

start here!

Build
Windows 8 Apps
with HTML5 and
JavaScript

Beginner



Dino Esposito
Francesco Esposito

Ready to learn Windows 8 programming?

start here!™

Learn the fundamentals of Windows 8 programming—and begin creating apps for desktops, laptops, tablets, and other devices. If you have previous experience with HTML5 and JavaScript—simply start here! This book introduces must-know concepts and getting-started techniques through easy-to-follow explanations, examples, and exercises.

Here's where you start learning Windows 8 app development

- Build on your knowledge of HTML5, CSS, and JavaScript
- Create photo and media galleries with built-in HTML widgets
- Interact with the system through live tiles, contracts, and view state detection
- Store and access data on the local device and via the Internet
- Access webcam, GPS, and other sensors embedded in the device
- Create your first programs and publish them to the Windows Store

Download HTML5 and JavaScript code samples at:

http://aka.ms/SH_W8AppsHTML5JS/files

microsoft.com/mspress

ISBN: 978-0-7356-7594-0



U.S.A. \$24.99
Canada \$26.99
[Recommended]

Programming/JavaScript

Build Windows 8 Apps with HTML5 and JavaScript

About You

SKILL LEVEL: Some experience required
PREREQUISITES: Beginner-level HTML5 and JavaScript skills

About the Authors

Dino Esposito, CTO of a software and mobile services company, has written a dozen Microsoft Press books, including *Architecting Mobile Solutions for the Enterprise*.

Francesco Esposito, 15 years old, already has significant experience developing mobile applications for Windows Phone, Android, iOS, and BlackBerry.



**Start
Here!**[™]

Build Windows 8 Apps with HTML5 and JavaScript

Dino Esposito
Francesco Esposito

Copyright © 2013 by Dino Esposito

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-7594-0

1 2 3 4 5 6 7 8 9 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at <http://www.microsoft.com/learning/booksurvey>.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Christopher Hearse

Editorial Production: S4Carlisle Publishing Services

Technical Reviewer: Russ Mullen

Indexer: Angela Howard

Cover Design: Jake Rae

Cover Composition: Karen Montgomery

Illustrator: S4Carlisle Publishing Services

To Michela and Silvia, who are stronger and smarter than they think.

—DINO

To Grandma Concetta, for providing free calories through amazing quantities of unbeatable homemade prosciutto.

—FRANCESCO

Contents at a glance

	<i>Introduction</i>	<i>xiii</i>
CHAPTER 1	Using Visual Studio 2012 Express edition for Windows 8	1
CHAPTER 2	Making sense of HTML5	23
CHAPTER 3	Making sense of CSS	47
CHAPTER 4	Making sense of JavaScript	73
CHAPTER 5	First steps with Windows 8 development	97
CHAPTER 6	The user interface of Windows Store applications	121
CHAPTER 7	Navigating through multimedia content	147
CHAPTER 8	States of a Windows 8 application	177
CHAPTER 9	Integrating with the Windows 8 environment	201
CHAPTER 10	Adding persistent data to applications	235
CHAPTER 11	Working with remote data	265
CHAPTER 12	Accessing devices and sensors	291
CHAPTER 13	Adding Live tiles	319
CHAPTER 14	Publishing an application	335
	<i>Index</i>	<i>351</i>

Contents

<i>Introduction</i>	<i>xiii</i>
Chapter 1 Using Visual Studio 2012 Express edition for Windows 8	1
Getting ready for development	2
The software you need	2
Configuring Visual Studio 2012	5
Start playing with Windows 8 apps	9
The “Hello Windows 8” application	10
Adding a bit more action	16
Summary	21
Chapter 2 Making sense of HTML5	23
Elements of a webpage	24
Building the page layout with HTML5	24
Miscellany of other new elements	32
Collecting data	34
Adjusting input fields	35
Form submission	38
Multimedia elements	41
The <i>audio</i> element	41
The <i>video</i> element	42
Summary	45

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

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

microsoft.com/learning/booksurvey

Chapter 3	Making sense of CSS	47
	Styling a webpage	47
	Adding CSS information to pages.	48
	Selecting elements to style	51
	Basic style commands	55
	Setting colors	55
	Controlling text.	58
	HTML display modes.	60
	Spacing and the boxing model.	63
	Advanced CSS scenarios	66
	CSS pseudo-classes	66
	Media Queries	70
	Summary.	71
Chapter 4	Making sense of JavaScript	73
	Language basics	74
	The JavaScript type system.	74
	Dealing with variables.	77
	Dealing with objects	79
	Dealing with functions	81
	Organizing your own JavaScript code.	89
	Linking JavaScript code to pages	89
	Practices and habits.	92
	Summary.	94
Chapter 5	First steps with Windows 8 development	97
	The Windows 8 Runtime (WinRT)	98
	Windows Store apps and other apps	98
	An overview of the WinRT API.	100
	The Windows Store app user interface	102
	Aspects of the Windows 8 UI.	102
	Inspiring principles of the Windows 8 UI.	104

	Components for the presentation layer	107
	Data binding	112
	Understanding the application's lifecycle	116
	States of a Windows Store application	116
	Background tasks	119
	Summary	120
Chapter 6	The user interface of Windows Store applications	121
	Foundation of Windows Store applications	121
	Defining the layout of the application	122
	Application attributes	129
	Getting serious with the <i>ToDoList</i> application	132
	Building an interactive form	133
	Putting data into the form	138
	Summary	146
Chapter 7	Navigating through multimedia content	147
	Foundation of page navigation	147
	The navigation model of Windows Store applications	147
	Inside the Navigation App template	149
	Building a gallery of pictures	152
	Introducing the <i>FlipView</i> component	152
	Navigating to a detail page	156
	Zooming the image in and out	161
	Building a video clip gallery	165
	Introducing the <i>SemanticZoom</i> component	165
	Dealing with video	172
	Summary	174
Chapter 8	States of a Windows 8 application	177
	States of a Windows Store application	177
	Full-screen view states	178

Snapping applications	179
Making the application reactive	182
Towards an adaptive layout	188
General principles of snapped and filled views	188
Fluid layouts	189
Summary	200
Chapter 9 Integrating with the Windows 8 environment	201
Contracts and common tasks	202
Aspects of Windows 8 contracts	202
Contracts and extensions	204
Consuming the <i>File picker</i> contract	206
Choosing a file to save data	207
Choosing a file to load data	214
The <i>Share</i> contract	216
Publishing an application's data	216
Adding share source capabilities to <i>ToDoList</i>	217
Providing a Settings page	224
Populating the Settings charm	224
Creating a functional Settings page	227
Summary	233
Chapter 10 Adding persistent data to applications	235
Persisting application objects	235
Making <i>Task</i> objects persistent	236
Choosing a serialization format	240
Creating <i>Task</i> objects from files	243
Using the application's private storage	247
Storage options in Windows 8	248
Creating tasks in the isolated storage	251
Summary	264

Chapter 11 Working with remote data	265
Working with RSS data	265
Getting remote data	266
Parsing and displaying downloaded data	272
Drilling down into data	276
Working with JSON data	278
Laying out a Flickr viewer	279
Enhancing the application	285
Summary	290
Chapter 12 Accessing devices and sensors	291
Working with the webcam	291
Capturing the webcam stream	292
Processing captured items	297
Working with the printer	301
The <i>Print</i> contract	301
Printing context-specific content	305
Working with the GPS system	309
Detecting latitude and longitude	309
Making use of geolocation data	312
Summary	318
Chapter 13 Adding Live tiles	319
What's a Live tile anyway?	319
Tiles in action	320
Creating Live tiles for a basic application	323
Adding Live tiles to an existing application	326
Bringing back the <i>ToDoList</i> application	326
Implementation of Live tiles	328
Summary	333

Chapter 14 Publishing an application	335
Getting a developer account	336
Registering as a developer of free applications	336
Registering as a developer of paid applications.	339
Steps required to publish an application	340
Choosing a name for the application.	340
Packaging an application	342
Sideloaded applications	349
Summary.	349
<i>Index</i>	351

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

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

microsoft.com/learning/booksurvey

Introduction

For years, programming has been the nearly exclusive domain of people that others typically thought of as super-experts, gurus, or geniuses. The advent of mobile programming, however, changed things a bit because the idea of programming for these devices regained its appeal for youngsters. Today, a teenager appearing suddenly in the spotlight due to his or her ability to build a cool Android or iPhone app is not uncommon. There are a few reasons why this is happening, and why it's happening now.

One reason is certainly that today's teenagers are the first generation of digital natives. You're far more likely to find a few programmers among this group than among the members of previous generations. Another reason is that mobile apps are much easier to write than any other type of modern software. Mobile apps are small pieces of code built around a smart idea. It's one thing to build a mobile app, and quite another to plan and maintain a multitiered enterprise system.

Being mobile added a new dimension to software development. In this context, Windows Phone is not simply yet another mobile platform to code for; It is by far the simplest (and even most pleasant) platform to code for, regardless of background. This makes coding for Windows Phone an excellent way to get started with programming. This is particularly true for the smart kids who constantly hunger after new technology and seek emotional gratification in that technology. I've seen this happen with my 14-year-old son, Francesco—who is the effective coauthor of this book.

Microsoft Windows 8 takes the “programming with pleasure” approach one step further. With Windows 8, you can not only build mobile apps for tablets, but also build standalone applications for yourself, either for fun or to help automate some of the repetitive tasks of everyday life. Windows 8, with its overall simplicity, brings back a dimension of craftsmanship in applications that went missing as the complexity of software architectures and websites increased over the past decade or so. On one hand, Windows 8 is a powerful client front end for rich and sophisticated middleware; on the other, it is simple enough for nearly everybody to program via HTML5 and JavaScript.

This book is intended as a quick (but juicy) beginner's guide for getting started crafting Windows 8 applications, and how to publish and sell them through the Windows Store. The key point of this book is to make it clear that if you have a good idea and are a quick learner, you can create a Windows 8 app regardless of your age or programming background. You'll see how to write functional applications for the new Microsoft operating system and have them run on desktop computers, as well as

tablets. As evidence, consider that Francesco is a teenager—and he wrote most of the examples and a few chapters of this book.

After completing the book, you won't be a super-expert, but you'll surely know enough to start writing your own apps, at least for fun.

Who should read this book

This book is a beginner's guide to Windows 8 programming using web technologies, such as HTML5 and JavaScript. But the scope of the word "beginner" needs some further explanation. One definition of a beginner, in a programming sense, is a person who has never learned any serious programming. While this book does target such beginners, it does require some minimal background knowledge about HTML5, JavaScript, and some familiarity with basic concepts of logic and formalism, such as *IF*, *WHILE*, and assignments. Another definition of a beginner, however, would include people who have never learned Windows programming, or people who perhaps wrote COBOL for decades—or even perhaps a person who built and maintained a Visual Basic 6 application for the past 15 years. While this book can also be useful for those more experienced "beginners," people with serious programming experience are not the target audience for this book.

This book attempts to provide a smooth approach to key topics of Windows 8 programming. If you are primarily interested in Windows 8 and are new to Windows Phone, Microsoft Silverlight, or even single-page applications, then you should definitely consider getting this book.

Who should not read this book

This book won't make you a top-notch Windows 8 developer. If you have some solid experience with Windows 8, with Windows Phone or Silverlight, or with other programming languages, then you might want to try another, more advanced book instead, or just rely on online MSDN documentation or StackOverflow links. You should be a true Windows 8 beginner to enjoy this book.

Organization of this book

This book is divided into three sections. Chapters 1-5 cover the basics of acquiring and using Microsoft Visual Studio 2012 Express and also provide a summary of what you need to know about HTML, CSS, and JavaScript. Chapters 6-11 deal with programming Windows 8 apps and cover the foundation of Windows 8 programming while providing step-by-step exercises that help you understand and deal with the user interface of Windows 8 apps, graphics, video, data storage, and Internet calls. Finally, Chapters 12-14 focus on advanced Windows 8 programming, with an emphasis on working with device sensors and accessories (such as printers, GPS, webcams, and so forth), interacting with the system (Live tiles), and publishing your completed application.

Finding your best starting point in this book

Overall, the scenarios for using this book are quite simple. We recommend you read it cover to cover, because it is designed to guide you through the key topics you need to know to program Windows 8 with HTML5 and JavaScript. However, if you already have a solid grasp of the technologies used in this book—Visual Studio 2012 Express, HTML5, CSS, and JavaScript—you may be able to skip Chapters 1-4 without compromising your understanding of the rest of the book.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- Each exercise consists of a series of tasks, presented as numbered steps (1, 2, and so on) listing each action you must take to complete the exercise.
- Boxed elements with labels such as “Note” provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Press Alt+Tab” means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (such as, File | Close), means that you should select the first menu or menu item, then the next, and so on.

System requirements

You will need the following hardware and software to set up yourself on the various mobile platforms and compile the sample code:

- A PC equipped with Windows 8 and Visual Studio 2012 Express for Windows 8.

Code samples

Most of the chapters in this book are built around exercises that are reflected in the sample code for the chapter. All sample projects in their finalized form can be downloaded from the following page:

http://aka.ms/SH_W8AppsHTML5JS/files

Follow the instructions to download the `starthere-buildapps-winjs-sources.zip` file.

Installing the code samples

Follow these steps to install the code samples on your computer so that you can use them with the exercises in this book.

1. Unzip the `starthere-buildapps-winjs-sources.zip` file that you downloaded from the book's website (name a specific directory along with directions to create it, if necessary).
2. If prompted, review the displayed end user license agreement. If you accept the terms, select the accept option, and then click Next.



Note If the license agreement doesn't appear, you can access it from the same webpage from which you downloaded the `starthere-buildapps-winjs-sources.zip` file.

Errata and book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/SH_W8AppsHTML5JS/errata

If you find an error that is not already listed, you can report it to us through the same page.

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

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

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

<http://www.microsoft.com/learning/booksurvey>

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *<http://twitter.com/MicrosoftPress>*

Acknowledgements

Dino:

I'll be honest: **Russell Jones**, my editor at O'Reilly Media, convinced me to try this project. If the book is in your hands, both some of the good and some of the bad are on him! When Russell first mentioned this book, when it was still just an idea, I first declined, making the point that I have never written a book for beginners.

But then my son, Francesco (proud and efficient coauthor), made me look at the subject from a different perspective. It was one of those powerful forms of lateral thinking that only young people can sometimes contribute. Francesco said something like, "Dad, I don't think you only admit experts to your software design or ASP.NET MVC classes. If I were a true expert, I'd probably rarely take a class; if I need a class it is because I want someone to show me the way slowly and effectively. If I decide to invest money on a class it is because I feel somehow that I'm a beginner. Why should this be different for a book?"

That message hit home; I found that to be a valuable bit of wisdom; even coming from a 14-year-old boy.

