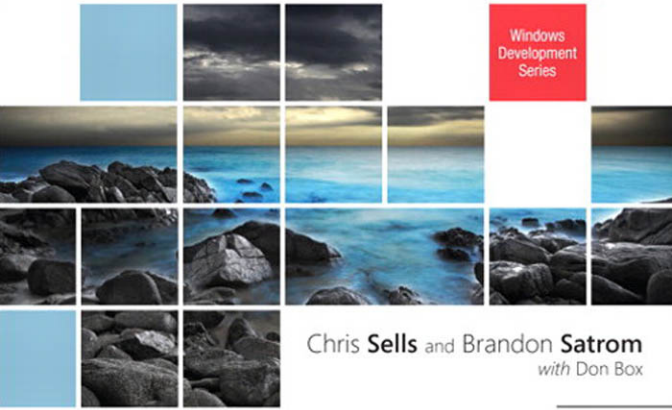
Forewords by Chris Anderson and Rey Bango

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



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

telerik

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."

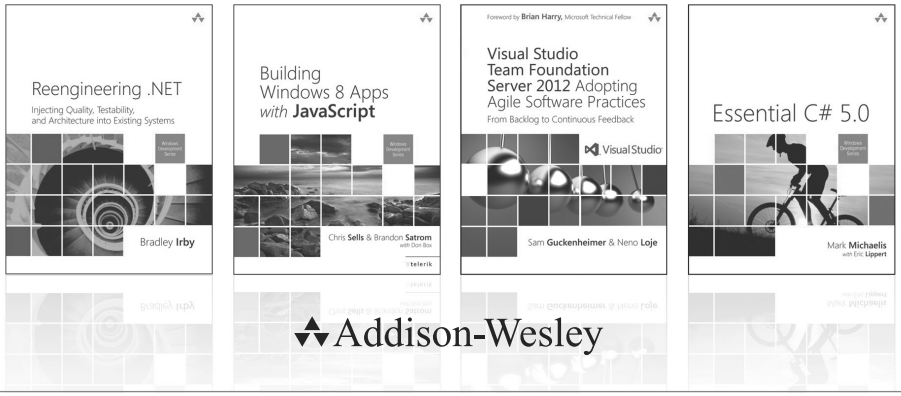
—Derick Bailey, independent consultant, screencaster, speaker, and author, http://mutedsolutions.com and http://watchmecode.net

"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



Addison-Wesley

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

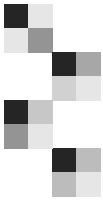
The Windows Development Series grew out of the award-winning Microsoft .NET Development Series established in 2002 to provide professional developers with the most comprehensive and practical coverage of the latest Windows developer technologies. The original series has been expanded to include not just .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



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

Library of Congress Control Number: 2012953216

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

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 manifest[1] 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.

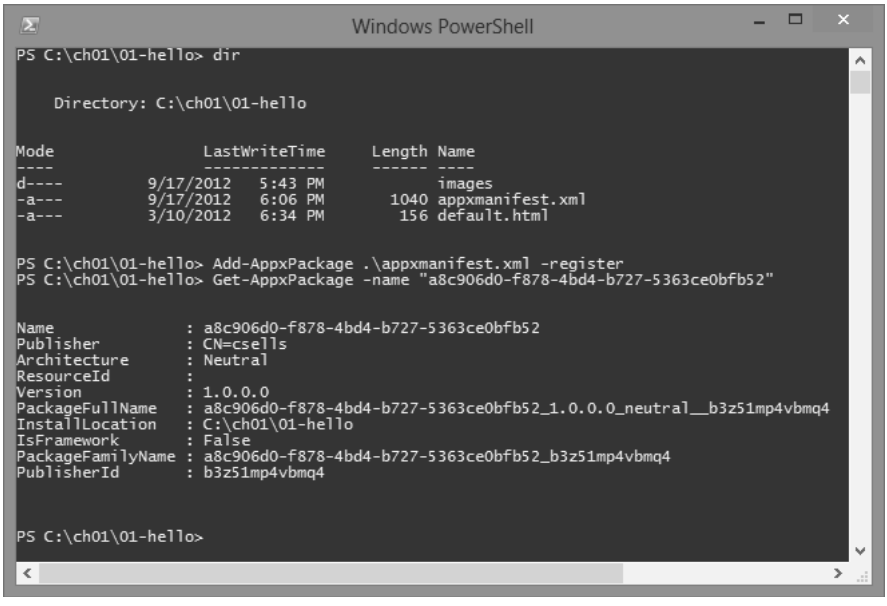


Figure 1.1: Adding an appx file and verifying that it's been added

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 free[6] 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.

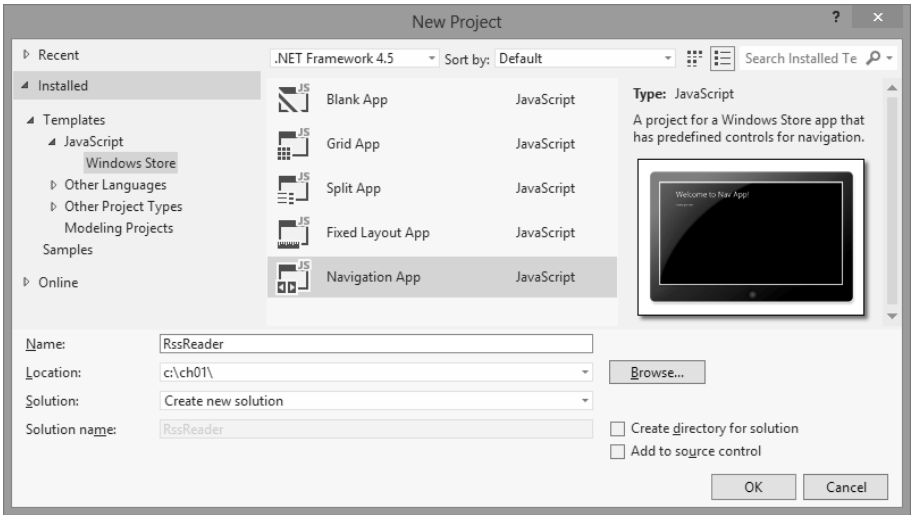


FIGURE 1.3: Creating a Windows Store Navigation App in Visual Studio 2012

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.

