting your app to, 463–476
- Windows Store apps
 - animation in, 212–224
 - audio formats supported in Win8, 159
 - building your first app, 2–6
 - adding and verifying .appx file, 5
 - HTML file, 2
 - metadata and resources, 2
 - CSS and, 558–564
 - network capabilities available for, 268
 - typography in, 119–135
 - using and manipulating icon fonts in, 147–153
 - video element extension effects, 167
 - video formats supported in Win8, 156
 - WebVTT and TTML support, 164
- WindowsStoreProxy.xml file, 483
 - in-app purchase information, 489
 - LicenseInformation element, 486
 - ExpirationDate element, 487
- Windows.UI.Notifications namespace, 336
- Windows.UI.StartScreen namespace, 346
- Windows.UI.ViewManagement namespace, 113
- Windows.UI.WebUI namespace, 239, 353
- Windows.Web.Syndication namespace, 32, 277
- WinJS, 9
 - activation, 505
 - Animation library, working with, 220–224
 - benefits of using, 221
 - element animations, 221
 - page animations, 223
 - application lifetime states and events, 242
 - binding, types of, 42
 - class definitions via, 519
 - controls, 18, 63, 66–69
 - list of, 69
 - creating adaptive collections with
 - ListView, 111–112
 - file helpers, 255–256
 - initializers, 51
 - lifetime event helpers, 241
 - namespaces, 528
 - Navigation object, 244
 - networking in, 29–33
 - Rating control, 82
 - responding to layout changes in
 - JavaScript, 113–115
 - session helpers, 250–252
 - SettingsFlyout control, 230
 - stylesheets for touch controls, 369
 - touch-friendly controls, 369–370
 - WinJS.Application object, 230
 - WinJS.Binding.List object, 18, 182
 - dataSource property, 19
 - WinJS.Binding namespace, 45
 - WinJS.Binding.Template class, 60
 - WinJS.Class namespace, 73, 76
 - WinJS.Namespace namespace, 72
 - WinJS.Promise class, 30
 - WinJS.UI.AppBar, 373
 - WinJS.UI.AppBarCommand, 373
 - WinJS.UI.ListView, 26
 - WinJS.Utilities namespace, 76
 - WinJS.xhr, 273–276
 - parsing JSON results, 275
 - parsing XML results, 274
 - win-listview class, 112
 - winmd files, 406, 409
 - WinRT
 - and JavaScript environment, 411–413
 - application binary interface (ABI), 406
 - application lifetime states and events, 239
 - bringing web platform together with, 9
 - classes, 413–421
 - concurrency and asynchrony, 443–451
 - controls, 63, 64
 - delegates and functions, 435–440
 - events, 440–443
 - Geolocator object, 387
 - networking in, 29
 - objects, 421–424
 - onactivated event, 240
 - projecting constructs into different
 - languages, 407
 - SettingsPane object, 229
 - shaken event, 399

- sydication API, handling RSS and Atom, 279
- types in C++/CX and JavaScript, 424–435
- XmlDocument object, 283
- .win-star class, 370
- WOFF version of Heydings font, 148
- word-spacing property, 130
- WorkerGlobalScope object, 353
 - importScripts function, 354
- writeTextAsync function, 253
- writeText function, 255
- WRL (Windows Runtime Library), 407
- WYSIWYG Design for HTML (in Blend), 16

X

- XAML, 534
- XHR. *See* XMLHttpRequest object
- xhr function, 29
 - showing contents of RSS feed, 32
- XML
 - badge templates, 349
 - parsing results of XHR call, 274

- SVG as, 195
- toast element, 361
- TTML (Timed Text Markup Language), 163
 - use by Tile Update Manager in tile updates, 336, 342
- XmlDocument object, 283
- XMLHttpRequest object, 29, 273–276
 - parsing XML results, 274
 - progress and errors, 274
- XPath, 283
- X-WINS-Expires header, HTTP response, 343

Y

- YAML, WebVTT similarity to, 163

Z

- .zip-compliant appx packaging format, 457
- zoom effect, video element, 168
- /ZW option, C++ compiler, 409