So with that change of heart, I embarked enthusiastically on this project and asked Francesco to cooperate, because he was perfect for testing the material—essentially eating the dog food we were cooking up! Francesco did a truly fantastic job. At one point, I was on a plane about to leave and talking on the phone, giving suggestions on how to improve the gallery of photos and the downloading of JSON data from Flickr. From the outside, that phone call was nothing more than a classic business phone call—the last-minute kind you make just moments before the plane leaves the gate. But I was talking to my son! And, more importantly, he had diligently accomplished all the tasks by the time I got back. Thank you, Francesco!

Francesco:

I love technology and love the Microsoft software platform and tooling. In the beginning, for me, writing the book was primarily a way to get my hands on a Surface device. In the end, though, I spent most of the time working with the simulator and a secondary laptop.

Dad told me that exploring a technology near its birth is usually quite difficult, because you can't always rely on documentation or good examples being available. Frankly, to me that just sounded like one of those excuses that parents trot out when they're unable to do something themselves. Not knowing it might be hard, I just rolled

up my sleeves and worked out some examples. And in doing so, I also was able to contribute a list of points for Dad to expand on. I'm not sure this project would have been as pleasant for Dad without my help.

Working on the book was mostly fun, but I do recognize that this book is an important achievement for me. I know I'll feel better if I can share this moment with some people who make my life happier: my mom, my sister, Michela, my friends Francesco and Mattia, and all my waterpolo teammates at UISP Monterotondo. I love you all!

PS: Michela, do you remember that Christmas of 2009 when I was really giving you a hard time and in order to "save" you, Dad decided to initiate (or actually initialize?) me to programming?

Using Visual Studio 2012 Express edition for Windows 8

Differences of habit and language are nothing at all if our aims are identical.

— J. K. Rowling, *Harry Potter and the Goblet of Fire*

Microsoft Windows 8 marks the debut of a significantly revised runtime platform—the Windows RunTime (WinRT) platform. Like the .NET platform, WinRT supports several programming languages. You will find a pleasant surprise (and an old acquaintance) side by side with the popular .NET languages (such as, C#, Visual Basic, C++, F#)—the JavaScript language.



Note You may not even recall that a decade ago, when Microsoft first shipped the .NET Framework, developers were also given a chance to write applications using an adapted version of JavaScript called JScript .NET. It was not exactly a success; indeed, today you won't even find JScript .NET supported in Visual Studio—the premiere development environment for .NET code. Ten years ago, JavaScript was probably close to the bottom of its popularity. JScript .NET was a dialect of the standard JavaScript, and using JScript .NET didn't mean you could use HTML and CSS to shape up the user interface of the resulting application. This is different in Windows 8.

Building Windows 8 applications with JavaScript means that you define the layout of the user interface with HTML and add style and graphics using CSS. As for the application's logic, you use the standard JavaScript language enriched by any JavaScript libraries you wish (such as the common jQuery library), while you access WinRT system classes using an ad hoc Microsoft-created JavaScript wrapper—the WinJs library.

If you already know a bit of JavaScript development, building Windows 8 applications will not be a huge, new type of adventure. If you are not already a JavaScript developer, the JavaScript route probably represents the shortest path for learning to build Windows 8 applications.

This chapter sets up the preliminary aspects of such a learning path and discusses what you need to install—specifically Windows 8 and Microsoft Visual Studio—and how to configure the environment. In the next chapters you'll first see a summary of HTML (in particular, the latest version of HTML, known as HTML5), CSS, and JavaScript, and then attack the task of building Windows 8 applications with topics more specifically related to Windows 8 programming.



Important If you are already familiar with HTML5, CSS, and JavaScript, you might want to start directly with Chapter 5. If not, at the very minimum I recommend you look carefully at Chapters 2, 3, and 4. Better yet, I suggest you look into specific books for HTML5 and JavaScript, as the chapters you find here represent about 10 percent of the content you would find in a dedicated book. You might want to explore other books in this Microsoft Press series that address these topics directly: *Start Here! Learn HTML5* by Faithe Wempen (Microsoft Press, 2012) and *Start Here! Learn JavaScript* by Steve Suehring (Microsoft Press, 2012).

Getting ready for development

So you want to start building applications for Windows 8 using HTML, CSS, and JavaScript. First, you need to make sure that some software is properly installed on your development machine. The following section discusses the details.

The software you need

As obvious as it may sound, you need to have Windows 8 installed to develop, test, and run Windows 8 applications. The easiest way to develop and test applications for Windows 8 is by using the current version of Visual Studio—Visual Studio 2012.

There are various editions of both Windows 8 and Visual Studio 2012, but for the purposes of this book, you'll need at least the minimal versions of each product: Windows 8 Basic edition and the free Visual Studio 2012 Express edition for Windows 8 applications.

Installing Windows 8

Having a machine equipped with Windows 8 is a fundamental prerequisite to working through the information and exercises in this book. Windows 8 comes in a few flavors, as detailed in Table 1-1.

TABLE 1-1 Windows 8 editions

Version	Description
Windows 8	The Basic edition of Windows 8 is available for both the x86 and x86-64 architecture. It provides a new Start screen and redesigned user interface, live tiles, Internet Explorer 10, and more.
Windows 8 Pro	This edition offers additional features such as booting from VHD and support for virtualization via Hyper-V.
Windows 8 Enterprise	This edition adds IT-related capabilities such as AppLocker and Windows-To-Go (booting and running from a USB drive). This version also supports installation of internally developed applications from locations other than the Windows Store.
Windows 8 RT	Only available pre-installed on ARM-based tablets, it also natively includes touch-optimized versions of main Office 2013 applications.

If you don't have your copy of Windows 8 already, you can get a free 90-day trial version from the following location: <http://msdn.microsoft.com/en-us/evalcenter/jj554510.aspx>. Note that this link gets you a non-upgradeable copy of Windows 8 Enterprise. Before you embark on the download, consider that because it is a few gigabytes in size, it may not be quick!

Installing Visual Studio Express

Once you have Windows 8 installed, you can proceed to download Visual Studio 2012 Express edition. (Note that in the rest of the chapter—and the entire book—we'll be using the term *Visual Studio* or *Visual Studio 2012* often just to mean the Visual Studio 2012 Express edition.) As shown in Table 1-2, Visual Studio is available in different flavors.

TABLE 1-2 Visual Studio 2012 editions

Version	Description
Ultimate	The feature-complete version of Visual Studio 2012, offering the top-quality support for every feature.
Premium	Lacks some extensions in the area of modeling, debugging, and testing.
Professional	Lacks even more functionalities in the area of modeling, debugging, and testing but still offers a great environment to write and test code.
Express	Free but basic version of Visual Studio 2012 optimized for specific development scenarios. In particular, it is available for building web applications or Windows 8 applications.

You can read more about and compare Visual Studio features at the following page: <http://www.microsoft.com/visualstudio/11/en-us/products/compare>.

To start downloading Visual Studio Express for Windows 8, go to the Dev Center for Windows 8 applications at <http://msdn.microsoft.com/en-us/windows/apps> (see Figure 1-1).

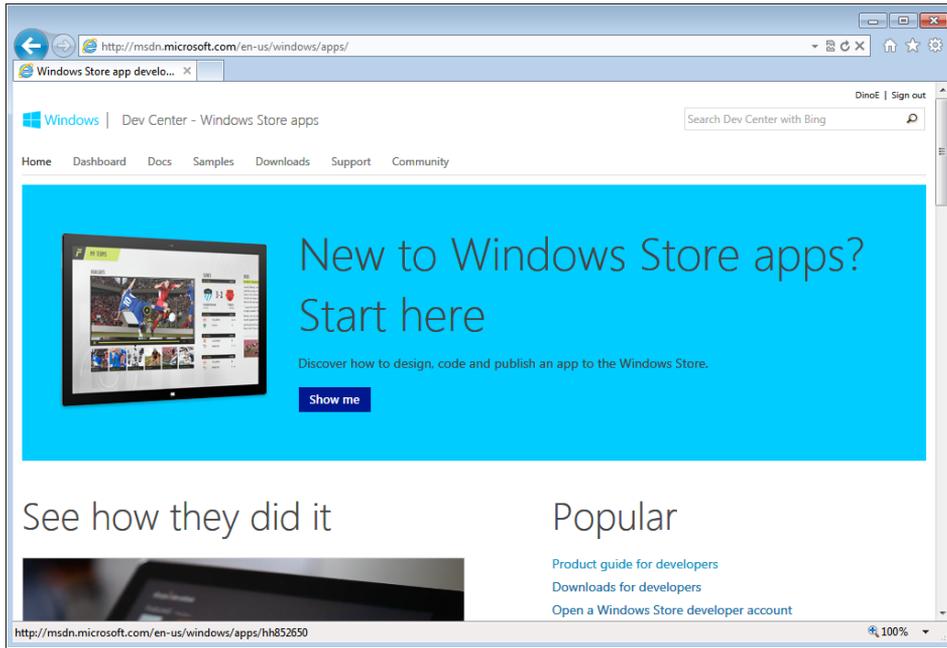


FIGURE 1-1 The home page of the Dev Center for Windows 8 applications.

After clicking the link to download the tools and Software Development Kit (SDK), you will be sent to another page where you can finally start the download process, as shown in Figure 1-2.

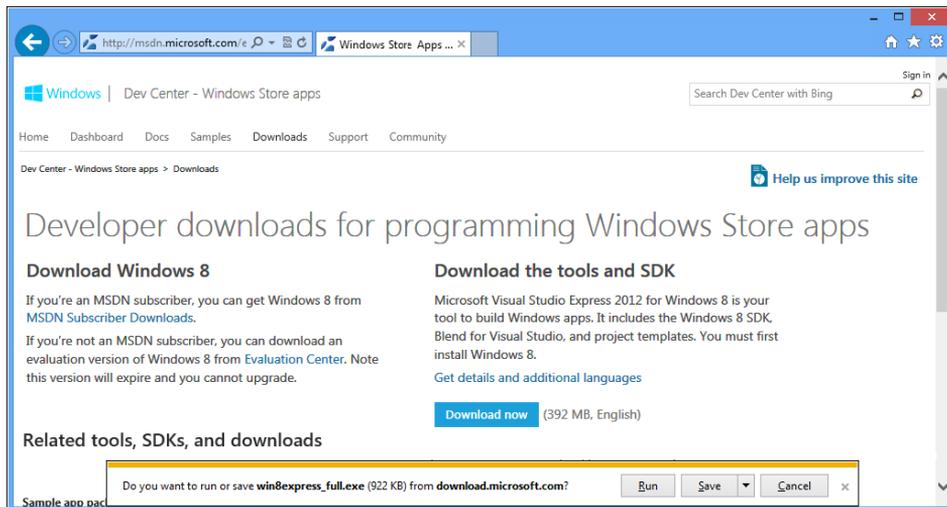


FIGURE 1-2 Downloading the tools for Windows 8 application programming.

Note that you can choose to save the setup program to your local disk or you can run it directly. If you plan to reuse the program on different machines, it could be useful to save it to a known location first.

4 Start Here! Build Windows 8 Apps with HTML5 and JavaScript

At various times during the setup, you'll be prompted to accept or modify options. For the purposes of this book, you can simply accept all the default options. The default setup installs the newest .NET Framework 4.5, the Windows 8 SDK, plus a bunch of other tools and project templates. At the end of the installation, if everything worked just fine, you should expect to see the screen reproduced in Figure 1-3. In the unfortunate case in which the software doesn't install correctly, you will get a message with some helpful directions. Please follow them carefully.



FIGURE 1-3 Visual Studio 2012 is ready to launch.

Now you're ready for some real fun: launching and configuring Visual Studio for your first Windows 8 application!

Configuring Visual Studio 2012

After completing the setup, Visual Studio 2012 Express requires a couple of more steps before it is ready to run.

Getting a product key

Upon launching for the first time, Visual Studio 2012 requires that you activate your copy. This happens through a screen like the one shown in Figure 1-4.

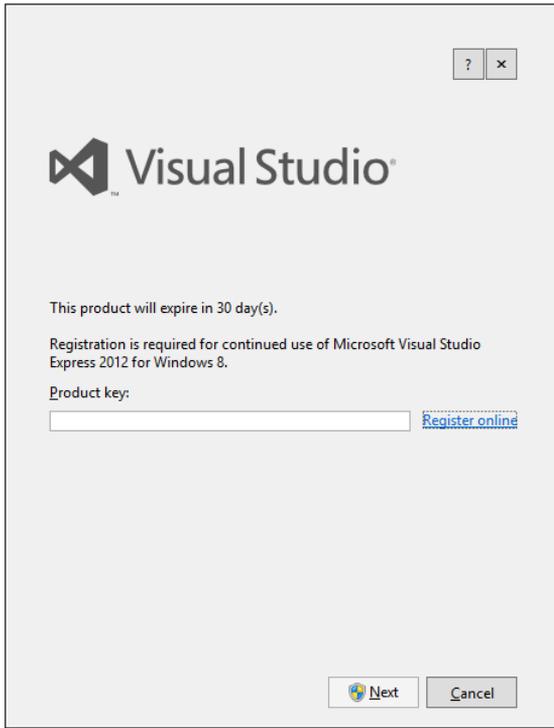


FIGURE 1-4 Product key required for Visual Studio 2012 Express.

Clicking the “Register online” link takes you to a page where you can insert your name, email address, and company details (see Figure 1-5).

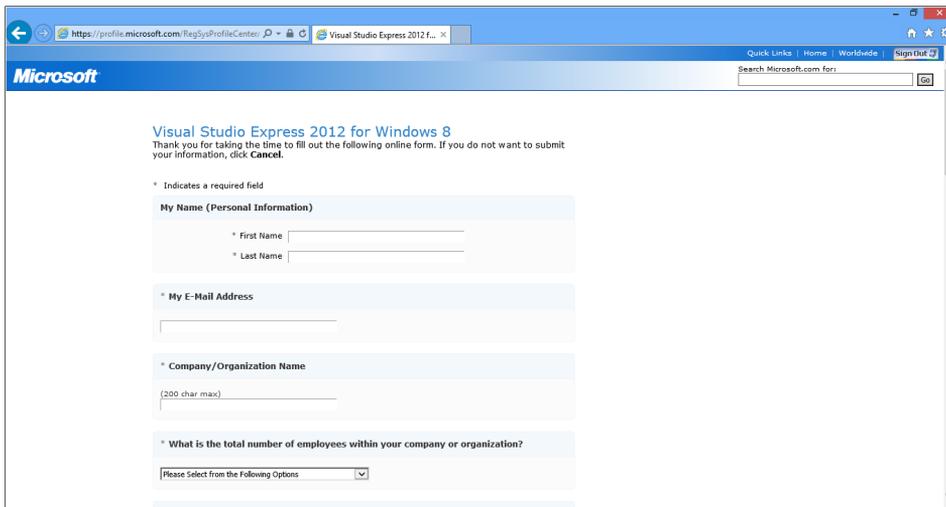


FIGURE 1-5 Registration required for Visual Studio 2012 Express.

You then submit the form. You'll receive an email containing the product key to unlock the current version of Visual Studio (see Figure 1-6).