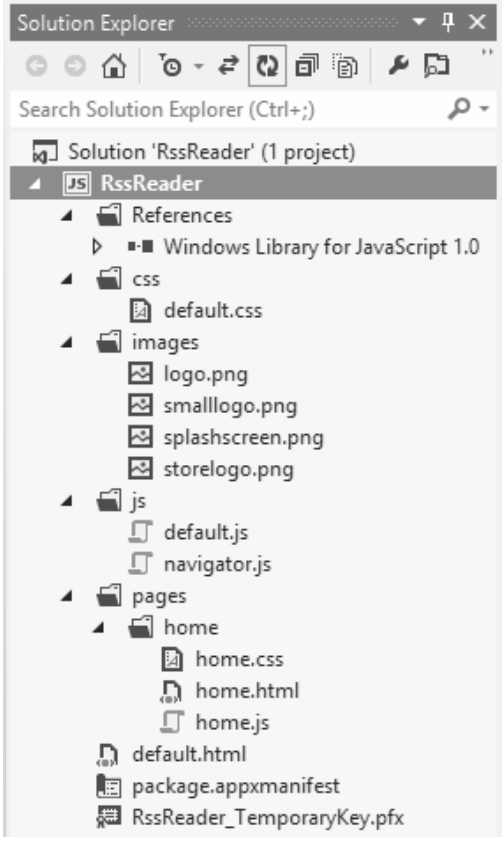


Figure 1.4: The files generated by the Visual Studio 2012 Windows Store Navigation App project template

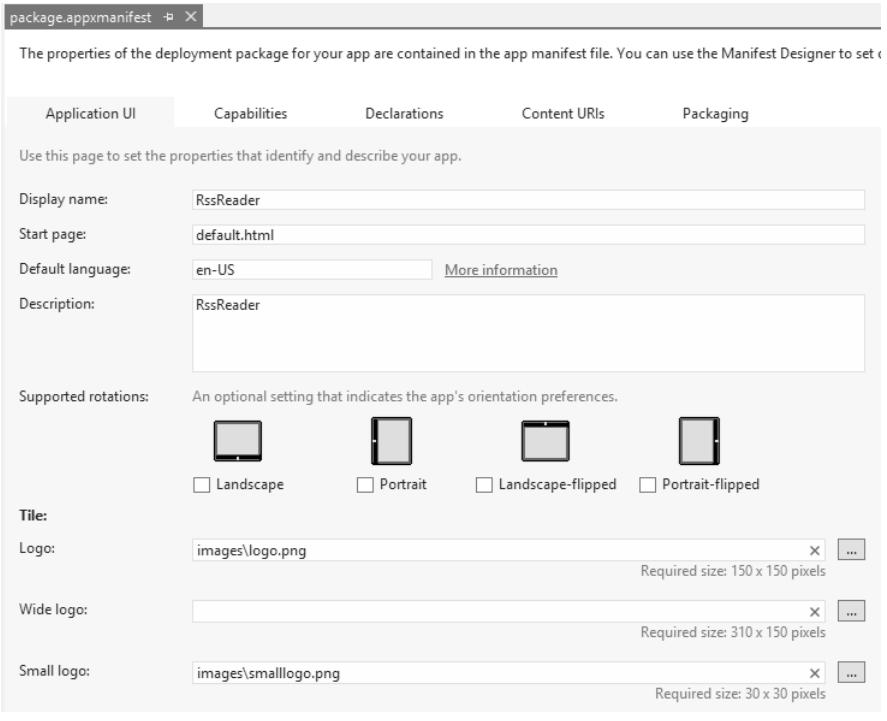


FIGURE 1.5: The Visual Studio 2012 Manifest Designer

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.

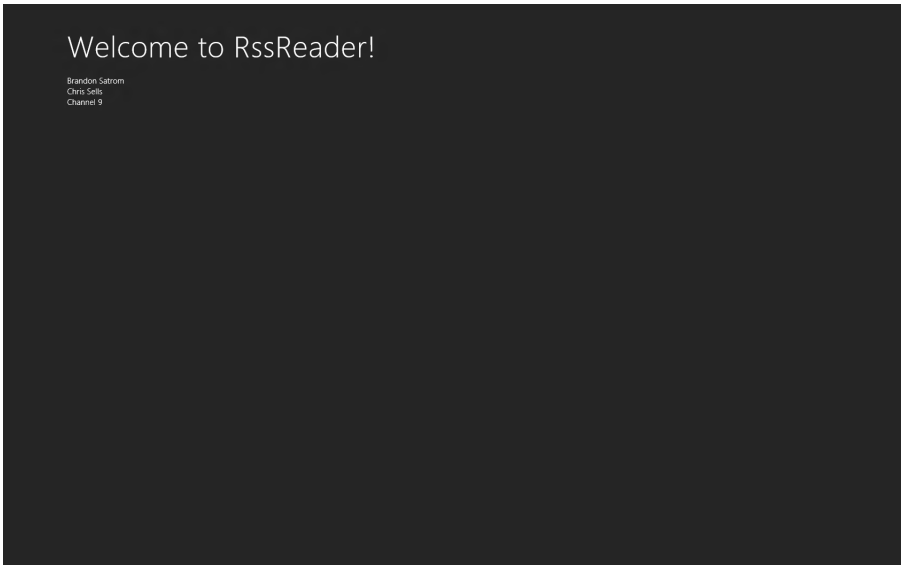


Figure 1.6: A list of feed titles in a Navigation App template project

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.

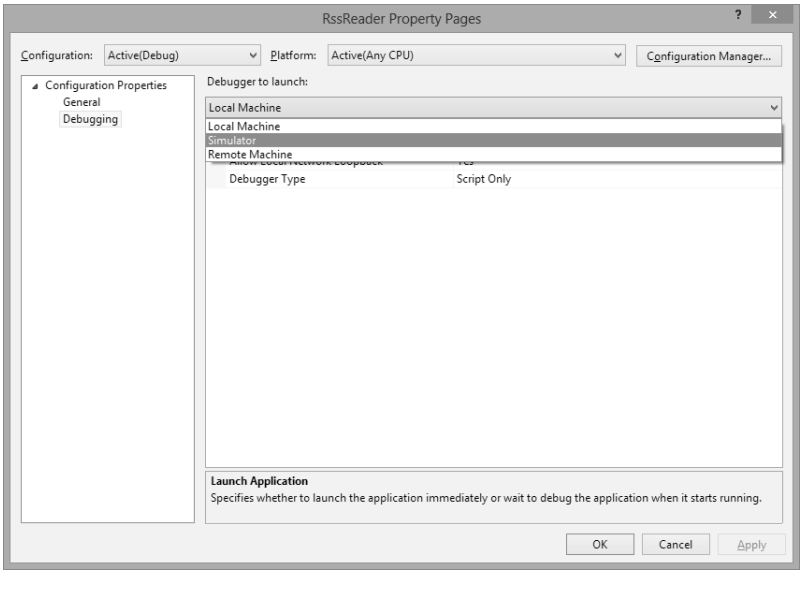


Figure 1.7: Choosing to debug against the local machine, the simulator, or a remote machine

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).