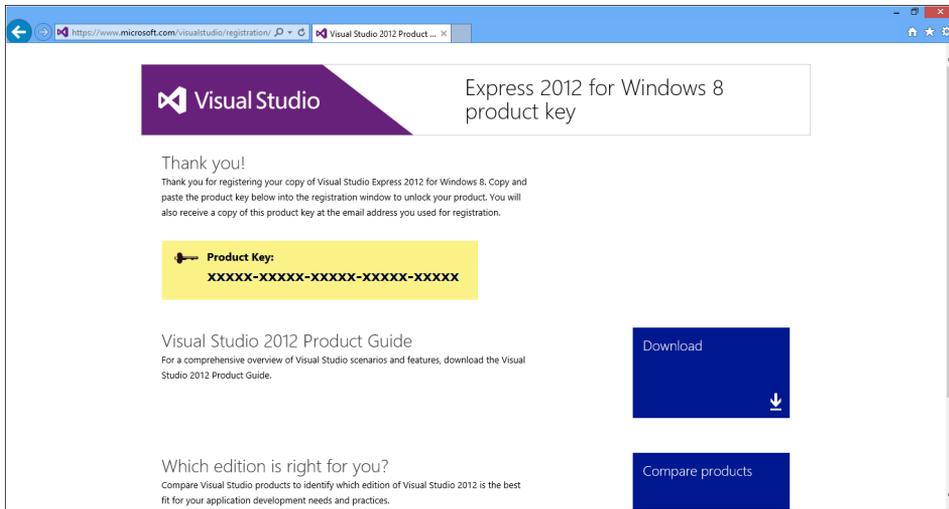


FIGURE 1-6 Product key retrieval for Visual Studio 2012 Express.

It usually takes only a few seconds to receive an email from Microsoft to your specified address. The email contains the product key as text; copy it to the clipboard and switch back to Visual Studio. In the same window you saw in Figure 1-4, paste the product key you just received.

Creating a developer account

To write and test Windows 8 applications, you need a developer license from Microsoft. The license is free and entitles you to be a registered Microsoft developer. Getting such a license requires only that you sign in using your Windows Live ID, as shown in Figure 1-7. (If you don't have a Windows Live ID, the dialog box that prompts you to enter it provides a quick "Sign Up" link.)

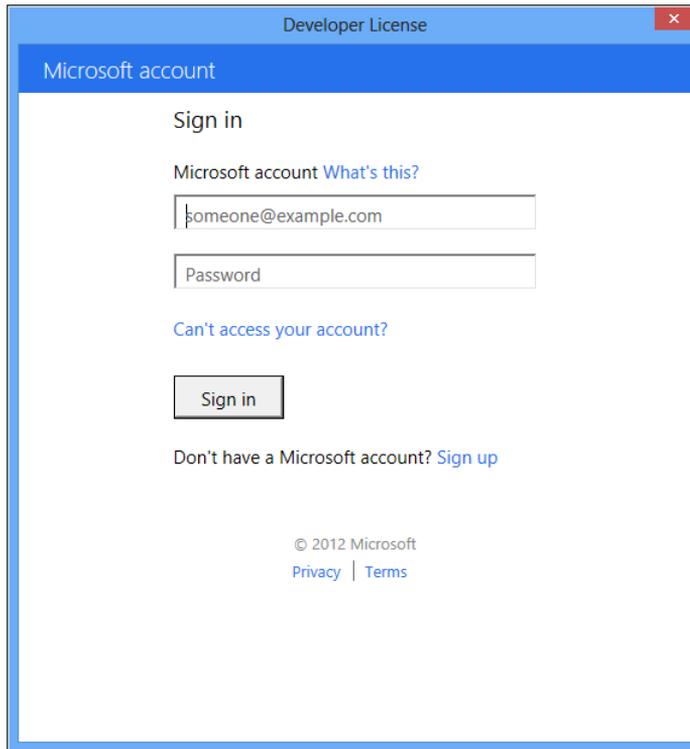


FIGURE 1-7 Creating your developer account.

A developer license successfully installed on a machine enables you to freely create and run Windows 8 applications outside the official Windows Store.



Note A Windows 8 machine can install only *certified applications*, either downloaded from the Windows Store (in much the same way in which it works for Windows Phone applications), or created by registered developers on a “signed” machine, so you’ll need the developer license to complete the examples in this book.

You won’t receive any further warning from the system until the developer license expires or you remove it from the machine. If your license expires, you can renew it directly from the Visual Studio environment. To renew a license, users of Visual Studio Express click the Store menu and then select Acquire Developer License, as shown in Figure 1-8.

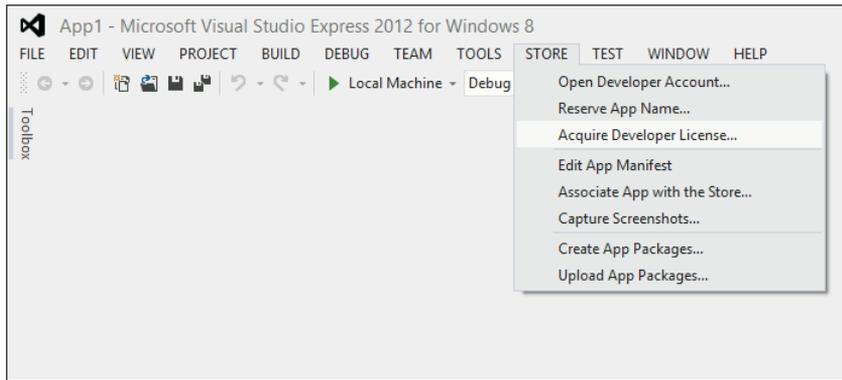


FIGURE 1-8 Renewing your developer license.

You can get as many developer licenses as you need, provided that you have a Windows Live ID account.

Windows Store account

Getting a developer license for Windows 8 is only the first mandatory step in developing and testing Windows 8 applications. Armed solely with a developer account, you can't publish a Windows 8 application to the Windows Store for others to download and install.

There's no direct relationship between developer accounts and Windows Store accounts. Each plays a specific role and you can get one without having the other. However, if you own a Windows Store account and then qualify for a developer license, then the default expiration of your developer license is automatically set to a longer time.

The point to remember here is that before you can publish your Windows 8 application to the Windows Store, you need to get a Windows Store account. You'll see how to obtain a Windows Store account later, in Chapter 14.



Important As a developer and user of a Windows 8 system, keep in mind that your machine can only run Windows 8 applications that have been downloaded from the Windows Store or custom applications for which a developer license has been installed on the machine. Another scenario enables you to host custom applications—when those applications have been “sideloaded” onto the machine by your organization, which in turn holds an enterprise store account.

Start playing with Windows 8 apps

With a developer license installed on your Windows 8 machine, you're now ready to play with Windows 8 applications. When you are about to create a brand new project, you must first choose a project template and a programming language. After you do that, Visual Studio provides some help

by generating some vanilla code appropriate to that template and language that you can customize and extend.

For the purposes of this book, your programming language of choice is JavaScript. It's worth remembering, though, that you could use other languages, such as C#, Visual Basic, and even C++.

The “Hello Windows 8” application

Without further ado, launch Visual Studio and discover what it takes to create a new project.

It couldn't be simpler, actually; from the start page, you just click the New Project link, as shown in Figure 1-9.

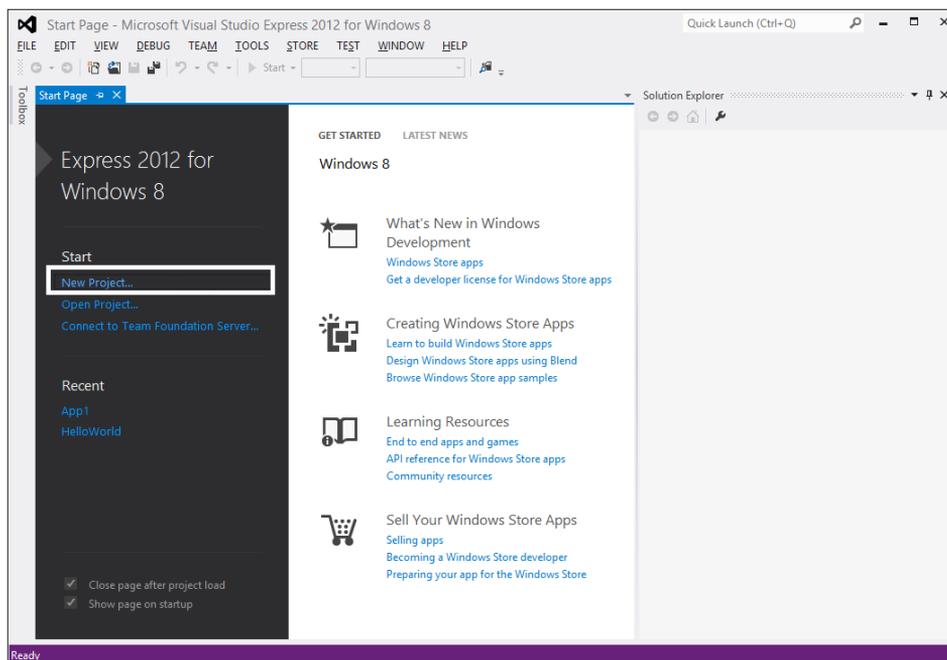


FIGURE 1-9 Creating a new project.

Choosing a project template

Visual Studio offers a few predefined templates for your new project, but choosing the project template only appears to be an easy task. It requires that you have a reasonably clear idea of the final result you want to achieve. The template you truly want to use depends on the interaction model you have in mind, the graphics, and the content you need to work on. Figure 1-10 shows the New Project window you will see after electing to create a new project.

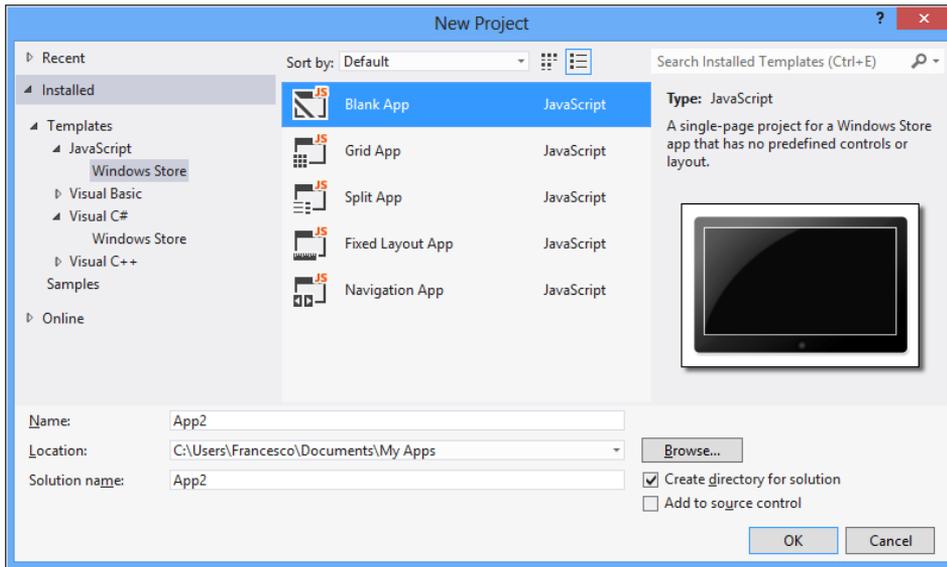


FIGURE 1-10 Choosing a project template.

Templates are grouped by programming language. In the Express edition of Visual Studio considered here, you can create just one type of application—a Windows 8 application for the Windows Store. If you acquire a more advanced edition of Visual Studio, you should expect to see more options available, including web, console, and desktop applications.

So how do you decide which template to use?

Templates have the primary purpose of saving you some work, at least for common application layouts. You are not forced to pick up a specific template, however. If none of the predefined templates seems to be right for you or, more likely, if you don't know exactly which one to pick, then you just select the template for a blank application. Table 1-3 provides more information on the predefined templates for JavaScript.

TABLE 1-3 Predefined project templates for Windows Store applications

Template	Description
Blank App	The application consists of a single and nearly empty page: no visual controls, no widgets, and no layout defined.
Grid App	A master-detail application made of three pages. The master page groups items in a grid. Additional pages provide details on groups and individual items.
Split App	A two-page master-detail application in which the master page shows selectable items and the details page lists related items alongside.
Fixed Layout App	A single-page application whose layout scales using a fixed aspect ratio.
Navigation App	A multipage application with predefined controls to navigate between pages.

For the purposes of this book, the easiest is starting with a brand new blank application. You'll experiment with other types of templates in the upcoming chapters.

Creating the sample project

Before you give Visual Studio the green light to create files, you might want to spend some time thinking about the location of the project. In Figure 1-10, you see *Location*; that's the place where you enter the disk path to the files being created for the project.

It is always preferable to save your sample applications in a well-organized structure. For the sample code of this book, you'll use a root directory named *Win8* containing *ChXX* directories for each chapter, where *XX* is a two-digit chapter number.

By default, Visual Studio saves your project files right under the *Documents* folder and creates a new directory for each solution. You can change the default location of a project by simply editing the path in the *Location* every time. Alternatively, you can set a new default path for every project by selecting *Options* from the *Tools* menu and then picking up the *General* node under the *Projects and Solutions* element (see Figure 1-11).

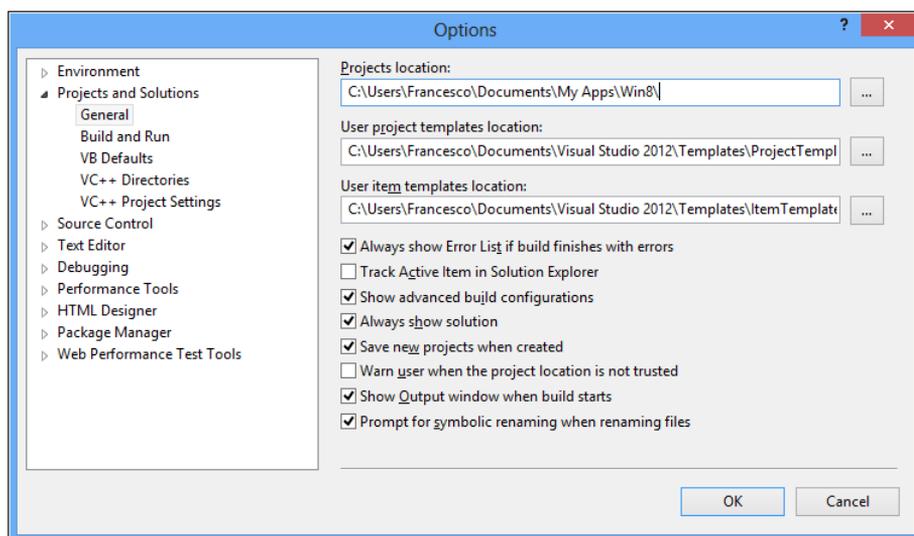


FIGURE 1-11 Changing the default project location.