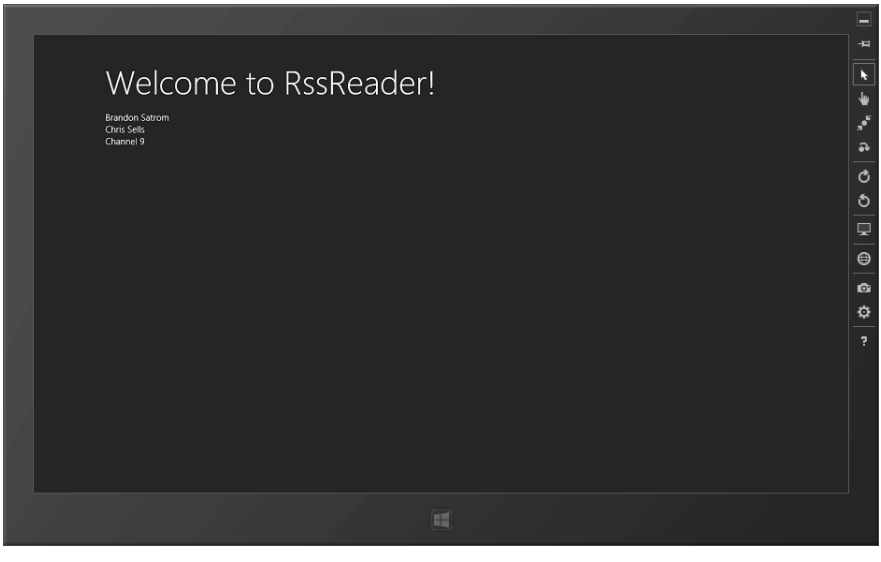


Figure 1.8: A Windows Store app running in the simulator

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.

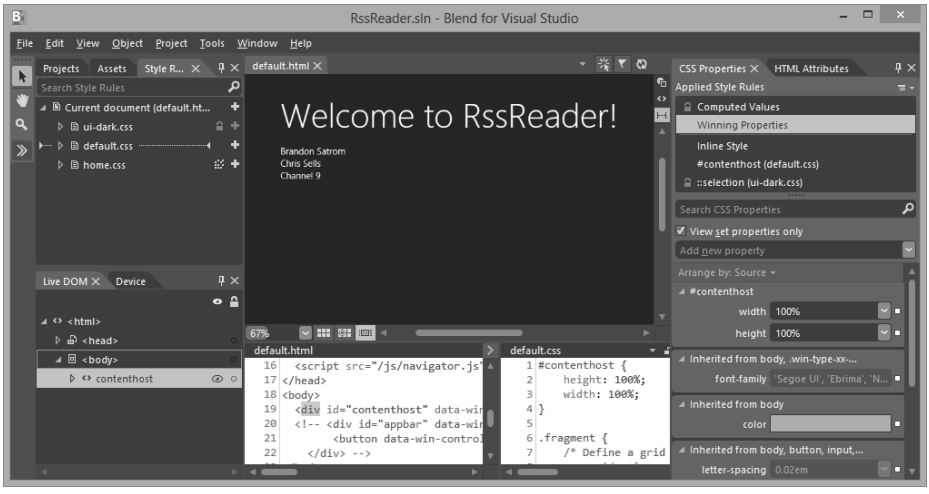


FIGURE 1.9: Microsoft Blend for Visual Studio 2012

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.

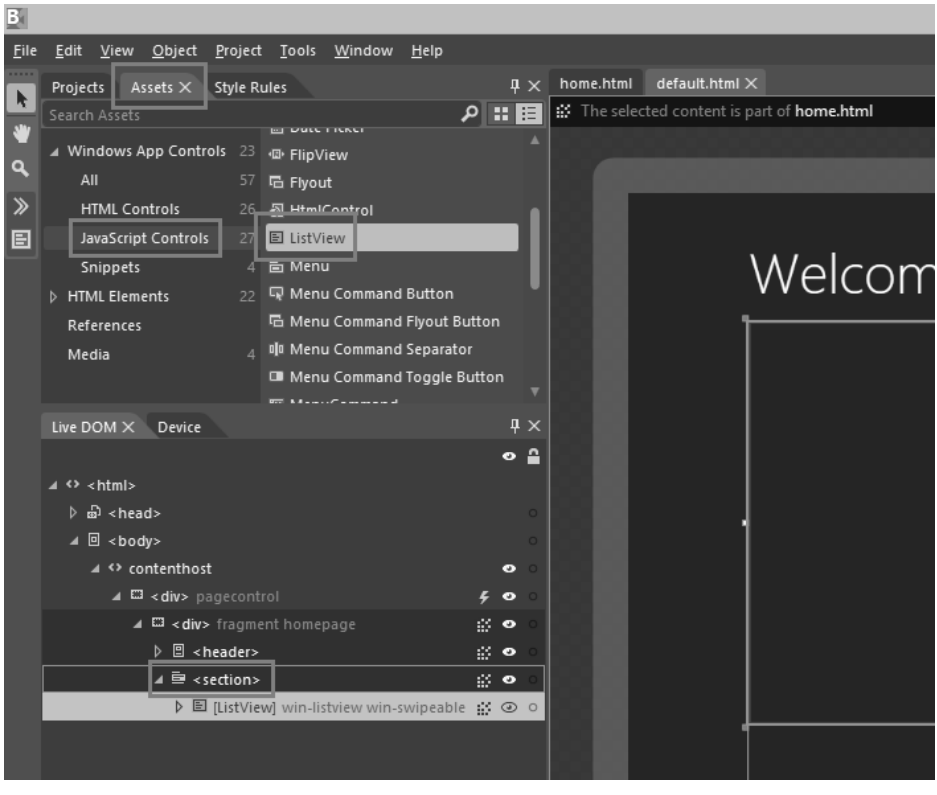


Figure 1.10: Using Blend to add a `ListView` control to a Windows Store app

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.

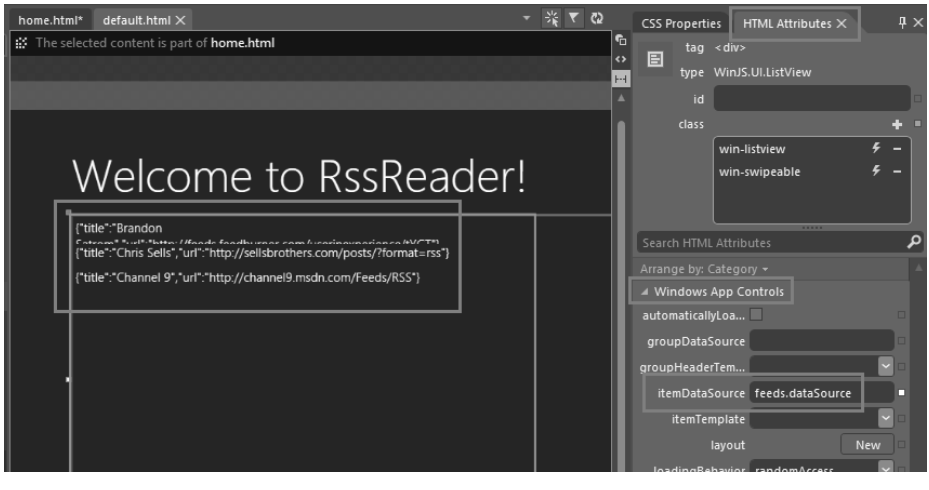


FIGURE 1.11: Using Blend to bind a `ListView` to a list of data

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.

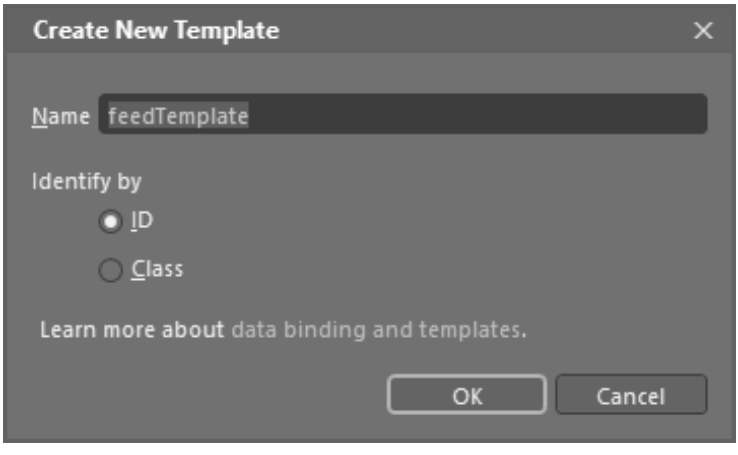


Figure 1.12: Using Blend to create a data template

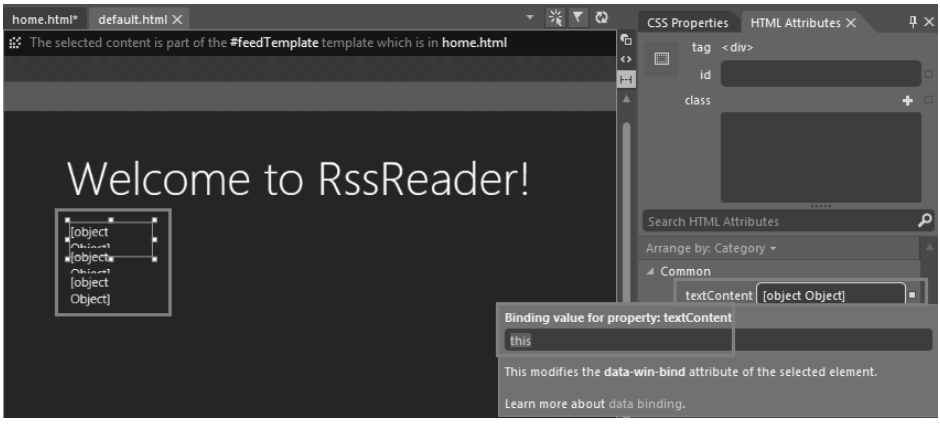


Figure 1.13: Using Blend to examine the contents of a data template

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).

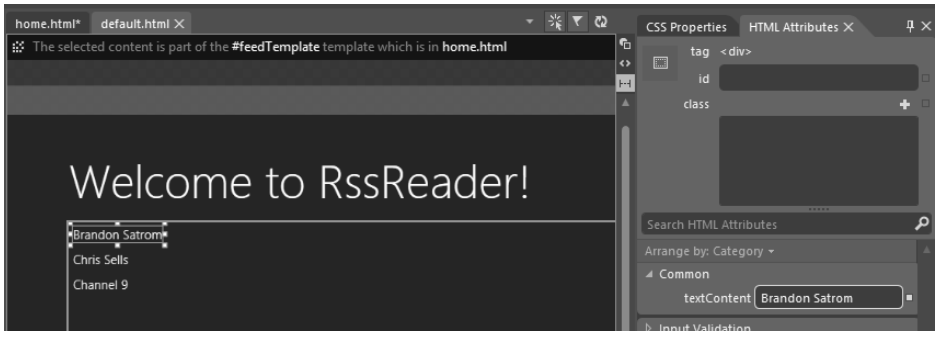


Figure 1.14: Using Blend to modify the contents of a data template

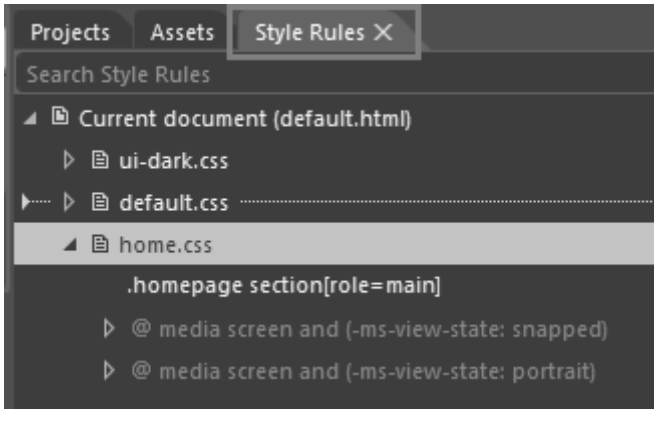


Figure 1.15: Using Blend to manage the styles in your project

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.