For the “Hello Windows 8” application, you’ll create a new blank application project named **HelloWin8** in the *Win8/Ch01* folder, as shown in Figure 1-12.

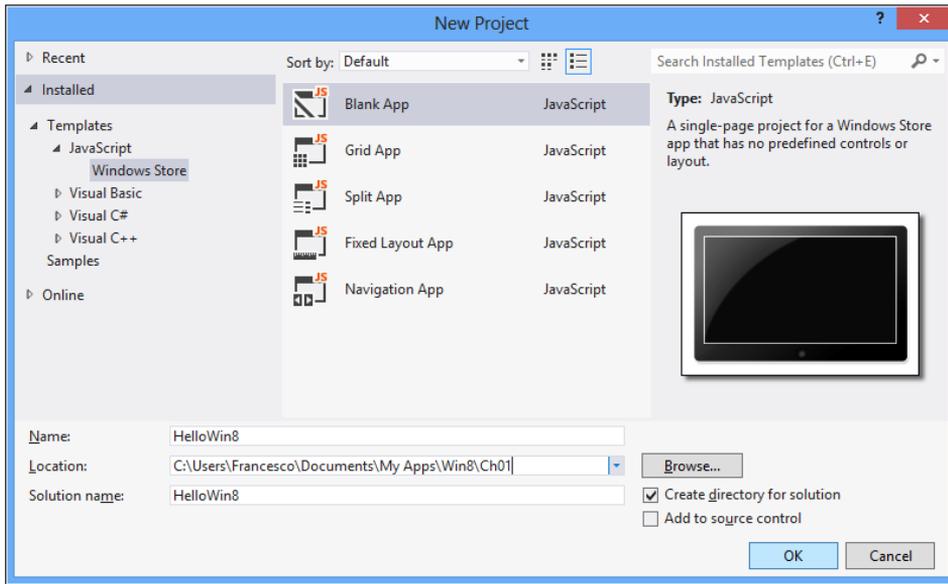


FIGURE 1-12 Creating the HelloWin8 project.

Click OK and you're officially a Windows 8 developer.



Note Visual Studio 2012 comes with a dark theme for windows and controls by default. For the sake of print, we changed it to the light theme which makes for screenshots that render better in print. Anyway, to change the Visual Studio theme, open the Tools | Options menu, and then select Environment from the window shown in Figure 1-11.

Tweaking the sample project

Right after creation, the *HelloWin8* project looks like the image shown in Figure 1-13. It references the Windows Library for JavaScript (under the References folder) and is centered on a HTML page named *default.html*. This page defines the entire user interface of the application and links a Cascading Style Sheet (CSS) file (*css/default.css*) for graphics and a JavaScript file for the logic that loads up the page content and provides any expected behavior (*js/default.js*). Just the *default.js* file is opened in Visual Studio by default.

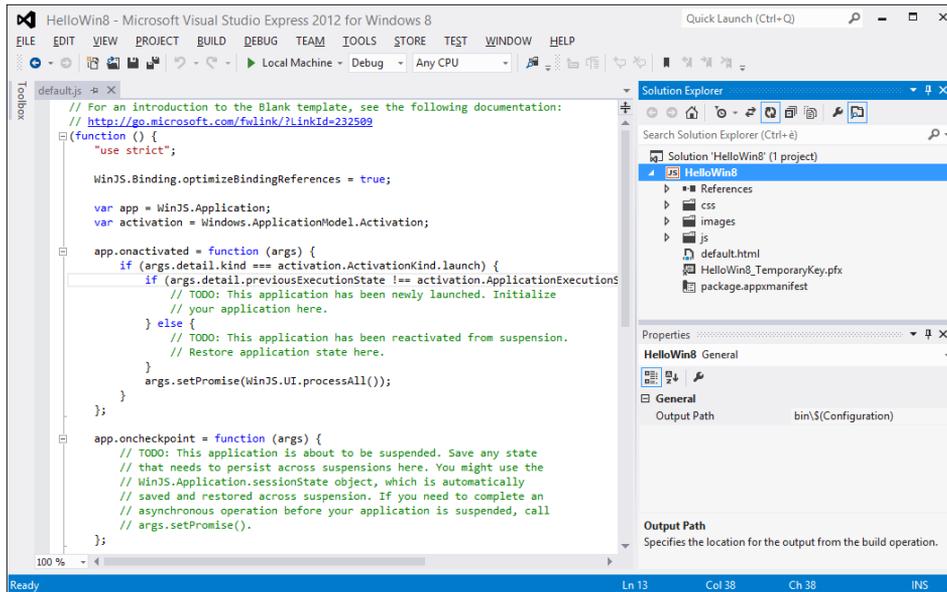


FIGURE 1-13 The *HelloWin8* project.

It turns out that a Windows 8 application written using JavaScript looks like a self-contained web application made of HTML pages properly styled using CSS and powered by JavaScript logic. If you are familiar with the web paradigm and client-side web development, then you only need to make sense of the Windows 8-specific application programming interface (API) exposed to you via a few JavaScript files to link.

Before compiling the project to see what happens, let's make some minimal change: Close the *default.js* file and open up the *default.html* file, which is responsible for the home page of the application (see Figure 1-14). To open a file that is part of the current project, you locate the file by name in the Solution Explorer panel and then double-click it. In general, if you need to open a file that is not included in the project for your reference, then you might want to use the Open item on the File menu.

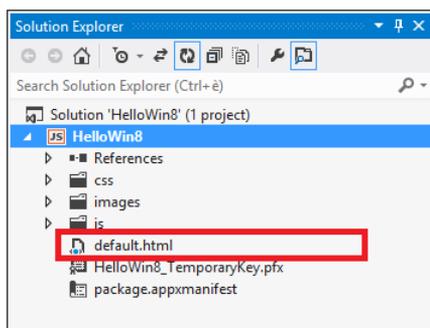


FIGURE 1-14 Opening *default.html*.

The Debug option in Figure 1-15 refers to the way in which the compiler produces the code. In debug mode, the binaries include additional information that allows you to set *breakpoints* on specific lines of code and proceed step by step. A breakpoint is a line of code where execution will pause. You typically use breakpoints to stop execution at a given point and investigate the state of the application and its internal data. You can have multiple breakpoints in the program. The Release mode is required for finished applications ready for distribution. In the book, you'll be using the Debug mode predominantly.

Figure 1-16 shows the application in action on the local machine.

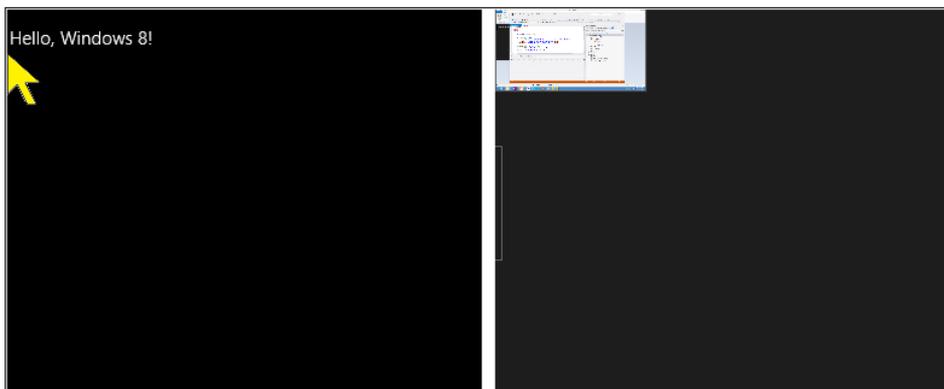


FIGURE 1-16 The HelloWin8 app in action.

If you run the application in the simulator, then the application runs in a separate window you can control at will. When you run it on the local machine, then the app runs full screen and it is not immediately apparent what you need to do to get back to Visual Studio to terminate the app. Here's how to exit the application: move the mouse towards the left border until you see a window icon to click to return to the desktop mode. To terminate the app, you then click the Stop button that has replaced the Play button in the Visual Studio user interface.

You're done. But it was way too simple, wasn't it? So let's make the sample application more colorful and add a bit of action too.

Adding a bit more action

Create a new project and name it **HelloWin8-Step2**. First, you'll make it more colorful by simply adding more HTML elements and style information. Next, you will transform it into a simple but fully functional application that generates a random number.

Adding style to the page

Open up the *default.html* page and edit its body tag. The body should now include title and subtitle separated with a line. You use a couple of HTML5 elements for this. Note that in the next chapter you'll learn a lot more about HTML5. Here's the modified body of the page:

```

<body>
  <header>
    Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
    <b>JavaScript</b>
    <hr />
  </header>
  <footer>
    <hr />
    Dino Esposito | Francesco Esposito
  </footer>
</body>

```

Now let's proceed with colors and fonts. The style of the page is defined in the *default.css* file from the CSS folder. By editing a CSS file, you can change nearly everything in a HTML page that has to do with appearance and layout. You'll find a summary of what's important to know about CSS in Chapter 3, "Making sense of CSS."

In the *default.css* file, you initially find something like below:

```

body {
}

```

This code describes the style to be applied to the tag *body* of any page that links the CSS file. You can edit the CSS file manually or you can create CSS styles using a builder tool available in Visual Studio. To use the tool, right-click a CSS element (that is, *body*) and select Build Style, as shown in Figure 1-17.

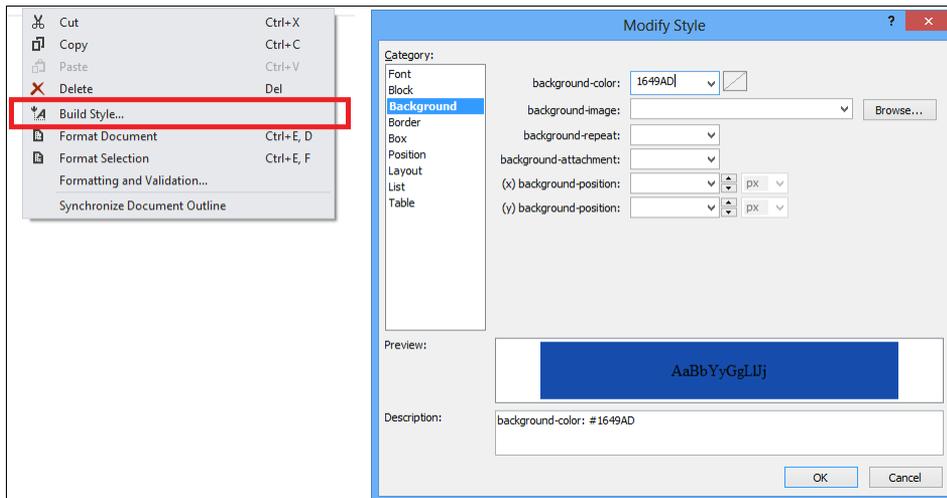


FIGURE 1-17 Editing the style of the page.

Here's a modified style for the page body that changes the background color and adds a bit of space around the content:

```
body {  
    background-color: #1649AD;  
    padding: 10px;  
}
```

You can also customize the header and footer elements slightly to define the color of the text, font size, and a vertical offset, as shown below:

```
header {  
    font-size: x-large;  
    color: #ffffff;  
    padding-bottom: 50px;  
}  
footer {  
    font-size: large;  
    color: #eeee00;  
    padding-top: 50px;  
}
```

Now run the application and be proud of it! (See Figure 1-18.)



FIGURE 1-18 The application running with a modified style.

Generating a random number

So far the application has no behavior at all and is limited to displaying some static text. Let's make it a bit less obvious and add any necessary structure and logic to make it generate and display a random number.

First off, add the following markup to the body of the page in *default.html*. The markup consists of two *DIV* blocks containing the placeholder for generated number and the button to click to get a new number. You insert the following markup between the header and footer elements:

```
<div>
    <label id="numberLabel">?</label>
</div>
<div>
    <input id="numberButton" type="button" value="Get number" />
</div>
```

Next, open the *default.js* file and add the following JavaScript functions at the bottom of the file:

```
function numberButtonClick() {
    var number = generateNumber();
    document.getElementById("number").innerHTML = number;
}
function generateNumber() {
    var number = 1 + Math.floor(Math.random() * 1000);
    return number;
}
```

The first function is the handler for the click event on the button. The second function just generates and returns a random number between 1 and 1000. The final step consists of binding the click handler to the actual button in the HTML markup. There are a number of ways to do this, the simplest of which is shown below:

```
<input id="numberButton" type="button" value="Get number"
    onClick="numberButtonClick()" />
```

A more elegant way—and the recommended way of doing it in Windows 8 programming—consists of making the binding dynamically as the page is loaded. So open the *default.js* file and modify the code of the *app.onactivated* function, as shown below:

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
        if (args.detail.previousExecutionState !==
            activation.ApplicationExecutionState.terminated) {
            // TODO: This application has been newly launched. Initialize
            // your application here.
            document.getElementById("numberButton").addEventListener(
                "click", numberButtonClick)
        } else {
            // TODO: This application has been reactivated from suspension.
            // Restore application state here.
        }
    }
}
```

```

    }
    args.setPromise(WinJS.UI.processAll());
  }
};

```

In the end, you just add one line to run when the application has been newly launched. The line just registers a handler for any click event raised by the specified button.

You can give the final touch to the application with a second pass on CSS to adjust the rendering of the label and button. Add the following to the *default.css* file:

```

#numberButton {
    font-size: x-large;
}
#numberLabel {
    font-size: xx-large;
    color: #eeee00;
    font-weight: bold;
}

```

The leading pound (#) symbol indicates that the style applies to any HTML element whose ID matches the name—for example, the style defined as *#numberButton* applies to all elements with an ID of *numberButton*. Figure 1-19 shows the modified application in action.



FIGURE 1-19 A Windows 8 application to get random numbers.

Although this is still a fairly simple application, it should be enough to get you started and to give you the overall feeling of how you approach Windows 8 programming with HTML5 and JavaScript. You'll start building more sophisticated applications in Chapter 5, "First steps with Windows 8 development."

Summary

This chapter provided a step-by-step guide to getting ready for Windows 8 programming. You started with the operating system and tools necessary for writing code, then installed and configured Visual Studio 2012 Express edition for Windows 8, and finally played a bit with the simplest type of application.

Before digging into more Windows 8 development, it is necessary to ensure that everybody reading this book is aligned to a minimum level of knowledge of web technologies such as HTML5, CSS, and JavaScript. Therefore, the next three chapters provide a summary of what you need to know about HTML5, CSS, and JavaScript to successfully work through the later chapters. If you feel you already know enough, then feel free to jump directly to Chapter 5. If the later chapters prove too difficult, I recommend you review Chapters 2–4 and/or brush up your knowledge with other resources on HTML5, JavaScript, and CSS.

Making sense of HTML5

Broadly speaking, the short words are the best, and the old words best of all.

—Winston Churchill

HTML5 is the latest version of the HTML language—the popular text-based language used to define the content of webpages. HTML appeared on the scene in the early 1990s. In the beginning it was merely a *markup language* apt at describing simple documents. A markup language is a language based on a set of markers that wrap text and give it a special meaning.

Initially, the set of HTML markup elements, called “tags” or (better) “elements,” was fairly limited. It contained elements to define references to other documents and headings, to link to images and paragraphs, and apply basic text styling such as bold or italic. Over the years, however, the role of HTML grew beyond imagination, progressing from being a simple language that described documents to a language used to define the user interface of web applications. That trend continues today with HTML5.

The latest version of HTML5 removes some of the older elements and makes it easier to keep elements that provide style information in one place, and elements that provide text and define the layout of the text, in another place. As you’ll see in more detail in the next chapter, style information can be defined through a special distinct file known as a Cascading Style Sheet (CSS). In addition, HTML5 adds some new elements suitable for including multimedia content and drawing, and several new frameworks for manipulating the content of the page programmatically.

With HTML5 alone, you still won’t be able to go too far toward building a complete application. However, the union of HTML5, CSS, and JavaScript functions as a close approximation to a full programming language.

- You use HTML5 to define the layout of the user interface and to insert text and multimedia.
- You use CSS to add colors, style, and shiny finishes.
- Finally, you use JavaScript to add behavior by gluing together pieces of native frameworks such as Document Object Model (DOM), local storage, geolocation and, for example, all the specific services of Windows 8 exposed via the Windows 8 JavaScript library (WinJs). The DOM,

in particular, is the collection of programmable objects that expose the structure of the current document to coders.

In the rest of this chapter, you'll briefly explore the basics of the HTML5 markup elements, including input forms and multimedia. Neither this chapter nor the rest of the book covers every aspect of the basics of HTML. If you need a refresher on the fundamentals of HTML, you can refer to the book *Start Here! Learn HTML5*, by Faithe Wempen (Microsoft Press, 2012).



Important This chapter and the next two provide an overview of HTML, CSS, and JavaScript. In these chapters, you'll get acquainted with new key elements of HTML5 and CSS3 and receive an end-to-end coverage of common programming techniques you use in JavaScript. The content of these chapters is not specific to Windows 8 apps; it is instead meant to be preliminary to upcoming chapters where you'll be using ad hoc elements from the WinJs library in a basic HTML skeleton, using custom CSS for graphics and custom JavaScript for behavior.

Elements of a webpage

HTML5 comes about a decade after its most recent predecessor (HTML4). Looking at what's new in HTML5, one could reasonably say that all these years have not passed in vain. HTML5 provides a set of new elements that offer several native functionalities that developers and designers used to have to code via artifacts and ingenious combinations of existing elements. Here's a quick look at what's relevant to creating a webpage with HTML5.

Building the page layout with HTML5

In the beginning of the web, most pages were designed as a text documents—meaning that their content developed vertically on a single logical column. Over the years, page layout became more and more sophisticated. Today, two-column and three-column layouts are much more common. In two- and three-column layouts, you also often find headers and footers surrounding the logical columns. Figure 2-1 shows the difference between the layouts at a glance.

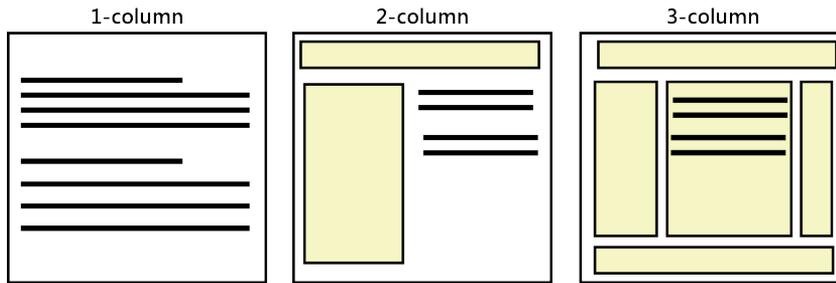


FIGURE 2-1 Different HTML page layouts.

Developers have been smart enough to build such complex layouts using basic HTML block elements such as *DIV*.



Note In HTML, a block element is an element whose content is rendered between two line breaks—one before and one after the content. Therefore, the content displays as a stand-alone “block” of content. One popular block element is *H1*, which renders some text as a first-level heading. The *DIV* element is another popular block element aimed at creating blocks out of any valid HTML content. Block elements are opposed to inline elements, namely elements whose content flows with the rest of the page with no line breaks applied.



Important Note that in this book, as well as in other books and articles, truly common HTML elements, such as `<div>` and `<h1>` often appear in text written without the brackets. However, the use of the angle brackets is mandatory if you are using those elements within HTML itself.

In HTML5, the multicolumn layout is recognized as a common layout and therefore gets full support via several new ad hoc markup elements.

Preparing the sample application

The examples you’ll be working with in this chapter are plain HTML pages showcasing some of the features available in HTML5 as supported in Internet Explorer 10. You won’t be creating an ad hoc Windows 8 application for each feature, but for this early example—to refresh what you saw in Chapter 1, “Using Visual Studio 2012 Express edition for Windows 8”—go ahead and create a container Windows 8 page that ties together all the links to the various standalone HTML5 pages.

Open up Visual Studio and create a new blank application. Name it **Htm15-Demos**. When done, add the following code to the body of *default.html* so that it serves as the main menu for navigating into all of the sample HTML5 pages you'll write throughout the chapter.

```
<body>
  <header>
    Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
<b>JavaScript</b>
    <hr />
    HTML5 samples
  </header>
  <div id="links">
    <a href="pages/multi.html">MULTI</a>
    <!-- Add here more links to HTML pages as we proceed in the chapter. -->
  </div>
  <footer>
    <hr />
    Dino Esposito | Francesco Esposito
  </footer>
</body>
```

Figure 2-2 shows the aspect of the resulting application. By clicking the links (such as, the MULTI link in the figure below) you force the operating system to open the webpage into Internet Explorer 10—the native browser in Windows 8.



FIGURE 2-2 The home page of the sample application for this chapter.

From now on, you'll be creating plain HTML5 pages and adding an anchor tag `<a>` to the body of *default.html*.

From generic blocks to semantic elements

A large share of websites out there have a common layout that includes header and footer, as well as a navigation bar on the left of the page. More often than not, these results are achieved by using *DIV* elements styled to align to the left or the right.

Let's add a new HTML page to the project: right-click the project node in Solution Explorer and choose Add | New Item from the subsequent flyout menu. What you get next is the window shown in Figure 2-3. From that window, you then choose a new HTML page and save it as *multi.html*.

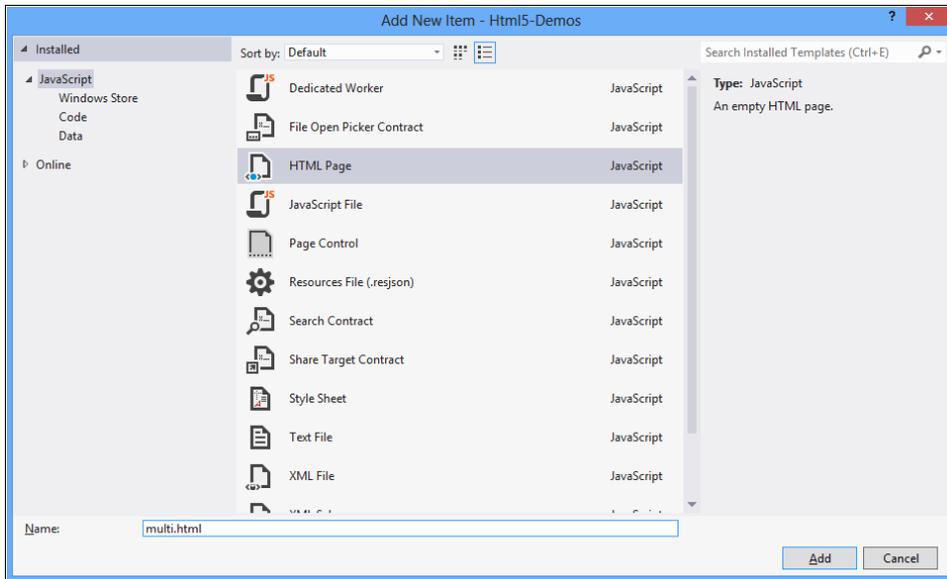


FIGURE 2-3 Creating a new HTML page in Visual Studio.

Next, from within Visual Studio double-click the newly created HTML page and replace the content with the following markup.

```
<!DOCTYPE html>
<html>
  <head>
    <title>MULTI-COLUMN LAYOUT</title>
  </head>
  <body>
    <a href="/default.html">Back</a>
    <hr />
    <div id="page">
      <div id="header">
        Header of the page
      <hr />
    </div>
    <div id="navigation-bar">
```

```

    <ul>
      <li> Home </li>
      <li> Find us </li>
      <li> Job opportunities </li>
    </ul>
  </div>
  <div id="container">
    <div id="left-sidebar">
      Left sidebar
      <ul>
        <li> Article #1 </li>
        <li> Article #2 </li>
        <li> Article #3 </li>
      </ul>
    </div>
    <div id="content">
      This is the main content of the page
    </div>
    <div id="right-sidebar">
      Right sidebar
    </div>
  </div>
  <div id="footer">
    <hr />
    Footer of the page
  </div>
</div>
</body>
</html>

```

The *id* attribute of the *DIV* elements are given self-explanatory names that help with understanding their intended role. Therefore, the HTML page includes header, navigation bar, footer, and a three-column layout in between the element named container. Figure 2-4 shows how *multi.html* renders on Internet Explorer.

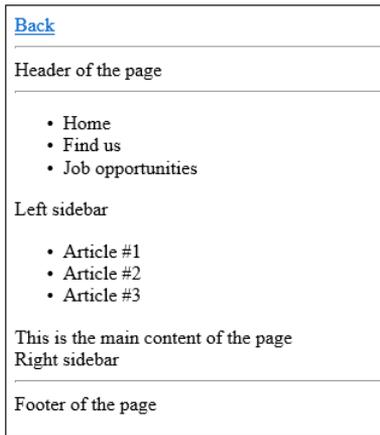


FIGURE 2-4 The *multi.html* page as it is rendered by Internet Explorer 10.

The preceding markup alone, however, doesn't produce the expected results and the page doesn't really show any multicolumn layout. For that, you need to add ad hoc graphic styles to individual *DIV* elements to make them float and anchor to the left or right edge. You add graphic style to an HTML page using CSS markup, placed in a CSS file. The next chapter provides a quick summary of CSS. The real point of this demo is a little different.

As you can see, each *DIV* element is made distinguishable from others only by the name of the *id* attribute. Yet, each *DIV* element plays a clear role that makes it fairly different from others—header is different from footer, and both are different from left or right sidebars.

Header and footer elements

HTML5 brings a selection of new block elements with specific names and clear behavior. The set of new elements was determined by looking at the most common layouts used by page authors. For example, in HTML5 header and footer are new plain block elements you use to indicate a header and footer. Similar elements exist for most of the semantic elements in the previous listing. Here's how you can rewrite the page *multi.html* using only HTML5-specific elements. Name this page **multi5.html**. The listing below shows the content of the body tag for the new page.

```
<header>
  Header of the page
  <hr />
</header>
<nav>
  <a href="#"> Home </a>
  <a href="#"> Find us </a>
  <a href="#"> Job opportunities </a>
</nav>
<article>
  <aside>
    Left sidebar
```

```

    <ul>
      <li> Article #1 </li>
      <li> Article #2 </li>
      <li> Article #3 </li>
    </ul>
  </aside>
<article>
  <h1>Article #1</h1>
  <hr />
  <section> Introduction </section>
  <section> First section </section>
  <section> Second section </section>
</article>
<aside>
  Right sidebar
</aside>
</article>
<footer>
  <hr />
  Footer of the page
</footer>

```

You can insert header and footer using specific elements with a very simple syntax, as below:

```

<header> Markup </header>
<footer> Markup </footer>

```

It is interesting to notice that you can have multiple header and footer elements in a HTML5 page. The most common use is to give the page a header and footer. However, you should consider these elements as blocks meant to represent heading content of a page or a section of a page and footers.

Section and article elements

HTML5 defines two similar-looking elements to represent the content of a page. The *<section>* element is slightly more generic, as it is meant to delimit a logical section of a HTML page. A logical section can be the content of a tab in a page designed as a collection of tabs.

At the same time, a logical section can also be a portion of the main content being displayed in the page. In this case, the *section* element is likely embedded in an *<article>* element.

```

<article>
  <h1>Article #1</h1>
  <hr />
  <section> Introduction </section>
  <section> First section </section>
  <section> Second section </section>
</article>

```



Note Elements such as `<section>`, `<article>`, `<header>`, and `<footer>` are semantic elements, in the sense that browsers treat them as block elements. If you look at final results, there's nearly no difference between semantic elements and plain *DIV* elements. The most significant difference is in the expressivity of the resulting markup. By using `<section>`, `<article>`, `<header>`, and `<footer>` elements, the resulting markup is much easier to read, understand, and maintain over time.

The *aside* element

A lot of HTML pages display part of their content on columns that lie side by side horizontally. The `<aside>` element has been introduced in HTML5 to quickly identify some content that is related to the content being displayed all around. The syntax of the `<aside>` element is straightforward:

```
<aside> Markup </aside>
```

A very common scenario where you might want to take advantage of the `<aside>` element is to define a sidebar in an *article* element and, more in general, to create multicolumn layouts for the content of the page or sections of the page.

The *nav* element

The `<nav>` element indicates a special section of the page content—the section that contains major navigation links. It should be noted that not all links you can have in a HTML page must be defined within a `<nav>` element. The `<nav>` element is reserved only for the most relevant links, such as those you would place on the main page navigation menu.

The syntax of the `<nav>` element is fairly intuitive. It consists of a list of `<a>` anchor elements listed within the `<nav>` element:

```
<nav>
  <a href="#"> Home </a>
  <a href="#"> Find us </a>
  <a href="#"> Job opportunities </a>
</nav>
```

The `<nav>` element plays an important role in HTML5 because it indicates the boundaries of the section of the page that contains navigation links. This allows special page readers—such as browsers for disabled users—to better understand the structure of the page and optionally skip some content.



Important All semantic elements in HTML5 are important in light of accessibility, and just for this reason, they should be considered for use in any webpage that has chances to be read by disabled users.

Miscellany of other new elements

Semantic block elements represent the largest family of new elements in HTML5. As mentioned, semantic elements are important not so much for the effect they produce in the page but because they increase the readability of the page significantly for developers, software, and especially browsers for disabled users.

Semantic block elements alone do not produce significant changes in the way in which HTML5-compliant browsers render the page. For example, to color and position a sidebar (that is, the *<aside>* element) where you like you still need to resort to CSS settings. However, using semantic elements reduces the noise of having too many generic *DIV* block elements whose role and scope is not immediately clear.

In addition to semantic elements, HTML5 provides a few new elements with an embedded behavior that couldn't be obtained in earlier versions of HTML without resorting to a combination of CSS, markup, and JavaScript. Let's see a few examples.

The *details* element

Many times you have small pieces of content in a page that you want to show or hide on demand. A good example is the title of some news and its actual content. Sometimes you want to display only the title but want to leave users free from clicking to expand the content and hide it to gain more space.

Before HTML5, you had to code all of this manually using a bit of HTML, CSS, and JavaScript code. In HTML5, the entire logic is left to the browser and all you have to do is type the following in an HTML page.