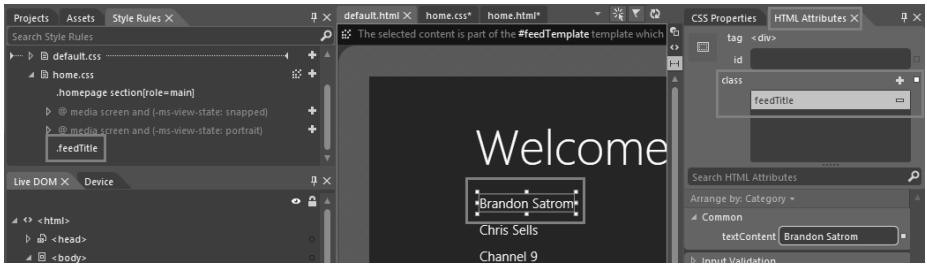


FIGURE 1.16: Using Blend to associate a class with one item from a template associates it with all items from that template.

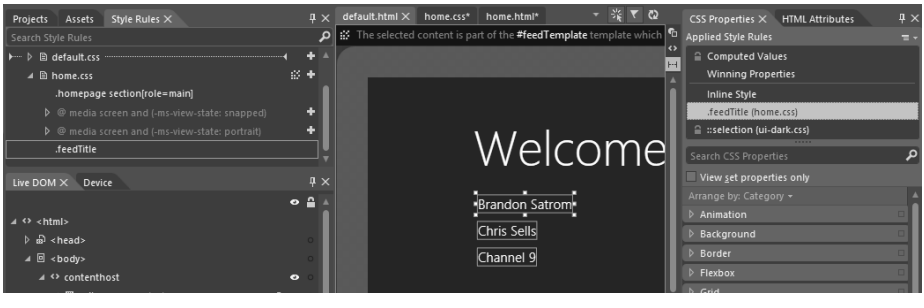


FIGURE 1.17: Selecting a CSS rule in Blend shows all elements to which that rule is applied.

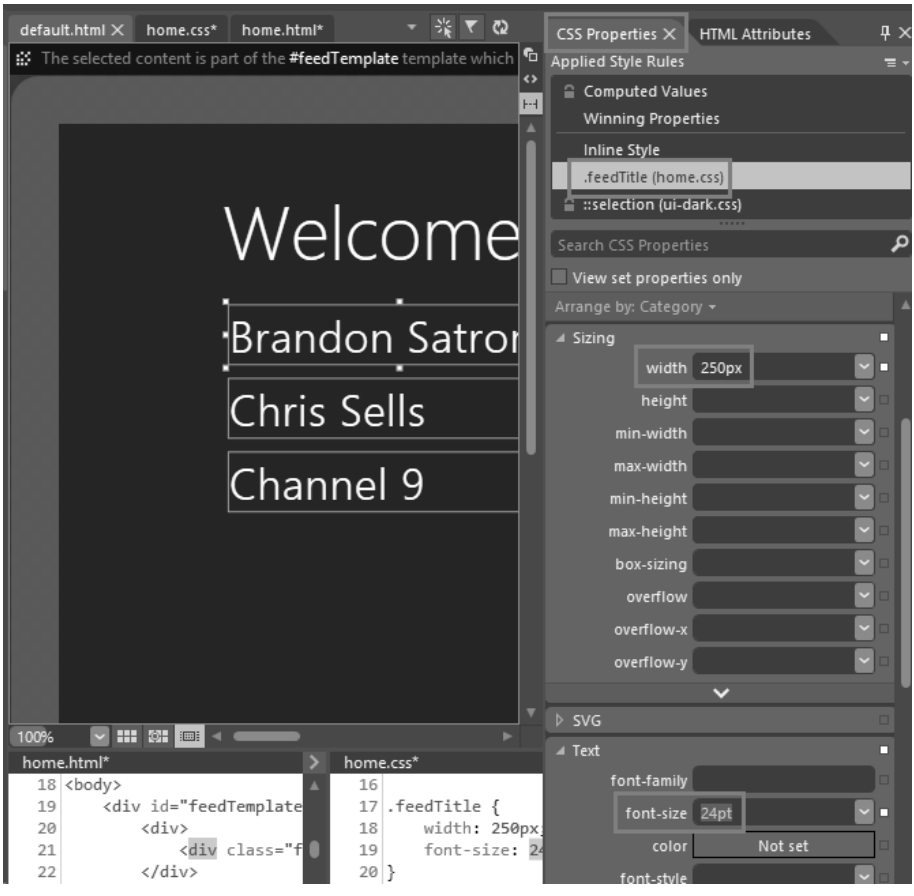


Figure 1.18 Using Blend to change CSS properties interactively

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 Hypertext 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.

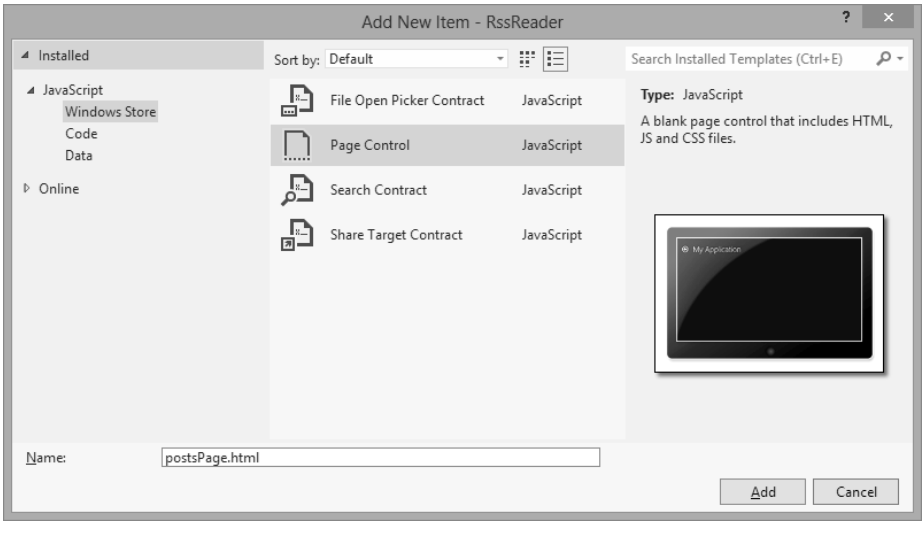


Figure 1.19: The Add New Item dialog for Windows Store apps

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.

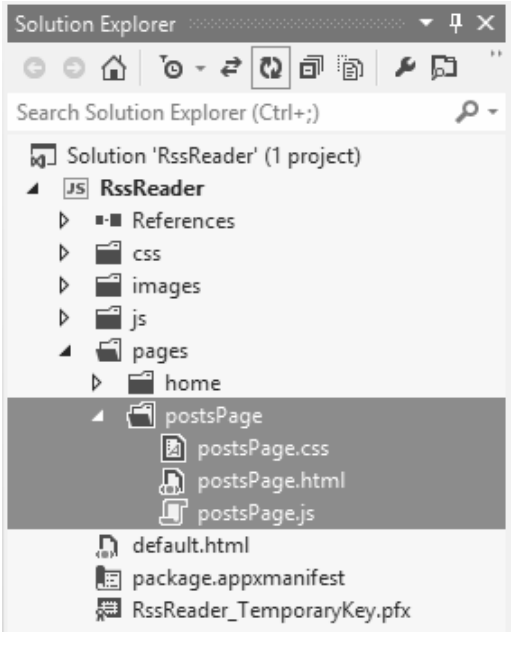


Figure 1.20: A new page control added to a Windows Store app

```
<!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

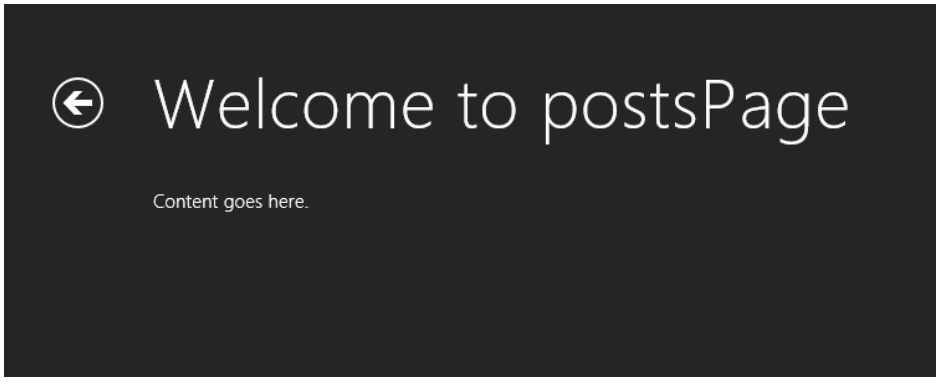


Figure 1.22: Navigating to a page control using the WinJS navigation service

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 AJAX[10]/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.

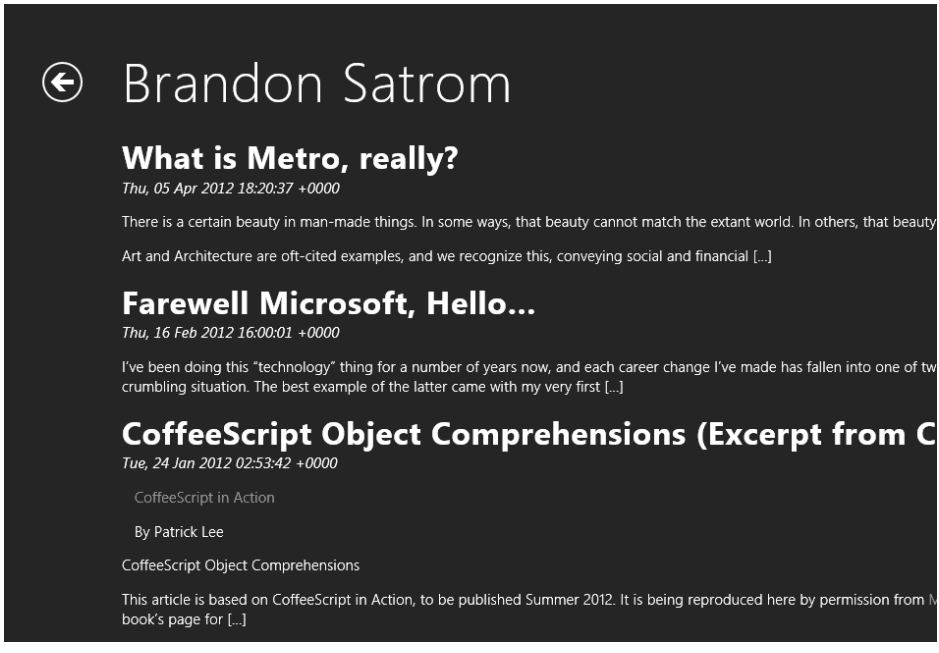


FIGURE 1.23: Showing the contents of an RSS feed using `WinJS.xhr`

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.

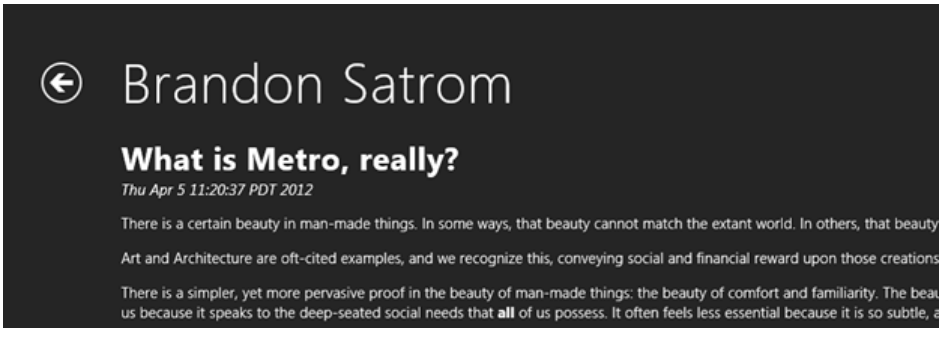


Figure 1.24: Showing the contents of an RSS feed using the WinRT `SyndicationClient`

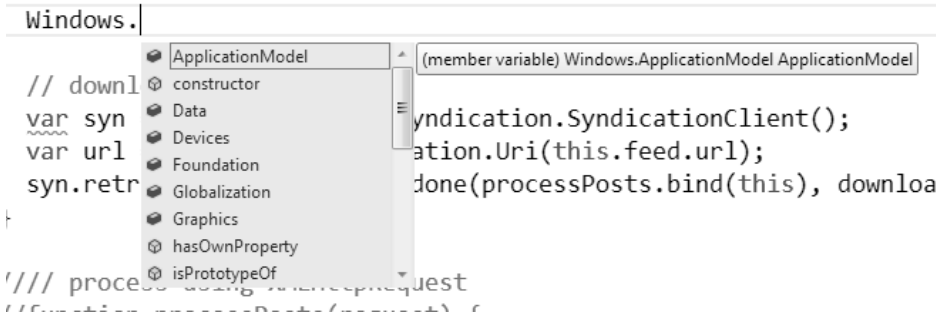


Figure 1.25: Visual Studio 2012 knows WinRT!

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.

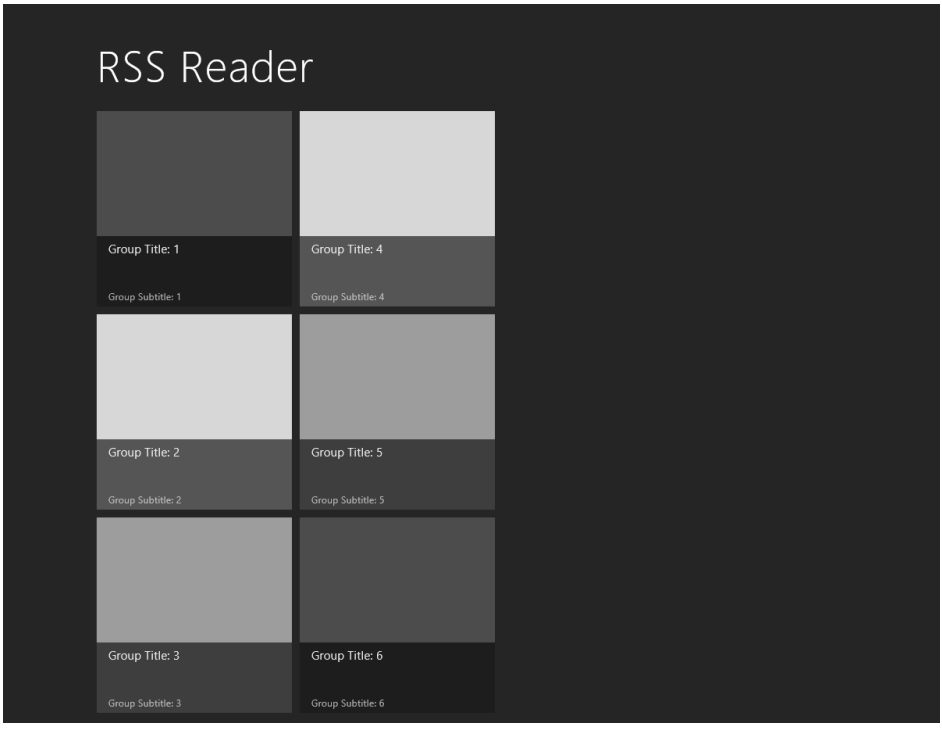


FIGURE 1.26: The `itemsPage.html` page from the Split App project template, showing groups of things

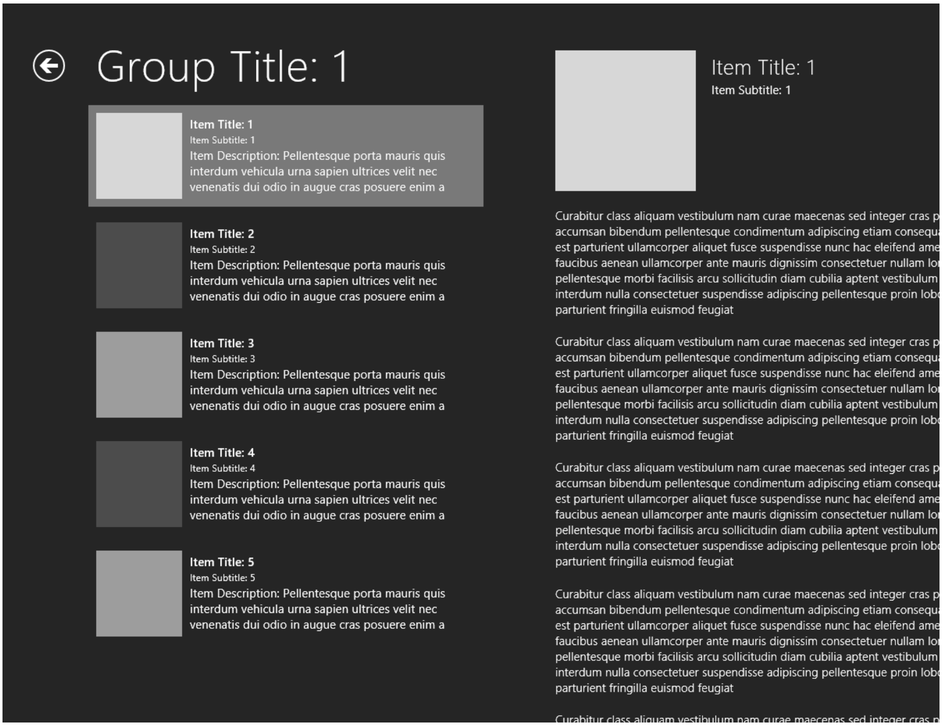


FIGURE 1.27: The `itemsPage.html` page from the Split App project template, showing a list of items

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

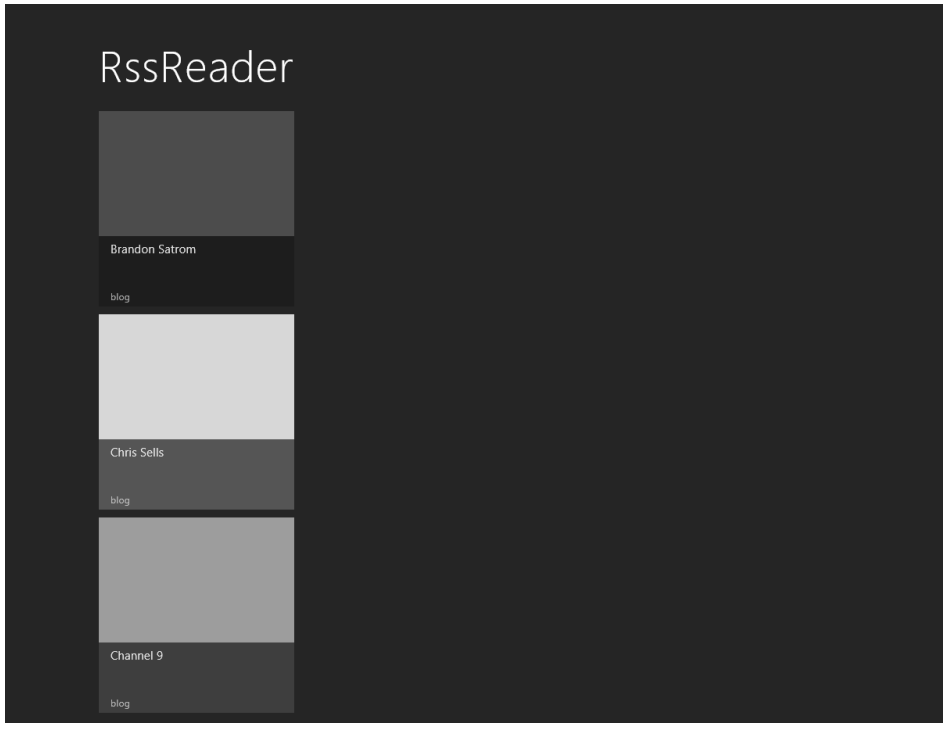


FIGURE 1.28: The `itemsPage.html` page from the Split App project template, showing real data

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."

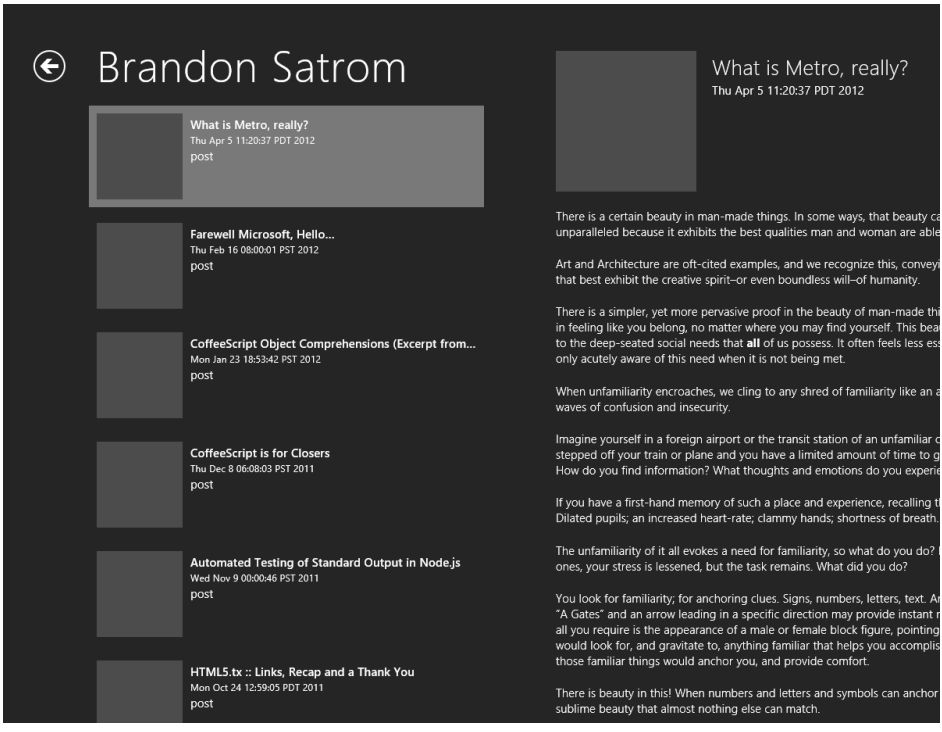


FIGURE 1.29: The `splitPage.html` page from the Split App project template, showing real data

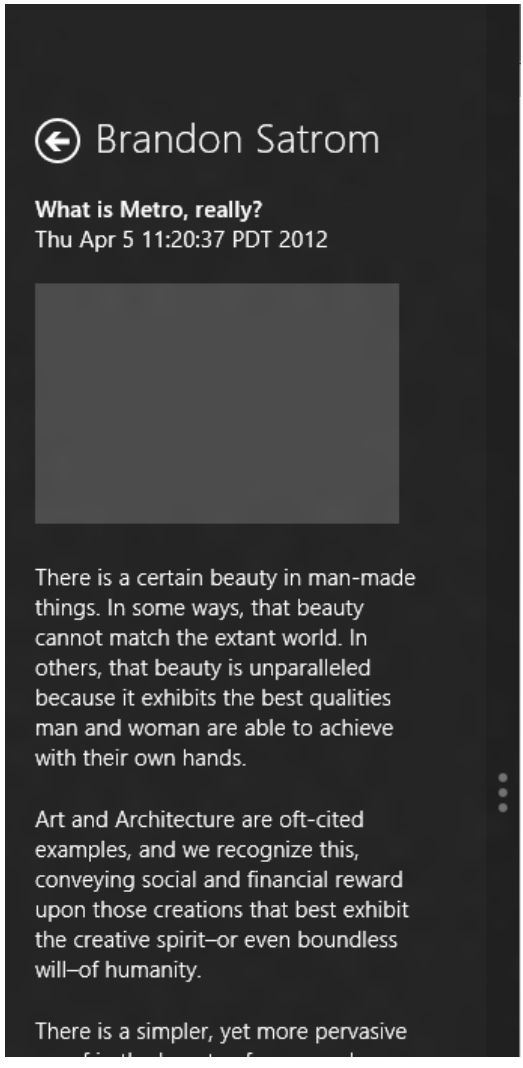


Figure 1.30: The snapped mode support built into the Split App project template

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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z