```
<details open>
  <summary>This is the title</summary>
  <div>
    This is the text of the news and was initially kept hidden from view
  </div>
</details>
```

The *<details>* element is interpreted by the browser and used to implement a collapsible panel. The *open* attribute indicates whether you want the content to be displayed initially or not. The *<summary>* child element indicates the text for the clickable placeholder, whereas the remaining content is hidden or shown on demand. Note that all parts of the *<details>* element can be further styled at will using CSS.



Important Although the Visual Studio editor recognizes the *<details>* element and even offers IntelliSense for it, the element is not supported by Internet Explorer 10. Other HTML5-compliant browsers, however, do support it—specifically the latest versions of Chrome and Opera.

The *mark* element

HTML5 also adds the `<mark>` element as a way to highlight small portions of text as if you were using a highlighter on a paper sheet. Using the `<mark>` element is easy; all you do is wrap some text in the `<mark>` element, as shown below:

The `<mark>DETAILS</mark>` element is not supported by Internet Explorer 10.

The entire text is rendered with default settings except the text enclosed in the `<mark>` element. Most HTML5 browsers have default graphical settings for marked text. Most commonly, these settings entail a yellow background. Needless to say, graphical effects of the `<mark>` element can be changed at will via CSS.

Figure 2-5 shows how the previous text looks using Internet Explorer 10.

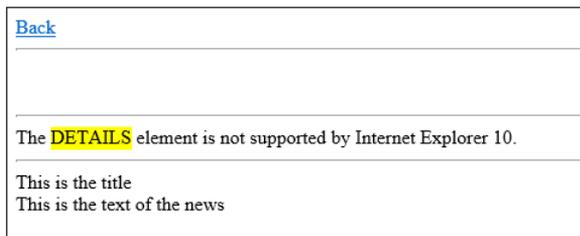


FIGURE 2-5 The *mark* element in action.

The *dataList* element

For a long time, HTML developers asked loudly for the ability to offer a list of predefined options for a text field. The use-case is easy to figure out. Imagine a user required to type the name of a city in a text field. As a page author, you want to leave the user free of entering any text; at the same time, though, you want to provide a few predefined options that can be selected and entered with a single click. Up until HTML5, this feature had to be coded via JavaScript, as HTML provided only two options natively: free text with no auto-completion or a fixed list of options with no chance of typing anything. The new `<datalist>` element fills the gap. Copy the following text to the body of a new HTML page named *datalist.html*.

```
<input list="cities" />
<datalist id="cities">
  <option value="Rome">
  <option value="New York">
  <option value="London">
  <option value="Paris">
</datalist>
```

In the example, the `<datalist>` element is bound to a particular input field—the input field named `cities`. It is interesting to notice that the binding takes place through a new attribute defined on the

`<input>` element: the `list` attribute. The attribute gets the name of a `<datalist>` element to be used as the source of the input options.

When the input field gets the input focus, then the content of the `<datalist>` element is used to autocomplete what the user is typing. Figure 2-6 shows the element in action on Internet Explorer.

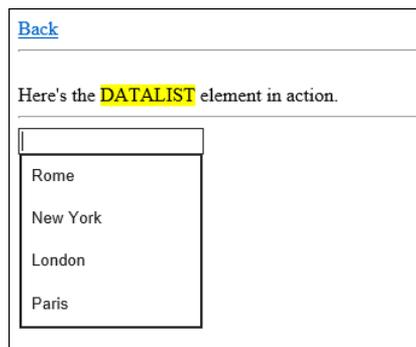


FIGURE 2-6 The `datalist` element in action.

Elements removed from older HTML versions

HTML5 adds some new elements, but also removes a few elements whose presence would only increase redundancy once combined with the new capabilities of CSS and new elements in HTML5.

The list of elements no longer supported most notably includes `frame` and `font` elements. It should be noted, though, that the `<iframe>` element remains available.

In addition, a few style elements such as `<center>`, `<u>`, and `<big>` are removed. The reason is that this functionality can be achieved easily through CSS. Probably due to the much larger use that page authors made over the years, HTML5 still supports elements such as `` (for bold text) and `<i>` (for italic text) that are logically equivalent to the now unsupported `<u>` and `<big>` elements.

Collecting data

HTML was originally devised to be a language for creating hypertext documents. Over the years, the language has been enriched with layout capabilities and basic features to collect data. Writing input forms to collect data from users proved to be a nontrivial task. One thing is to collect plain text; it is quite another to collect a date, a number, or an email address.

For too many years, HTML has only offered input text fields completely unable to distinguish numbers, dates, and email addresses from plain text. Subsequently, page developers were responsible for preventing users from typing unwanted characters and for client validation of the entered text.

With HTML5, a lot of this work has been pushed to the browser side. This means that by simply using a slightly more sophisticated set of elements developers can achieve the same level of form validation in a faster and safer way.

Adjusting input fields

In HTML5, you still create an input form by using the same markup elements you used from earlier versions of the language. In other words, the following markup will still give you the opportunity to upload any typed content to the given server.

```
<form action="http://www.yourserver.com/upload">
  <span> Your name </span>
  <input type="text" value="" />
  <input type="submit" value="Save" />
</form>
```

The `<input>` element is the element that inserts a graphical element (such as, an input box or a drop-down list) to collect some input data. You also use the `<input>` element to add a push button to start the submission process to the server. In HTML5, the `<input>` element comes with more options for the type of input boxes. For example, you can have date pickers, sliders, and search boxes offered by the browser. At the same time, the browser provides free form validation for most common scenarios, such as when a field is required and can't be left empty by the user.

New input types

If you look at the HTML5 syntax of the `<input>` element, the major difference with past versions is the list of values now allowed for the `type` attribute. Table 2-1 lists some of the new input types supported in HTML5.

TABLE 2-1 HTML5 specific values for the `type` attribute

Value	Description
<i>color</i>	Meant to let the browser display any UI that allows entry of a color. Note: This input type is not supported on Internet Explorer 10.
<i>date</i>	Meant to let the browser display any UI that allows entry of a date.
<i>email</i>	Meant to let the browser display any UI that allows entry of an email address.
<i>number</i>	Meant to let the browser display any UI that allows entry of a numeric input.
<i>range</i>	Meant to let the browser display any UI that allows entry of a numeric input.
<i>search</i>	Meant to let the browser display any UI that allows entry of a text to be searched for.
<i>tel</i>	Meant to let the browser display any UI that allows entry of a telephone number.
<i>time</i>	Meant to let the browser display any UI that allows entry of a time.
<i>url</i>	Meant to let the browser display any UI that allows entry of a URL.

Note that the list in Table 2-1 is incomplete and limited to input types that you can really find supported today on some web browsers. Other input types (for example, week) are part of the current HTML5 draft but are not implemented anywhere. You might want to refer to <http://www.w3schools.com/html5> for more details.

What you can really expect from browsers

What you have read in Table 2-1 represents the HTML5 standard that browsers are expected to comply with by the time the standard is officially approved and promulgated. At this time, Opera and Chrome are likely the browsers that offer the best support for the current draft of HTML5. In general, the support for HTML5 you can currently get from all browsers is not uniform.

The point is that HTML5 is not yet an approved standard and won't be definitive for a few more years. At the same time, companies are eager to use HTML5 goodies in real websites and applications. Websites and applications, however, require compliant browsers. As you can imagine, this situation determines a sort of catch-me game between browser vendors. Each new release of browsers improves support for the HTML5 standard or makes it more adherent to the actual standard, removing possible features resulting from misinterpretation of older drafts.

HTML5 is not going to be something unambiguously defined for a few more years. Internet Explorer 10—the browser you find integrated in Windows 8—improves significantly the support for HTML5 over Internet Explorer 9. Note that in Windows 8 you find two flavors of the Internet Explorer browser—the classic version, which builds on top of Internet Explorer 9, and the version with the native Windows 8 look-and-feel. This version, in particular, lacks the ability to run plug-ins (specifically, Flash and Silverlight) and organize favorites in folders.

Making input fields auto-focusable

HTML5 provides the definitive solution to a couple of common problems that developers faced for years and solved using a bit of JavaScript code. The first of these problems relates to giving the input focus to an input field.

Using JavaScript, you can tell the browser to assign the input focus to a particular input field upon display of the page. In HTML5, you can use a new attribute for the `<input>` element—the `autofocus` attribute. Try placing the following code in the body of a new HTML page named `autofocus.html`.

```
<form>
  <input type="text" value="Dino" />
  <input type="text" autofocus />
  <br />
  <input type="submit" value="Save" />
</form>
```

Save the page and display it in Internet Explorer. As Figure 2-7 shows, the cursor that indicates input focus is on the second field.

[Back](#)

Autofocus DEMO

Focus on second field.

Dino

Courtesy of [Start Here! Build Windows® 8 Applications with HTML5 and JavaScript](#)

FIGURE 2-7 The *autofocus* attribute in action.

Giving hints to users

Looking at Figure 2-7, it is quite hard to figure out which content goes in which field. Probably in a real-world page, one would use labels and a more sophisticated layout to make it easier for users to understand the expected content for each field. This is *just* the second problem I referred to a moment ago.

Recently, developers got into the groove of displaying a short text message in an input text field to instruct users. Before HTML5, this could only be accomplished by using a bit of JavaScript code. In HTML5, the new *placeholder* attribute makes it a lot easier and even more natural.

Create a new HTML page and save it as *placeholder.html*. Now edit the content of the body, as shown below:

```
<form>
  <input type="text" placeholder="First name" />
  <input type="text" placeholder="Last name" />
  <br />
  <input type="submit" value="Save" />
</form>
```

As Figure 2-8 shows, both empty fields now provide a hint to users about the expected content.

[Back](#)

Placeholder DEMO

Placeholder text

First name Last name

Courtesy of [Start Here! Build Windows® 8 Applications with HTML5 and JavaScript](#)

FIGURE 2-8 The *placeholder* attribute in action.

Form submission

Sometimes developers have no other option besides writing the same boilerplate code over and over again, no matter how annoying it is. A good example of boilerplate code that it would be great to stop writing is validation of input forms in HTML pages. Any data collected from an HTML input form should be carefully validated on the server before being used for some business tasks. However, some basic validation tasks can be easily delegated to the browser and commanded by the developers using markup instead of JavaScript code.

HTML5 helps reduce the amount of boilerplate code requested to build effective input forms. I've already mentioned the newest attributes of the HTML5 `<input>` element; the next step is to take a look at other attributes you can leverage to control the whole process of form submission, including ways to ensure that the user has not left required fields blank, and that the user input matches expected patterns. For example, if you ask for a phone number, the user shouldn't be allowed to enter something that couldn't possibly be a valid phone number.

Detecting required fields

By adding the *required* attribute to a `<input>` element, you tell the browser that the input field cannot be blank when the form that contains the `input` element is submitted. You use the *required* attribute only if the field is not considered optional.

Consider the following content of an HTML page named *required.html*:

```
<body>
<form>
  <input type="text" placeholder="Your PIN" required />
  <br />
  <input type="submit" value="Enter" />
</form>
</body>
```

When the user pushes the submit button and the text field is empty, the browser automatically denies the post and displays an error message. The specific content, form, and shape of the error message may change according to the browser; the overall behavior, though, is similar on all HTML5-compliant browsers. Figure 2-9 shows how Internet Explorer 10 deals with required fields left empty.



FIGURE 2-9 The *required* attribute in action.

HTML5 browsers allow you to customize the error message by using the *oninvalid* attribute, as shown below:

```
<form>
  <input type="text" placeholder="Your PIN" required
        oninvalid="this.setCustomValidity('PIN is mandatory')" />
  <br />
  <input type="submit" value="Enter" />
</form>
```



Note In general, you use the *oninvalid* attribute to specify any JavaScript code that should run when the content of an input field is invalid, either when that field value was required and left blank or when its content failed validation.

Validating against regular expressions

Table 2-1 lists popular new types of input fields supported by HTML5-compliant browsers. If your page is expected to collect a date, then you can use an input date field; likewise, you can use a numeric input field if you need to collect a number and so forth. But what if you intend to collect data formatted in a specific way that none of the predefined input types can guarantee? For example, what if you need users to enter a string with two letters followed by exactly six digits?

In HTML5, you can use the *pattern* attribute, as shown in the example below:

```
<form>
  <input type="text"
        placeholder="Your PIN"
        title="2 letters + 6 digits"
        pattern="[a-zA-Z]{2}\d{6}" />
  <br />
  <input type="submit" value="Enter" />
</form>
```

When you use the *pattern* attribute, Internet Explorer 10 requires that you also indicate the *title* attribute—usually used to add a tooltip to most HTML elements. The text of the *title* attribute is combined with a default static message to produce some feedback to the user when the content of the field is invalid.

Figure 2-10 shows how Internet Explorer 10 deals with patterns when the submitted content is invalid.

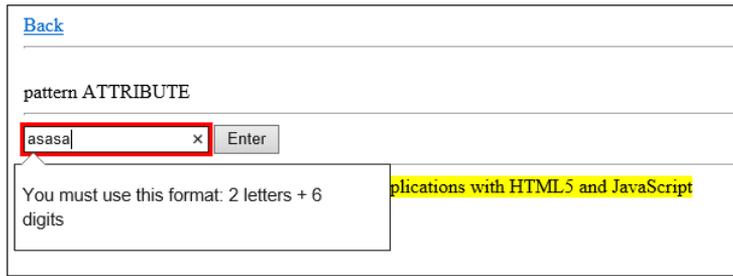


FIGURE 2-10 The *pattern* attribute in action.

The value of the *pattern* attribute has to be a *regular expression*. Regular expressions can get very complex; in fact, they're a topic worthy of a complete book, but learning the basics of regular expression use isn't too difficult. For more information on regular expressions, you can check out <http://www.regular-expressions.info>.

Forms and input validation

Each HTML form should contain a submit button; when the submit button is pushed the browser collects the content of the input fields and prepares to post it to the specified URL. Up until HTML5, the browser was not responsible for validating the content of the form. Developers, though, could hook up validation to the process using a bit of JavaScript code.

Validating a form entails checking that each input field in the form contains valid content. Although the HTML5 standard doesn't mandate browsers to validate the content of a form, this is indeed what happens by default with most browsers. HTML5 browsers give you a chance to disable validation on the entire form, but not on individual fields. You can disable form validation by using the *novalidate* attribute, as shown below in the file *novalidate.htm*:

```
<form novalidate>
  <input type="text" placeholder="Your PIN"
        title="2 letters + 6 digits"
        pattern="[a-zA-Z]{2}\d{6}" />
  <br />
  <input type="submit" value="Enter" />
</form>
```

In this case, the content of the form is submitted to the server regardless of the data held by input fields.

If the form contains multiple submit buttons, you can enable or disable validation on a per-button basis so that validation occurs if users, say, click the first button but not the second. To disable validation when the form is submitted via a particular submit button, you add the *formnovalidate* attribute as follows:

```
<input type="submit" value="..." formnovalidate />
```



Note The *formnovalidate* attribute overrides the form's *novalidate* attribute if both are set.

Multimedia elements

HTML5 offers two new markup elements that developers can use to play audio and video files from within webpages without resorting to external plug-ins such as Flash and Silverlight. The entire infrastructure to play audio and video (including graphical feedback to users) now is provided natively by the browser.

The *audio* element

To embed audio content into HTML documents, you use the `<audio>` element. The syntax is trivial, as the example below shows:

```
<audio src="/hello.mp3">
  <p>Your browser does not support the audio element.</p>
</audio>
```

Optionally, you can incorporate some markup in the body of the `<audio>` element to be used in case the browser can't deal successfully with the `<audio>` element. Next, you'll explore a bit more about how to embed audio in HTML5 pages.

Using the `<audio>` element

The `<audio>` element supports a variety of attributes, as listed in Table 2-2. Of these, the most important is *src*, which you use to point to the location of the actual audio stream.

TABLE 2-2 Attributes of the `<audio>` element

Attribute	Description
<i>autoplay</i>	Indicates that the audio will start playing as soon as the content is available to the browser.
<i>controls</i>	Instructs the browser to display audio controls, such as the play and pause buttons.
<i>loop</i>	Indicates that the audio will automatically restart after it is finished.
<i>preload</i> Note that the <i>preload</i> attribute is ignored if the <i>autoplay</i> attribute is also present.	Instructs the browser on how to load the audio content when the page loads. Allowed values are <i>none</i> , meaning that no content should be preloaded; <i>auto</i> , meaning that the entire content should be downloaded when the page loads; and <i>metadata</i> , meaning that only content metadata should be preloaded on page display.
<i>src</i>	Indicates the URL of the audio file, whether local or remote.

So far, I have referred to audio content in a rather generic way, without mentioning specific audio formats such as MP3 or WAV. This is a major problem with HTML5-compliant browsers: Not all browsers support the same set of audio formats by default (without resorting to external components).

The problem of codecs

An audio file is a sequence of bytes that codec software decodes for playing. An audio file, therefore, can be encoded in a variety of formats, each requiring an ad hoc codec—MP3, WAV, OGG, and more. More often than not, a codec is a piece of software that implements patented algorithms, so embedding a codec directly in the browser may pose copyright issues.

The current HTML5 standard doesn't make an official ruling about codecs, so deciding on the format to support will remain up to the browser vendors for now.

From a developer's perspective, this is not great news. Different browsers support different audio formats, leaving developers with the problem of working out the most effective way to play audio from the same page on different browsers.

Supported codecs

The simplest way of approaching the problem of multiple codecs is to provide multiple files so the browser can choose the most appropriate for its capabilities. In other words, instead of linking the `<audio>` element to just one audio file and codec, you link it to multiple sources. You no longer use the `src` attribute; instead, you resort to a set of `<source>` elements inside the `<audio>` element. Here's an example of playing an audio file using `<source>` elements:

```
<audio controls autoplay>
  <source src="hello.ogg" type="audio/ogg" />
  <source src="hello.mp3" type="audio/mp3" />
  <p>Your browser does not support the audio element.</p>
</audio>
```

The `<source>` elements link to different audio files. The browser will use the first format it knows how to support. While simple to implement, this approach is not free of issues—in the sense that it requires you to have each audio file available converted into multiple formats and stored on the server in multiple copies.

A basic guideline is that the OGG format is not subject to software patents. OGG will work in Firefox, Opera, and Chrome. To target Safari and Internet Explorer, you need to use MP3 encoding instead.

The *video* element

To embed video content into HTML documents, you use the `<video>` element. The syntax is just as trivial as what you have seen for the `<audio>` element:

```
<video src="/hello.mp4">
  <p>Your browser does not support the video element.</p>
</video>
```

Similarly, you can optionally incorporate some markup in the body of the `<video>` element to be used in case the browser can't deal with video successfully.

Using the `<video>` element

Table 2-3 presents the list of attributes you can use to customize the aspect and behavior of the `<video>` element in HTML5-compliant browsers.

TABLE 2-3 Attributes of the `<video>` element

Attribute	Description
<i>autoplay</i>	Indicates that the video will start playing as soon as the content is available to the browser.
<i>controls</i>	Instructs the browser to display video controls such as the play and pause buttons.
<i>height</i>	Indicates the desired height of the video player in the HTML document.
<i>loop</i>	Indicates that the video will automatically restart after it is finished.
<i>muted</i>	Indicates that the video sound should be muted off.
<i>poster</i>	Instructs the browser to display a specified image while the video content is downloading, or until the user chooses to play the video.
<i>preload</i> Note that the <i>preload</i> attribute is ignored if the <i>autoplay</i> attribute is also present.	Instructs the browser on how to load the video content when the page loads. Allowed values are <i>none</i> , meaning that no content should be preloaded; <i>auto</i> , meaning that the entire content should be downloaded when the page loads; and <i>metadata</i> , meaning that only content metadata should be preloaded on page display.
<i>src</i>	Indicates the URL of the video file to play, whether local or remote.
<i>width</i>	Indicates the desired width of the video player in the HTML document.

It is highly recommended that you always set both width and height in a `<video>` element. This helps the browser to reserve enough space while rendering the page. In addition, you should always set width and height to the real size of the video clip you plan to incorporate. If you downsize the video player, you force the browser to do even more work. Keep in mind that downsizing a video won't save the user any download time. If you have a video that is too large for the page, you should resize it with an ad hoc program first, and then link it using its new size.

Figure 2-11 shows how Internet Explorer 10 renders a `video` element.

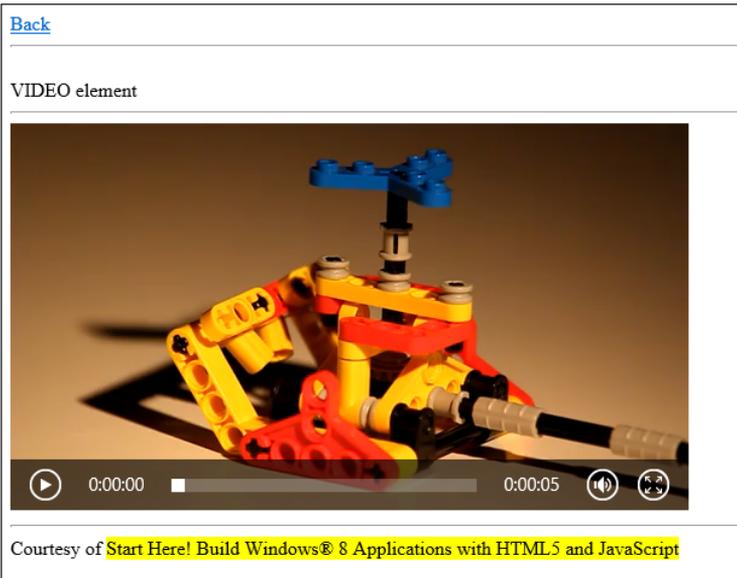


FIGURE 2-11 The *video* element in action.

Supported codecs

When it comes to codecs, video suffers from the same issues that audio does. Therefore, it requires the same workaround.

You should not use the *src* attribute unless you are well aware of the concrete risk that the video may not be playable on some browsers. To gain the widest support from HTML5-compliant browsers, you should use the *<source>* element. Here's the rewritten content of the sample *video.html* file:

```
<video controls width="320" height="240">
  <source src="/sample.ogv" type="video/ogg" />
  <source src="/sample.mp4" type="video/mp4" />
  <p>Your browser does not support the video element.</p>
</video>
```

Just as for audio, the *<source>* elements link to different video clips and the browser will use the first format it knows how to support. As a guideline, you should plan to have an MP4-encoded video for Internet Explorer and Safari, and OGG for all other browsers.

Summary

HTML has been around for a couple of decades, but it only recently underwent significant syntax changes. The new HTML5 specification clears out some obsolete elements and adds new markup elements for specific (and common) tasks. New elements have stronger semantics that make it obvious what they are for—such as *header*, *footer*, *menu*, *section*, and more.

These new elements, however, live side by side with older and, semantically speaking, more generic elements such as *DIV*. The result is that sometimes you have two or more ways to achieve the same rendering—using direct HTML5 elements or a combination of more generic elements. If you plan to target HTML5 browsers, using new elements keeps your markup easier to read and understand—in a word, simpler.

The purpose of this book is to build Windows 8 applications, as opposed to classic websites, which makes the differences between older HTML and HTML5 unimportant: this book uses HTML5 all the way through. However, if your goal is to build a website for the general public, then integrating HTML5 in the markup of the pages is much more difficult. For web applications, you will need to deal with browser differences and ensure that the behavior is uniform across major browsers.

Index

Symbols

- : (colon)
 - preceding pseudo-classes, 67
 - in style attribute, 48
 - in style command, 51
 - , (comma), in compound selectors, 69
 - { } (curly brackets)
 - enclosing object literal, 80
 - in <style> element, 49
 - .(dot)
 - in compound selectors, 69
 - preceding CSS classes, 53
 - () (parentheses), enclosing immediate functions or objects, 82
 - + (plus sign), in compound selectors, 69
 - # (pound symbol)
 - preceding element IDs, 20, 52
 - preceding hexadecimal RGB values, 55
 - > (right angle bracket), in compound selectors, 69
 - ;(semi-colon)
 - in style attribute, 48
 - in style command, 51
 - [] (square brackets), enclosing attribute pseudo-class, 68
-
- ## A
- <a> element, HTML, 26
 - in <nav> element, 31
 - pseudo-classes for, 67–68
 - about page, for Settings, 225–227
 - accessibility, semantic elements for, 31, 32
 - Account picture provider, 205
 - action attribute, <form> element, 35
 - activated event, app object, 115, 117, 119, 134
 - active pseudo-class, anchors, 67–68
 - alpha channel for color transparency, 56
 - anchor pseudo-classes, 67–68
 - anchors. *See* <a> element, HTML
 - AND keyword, media queries, 71
 - anonymous functions, JavaScript, 81, 88–89
 - App bar, 103, 108, 109–112
 - AppBar object, 110
 - AppBar widget, 107
 - application objects, persisting, 235–247
 - appending data to files, 240
 - deleting files, 240
 - reading files, 243–247
 - serialization formats for, 240–243, 245–247
 - writing files, 237–240
 - Application.PageControlNavigator object, 150
 - applications. *See also* Windows 8 applications
 - prior to Windows 8, standard mode for, 98–99
 - types of, 11
 - application settings, persisting, 228–230, 248–249
 - ApplicationViewState object, 178
 - app object
 - onactivated event, 115, 117, 119, 134
 - oncheckpoint event, 118
 - onready event, 165
 - onsettings event, 224
 - Array object, JavaScript, 75
 - arrays, binding list for, 167–168
 - <article> element, HTML, 30–31
 - <aside> element, HTML, 31
 - asynchronous functions, 101–102
 - attribute pseudo-classes, 68–69
 - <audio> element, HTML, 41–42
 - audio files
 - choosing device to play to, 205
 - embedding in HTML, 41

autofocus attribute, <input> element

- format for, 42
- playing, 41
- autofocus attribute, <input> element, 36–37
- autoplay attribute
 - <audio> element, 41
 - <video> element, 43
- AutoPlay event, 205
- AutoPlay provider, 205

B

- background-attachment property, 58
- background-color property, 57
- background gradients, 170
- background-image property, 57–58
- background-repeat property, 57–58
- Background task provider, 205
- background tasks, 119
- BadgeLogo.png file, 123
- <big> element, HTML, 34
- binding
 - data to UI, 112–116
 - for arrays, 167–168
 - for TodoList project example, 134–135, 138–146
 - two-way binding, 144–146
 - for Video project example, 167–170
 - handlers to events, 19, 137–138
- binding list, 167–168
- Bing SDK, 312–318
- bitmap images
 - as background of HTML elements, 57
 - for shared data, 217
- Blank App template, 11
- block elements, 25, 31, 60
- <body> element, HTML, 15
- books and publications
 - Start Here! Learn HTML5 (Microsoft Press), 2
 - Start Here! Learn HTML (Microsoft Press), 24
 - Start Here! Learn JavaScript (Microsoft Press), 2
- border property, 63, 64
- boxing model, 63–66
- browsers, HTML5 support in, 36
- building application, 141
- built-in objects, JavaScript, 75
- <button> element, HTML, 137

C

- Cache updater contract, 205
- callback functions, JavaScript, 85–88
- CameraCaptureUI object, 294, 295
- Camera settings provider, 205
- Cascading Style Sheet. *See* CSS
- <center> element, HTML, 34
- certificate, for application, 124
- certification of application, 346–349
- Chakra runtime environment, 74
- Charms bar, 103, 108
 - accessing, 202
 - contracts used in, 202–203
 - list of available printers in, enabling, 301–302
- checked pseudo-class, <input> element, 68
- checkpoint event, app object, 118
- class attribute, HTML, 53
- classes, CSS, 51, 53
- clipboard, alternative to. *See* Share contracts
- cloud, 107
- codecs
 - audio files, 42
 - video files, 44
- code, readability of, 121
- colon (:)
 - preceding pseudo-classes, 67
 - in style attribute, 48
 - in style command, 51
- color property, 56
- colors, 55–58
 - background color, 57–58
 - defining, 55–56
 - foreground color, 56
 - gradients, 57, 170
 - transparency of, 56
- comma (,), in compound selectors, 69
- compound selectors, 69
- constructor functions, JavaScript, 83–85
- Contact provider, 205
- contract-based callback functions, JavaScript, 87–88
- contracts, 106, 201–206. *See also* extensions
 - Cache updater contract, 205
 - Charms bar elements using, 202–203
 - consuming services using, 204
 - File picker contract, 204, 206–216
 - file extensions allowed for, 211

- file name for, 211–213
- folder selections only, 216
- multiple selections for, 215
- open option for, 210, 214–216, 244
- save option for, 210–213, 237–238
- starting location for, 210–211
- suggested file name for, 211
- unsnapping required for, 209–210
- list of, 204–205
- Play to contract, 205
- Print contract, 301–305
- publishing services using, 204
- Search contract, 203, 205
- Settings contract, 203, 205, 224–233
 - about page for, 225–227
 - persisting application settings, 228–230
 - privacy page for, 225
 - Save button not recommended, 233
 - settings page for, 227–233
- Share contracts, 203, 204, 216–224
 - application targets for, 224
 - conditional sharing, 223
 - data format for, 217
 - HTML format for, 221–222
 - if application doesn't support, 218
 - plain text format for, 219–220
 - programmatically sharing, 223
 - Share source contract, 204
 - Share target contract, 204
- controls attribute
 - <audio> element, 41
 - <video> element, 43
- CSS (Cascading Style Sheet), 47–48
 - boxing model, 63–66
 - classes defined in, 51, 53
 - colors, setting, 55–58
 - compound selectors, 69
 - default file for, in project, 13–14, 17
 - editing, 17–18
 - element IDs, referencing, 20, 52
 - embedded styles, 49–50, 52
 - external CSS files, 50–51
 - floating elements, 61–63
 - for global settings, 128
 - hierarchy of styles applied, 53–54
 - history of, 48
 - inline-block elements, 62–63

- inline styling, 48–49
- media attribute, 70–71, 306–307
- media queries, 70–71, 194–195
- for page-specific settings, 128
- pseudo-classes, 66–69
- role of, in application, 23
- selectors (elements to style), 51–55, 69
- style commands, 51
- tag name, referencing, 53
- text fonts, setting, 58–60
- for UI theme, 124
- version of, 48
- Css folder, 123
- curly brackets ({})
 - enclosing object literal, 80
 - in <style> element, 49
- custom objects, JavaScript, 83–85

D

- database, persisting data to, 235, 251
- data binding, 112–116
 - for arrays, 167–168
 - for TodoList project example, 134–135, 138–146
 - two-way binding, 144–146
 - for Video project example, 167–170
- data collection. *See* forms
- <datalist> element, HTML, 33–34
- DataPackage object, 217
- datarequested event, 223
- DataTransferManager object, 217, 218
- data-win-bind attribute, element, 114
- data-win-control attribute, <div> element, 127, 135
- Date object, JavaScript, 75
- DatePicker widget, 107, 135
- debugging, 141–143
- debug mode, running applications in, 16–17
- decodeUri function, JavaScript, 76
- default.css file, 127–128
- default.html file, 124–125
- default.js file, 134
- design of user interface. *See* UI (user interface)
- desktop mode, Windows 8, 177
- <details> element, HTML, 32
- developer account, 336–340
 - cost of, 339

device-centric design

- creating, 7–9, 336–339
 - for free applications, 336–339
 - for paid applications, 339–340
 - personal or business, 337–338
 - renewing, 8
 - device-centric design, 104
 - devices
 - choosing, for playing files, 205
 - connecting, AutoPlay event for, 205
 - GPS device, 309–318
 - latitude and longitude, detecting, 309–312
 - mapping location data, 312–318
 - user permission for, 311
 - printer, 301–309
 - CSS media attribute for, 306–307
 - Print contract, 301–305
 - printing specific content, 305–309
 - templates for, 307–309
 - user interface for, 304–305
 - synchronizing settings across, 249
 - webcam, 291–301
 - capability for, setting in manifest file, 293–294
 - capturing photos from, 294–297
 - configuring, 294
 - displaying photos from, 298
 - saving photos from, 298–301
 - user permission for, 296
 - Devices functions, 101
 - DirectX functions, 101
 - disabled pseudo-class, <input> element, 68
 - disabled users. *See* accessibility
 - display modes, 60–63
 - display property, 60–61, 62–63
 - <div> element, HTML, 19, 25
 - data-win-control attribute, 127, 135
 - id attribute, 27–28
 - document object
 - getElementById method, 111, 112
 - querySelectorAll method, 112
 - querySelector method, 112
 - DOM (Document Object Model), 23, 112
 - dot (.)
 - in compound selectors, 69
 - preceding CSS classes, 53
- E**
- ECMA-262 standard, 74
 - element IDs, 20, 28, 52
 - embedded styles, 49–50, 52
 - enabled pseudo-class, <input> element, 68
 - encodeURIComponent function, JavaScript, 76
 - Enterprise Sideload, 349
 - equality operators, JavaScript, 76
 - errors, 141–142. *See also* debugging
 - escape function, JavaScript, 76
 - eval function, JavaScript, 76
 - event handlers, JavaScript, 19, 89, 90, 137–138
 - events
 - binding to handlers, 19, 89, 90, 137–138
 - daterequested event, 223
 - onactivated event, app object, 115, 117, 119, 134
 - oncheckpoint event, app object, 118
 - onready event, app object, 165
 - onresize event, window object, 184, 188, 195
 - onsettings event, app object, 224
 - example projects
 - FlickrPhotoViewer project, 279–290
 - displaying photos, 284–285, 289–290
 - downloading JSON data for, 282–283
 - Flickr URL parameters, 281
 - invalid JSON data, handling, 285–289
 - Gallery project, 149–164
 - detail page for photos, 156–159
 - FlipView widget for, 152–156
 - Navigation App template for, 149–151
 - persisting information with, 160–161
 - photo template, 155–156
 - zooming images, 161–164
 - HelloWin8 project, 12–16
 - HelloWin8-Step2 project, 16–20
 - Html5-Demos project, 26–34
 - InstantPhoto project, 292–301
 - configuring webcam, 294
 - displaying photos from webcam, 298
 - saving photos from webcam, 298–301
 - webcam capability, setting, 293–294
 - LatLong project, 309–318
 - address information, determining, 316–318
 - latitude and longitude, detecting, 309–312
 - mapping location data, 312–318
 - RssReader project, 266–278
 - displaying data, 275–278
 - manifest file, 271–272
 - parsing data, 272–275
 - reading remote data, 266–272
 - SnapMe project, 182–195
 - TilesDemo project

- creating Live tiles, 323–324
 - notification object for, 324–326
- ToDoList project, 122–146
 - about page for, 225–227
 - building application, 141
 - CSS files for, 127–129
 - debugging, 141–143
 - deleting tasks, 262–264
 - editing tasks, 260–262
 - file picker for, 206–213
 - files and folders for, 122–124
 - form for, 133–146
 - header and footer for, 125–127
 - Live tiles for, 326–333
 - logos for, 130–132
 - manifest file, 129–130
 - persisting data, 236–247, 251–264
 - saving tasks, 259–260
 - settings page for, 227–233
 - sharing data from, 216–224
 - splash screen for, 132
 - WinJS references in, 124–125
- Video project, 165–174
 - adapting to view states, 196–200
 - binding data, 167–170
 - playing videos, 172–174
 - SemanticZoom widget for, 165–172
- exceptions, 142. *See also* debugging
- Extensible Application Markup Language. *See* XAML
- extensions, 205–206

F

- file access, 102
- File activation provider, 205
- file formats
 - audio files, 42
 - handling, File activation provider for, 205
 - images, 153, 294
 - serialization formats, 240–243
 - for shared data, 217
 - video files, 44, 174
- FileIO object, 237, 240
- FileOpenPicker object, 210
- File picker contract, 204, 206–216. *See also* Cache
 - updater contract
 - file extensions allowed for, 211
 - file name for, 211–213
 - folder selections only, 216
 - multiple selections for, 215
 - open option for, 210, 214–216, 244
 - save option for, 210–213, 237–238
 - starting location for, 210–211
 - suggested file name for, 211
 - unsnapping required for, 209–210
- FileSavePicker object, 210
- filled view state, 180–182
- first-child pseudo-class, 69
- Fixed Layout App template, 11
- flexible boxes, 190–193
- FlickrPhotoViewer project example, 279–290
 - displaying photos, 284–285, 289–290
 - downloading JSON data for, 282–283
 - Flickr URL parameters, 281
 - invalid JSON data, handling, 285–289
- FlipView widget, 107, 152–156
- float property, 61–62
- fluid layouts, 189–200
 - CSS media queries for, 194–195
 - flexible boxes for, 190–193
- Flyout widget, 107
- focus pseudo-class, <input> element, 68
- FolderPicker object, 216
- element, HTML, 34
- font-family property, 58–59
- font-size property, 58–59
- fonts, setting, 58–60
- font-style property, 59–60
- font-weight property, 59–60
- <footer> element, HTML, 29–30, 125–127
- foreground color, 56
- <form> element, HTML, 137
 - action attribute, 35
 - novalidate attribute, 40
- formnovalidate attribute, <input> element, 40–41
- forms, 34–41
 - input fields
 - adding, 35
 - focus on, assigning, 36–37
 - hints for, 37
 - list of values for, 33–34
 - types of, 35
 - input validation
 - disabling, 40–41
 - error message for invalid fields, 39
 - regular expressions for, 39–40
 - required fields, 38–39
 - submit button for, 40–41
 - for ToDoList project example, 133–146

<frame> element, HTML

- <frame> element, HTML, 34
- full-screen view states, 178–179
 - landscape mode, 178
 - portrait mode, 178–179
 - rotation, detection of, 179
- functions, JavaScript, 81–89
 - anonymous functions, 81, 88–89
 - callback functions, 85–88
 - calling, 81
 - constructor functions, 83–85
 - defining, 81
 - extending objects using, 83
 - immediate functions, 82–83
 - importing, with contracts, 106–107
 - named functions, 81–82
 - parameters for, 81–82
- functions, WinRT API, 101–102

G

- Gallery project example, 149–164
 - detail page for photos, 156–159
 - FlipView widget for, 152–156
 - Navigation App template for, 149–151
 - persisting information with, 160–161
 - photo template, 155–156
 - zooming images, 161–164
- generateNumber() function example, 19
- getElementById method, document object, 111, 112
- global namespace, JavaScript, 78
- global object, JavaScript, 75–76
- global variables, JavaScript, 78, 92–93
- GPS device, 309–318
 - latitude and longitude, detecting, 309–312
 - mapping location data, 312–318
 - user permission for, 311
- gradients, 57, 170
- Grid App template, 11, 122

H

- <h1> element, HTML, 25
- handlers, binding to events, 19, 137–138
- <header> element, HTML, 29–30, 125–127
- height attribute, <video> element, 43
- height property
 - for HTML elements, 65–66
 - for media queries, 71
- HelloWin8 project example, 12–16

- HelloWin8-Step2 project example, 16–20
- highlighting text. *See* <mark> element, HTML
- hoisting, in JavaScript, 79
- home.css file, 149
- home folder, 149
- home.html file, 149
- home.js file, 149
- home screen for application, 124, 124–125, 129
- hover pseudo-class, anchors, 67–68
- HTML5, 23–24. *See also* specific elements
 - block elements, 25, 31, 60
 - browser support for, 36
 - default file for, in project, 13–14
 - display modes, 60–63
 - editing, 14–15, 16–17
 - element IDs
 - defining, 28, 52
 - referencing in CSS, 20, 52
 - inline elements, 25, 60–61
 - input forms, 34–41
 - linking JavaScript to, 89–92
 - multimedia, 41–44
 - page layout, 24–31
 - removed (unsupported) elements, 34
 - resources for, 2
 - role of, in application, 23
 - semantic elements, 31, 32
- Html5-Demos project example, 26–34
- HtmlControl widget, 108
- HTML format, for shared data, 217, 221–222
- HTTP requests
 - errors in, handling, 268–269
 - timeouts for, 270
 - WinJS.xhr object for, 266–270
- hyperlinks
 - <a> element for, 26
 - in <nav> element, 31
 - pseudo-classes for, 67–68
 - for navigation, 148
- Hypertext Markup Language. *See* HTML5

I

- icons, compared to Live tiles, 319
- id attribute, <div> element, 27–28
- <iframe> element, HTML, 173–174
- images. *See also* webcam
 - as background of HTML elements, 57
 - displaying, choosing device for, 205

- file formats for, 153, 294
- gallery of. *See* Gallery project example
- logos for application, 123, 130–132
- Paint.NET tool, 130–131
- splash screen, 123, 132
- for user account, 205
- zooming, 161–164
- Images folder, 123
- immediate functions, JavaScript, 82–83
- immediate objects, JavaScript, 82–83
- Infinity value, JavaScript, 75
- inline-block elements, 62–63
- inline elements, 25, 60–61
- inline styling, 48–49
- innerHTML property, 113
- innerText property, 113, 115
- <input> element, HTML, 35–37
 - autofocus attribute, 36–37
 - formnovalidate attribute, 40–41
 - list attribute, 33–34
 - onclick attribute, 90
 - oninvalid attribute, 39
 - pattern attribute, 39–40
 - placeholder attribute, 37
 - pseudo-classes for, 68
 - required attribute, 38–39
 - type attribute, 35, 136, 162
- input fields
 - adding, 35
 - focus on, assigning, 36–37
 - hints for, 37
 - list of values for, 33–34
 - types of, 35
- input forms. *See* forms
- input validation, 38–41
 - disabling, 40–41
 - error message for invalid fields, 39
 - regular expressions for, 39–40
 - required fields, 38–39
 - submit button for, 40–41
- InstantPhoto project example, 292–301
 - configuring webcam, 294
 - displaying photos from webcam, 298
 - saving photos from webcam, 298–301
 - webcam capability, setting, 293–294
- Internet Client capability, 271
- Internet data. *See* remote data
- isFinite function, JavaScript, 76
- isNaN function, JavaScript, 76
- ISO/IEC 16262:2011 standard, 74

- isolated storage
 - in TodoList project, 251–264
 - types of, 250–251

J

- JavaScript, 73–74
 - built-in objects, 75
 - Chakra runtime environment for, 74
 - default file for, in project, 13–14
 - equality operators, 76
 - event handlers, 89, 90
 - functions, 81–89
 - global namespace, 78
 - global object, 75–76
 - hoisting, 79
 - as interpreted language, 74
 - linking to HTML pages, 89–92
 - maintaining state, 93
 - null values, 76
 - objects, 79–80
 - primitive types, 74–75
 - resources for, 2
 - role of, in application, 23
 - standards for, 74
 - undefined values, 76
 - variables, 77–79, 92–93
- jQuery library, 91–92
- JScript .NET, 1
- Js folder, 124
- JSLint tool, 78
- JSON data, 242–243, 278–290
 - deserializing, 245–247
 - displaying, 284–285
 - downloading, 282–283
 - invalid, handling, 285–289
- JSONLint website, 286–288

K

- kernel, operating system, 100
- Knockout library, 116

L

- landscape mode, 178
- last-child pseudo-class, 69
- latitude and longitude, detecting, 309–312

LatLong project example

- LatLong project example, 309–318
 - address information, determining, 316–318
 - latitude and longitude, detecting, 309–312
 - mapping location data, 312–318
- lifecycle of applications, 116–119
- <link> element, HTML, 51
- links
 - <a> element for, 26
 - in <nav> element, 31
 - pseudo-classes for, 67–68
 - for navigation, 148
- liquid layouts. *See* fluid layouts
- list attribute, <input> element, 33–34
- ListView widget, 108
- Live tiles, 103, 319–322
 - adding to applications, 326–333
 - background agents for, 333
 - compared to icons, 319
 - creating, 323–326
 - moving, 320–321
 - notification object for, 324–326, 333
 - resizing, 322
 - templates for, 324–325, 328–329
 - unpinning applications from, 322
- local disk, storing application data to, 249–250. *See also* File picker contract
- local isolated storage, 250
- localization, 94
- local machine, running applications on, 15, 16
- localSettings object, 228–230, 248–249
- local variables, JavaScript, 77
- location (path) for projects, 12
- Lock screen, background task information on, 119
- Logo.png file, 123
- logos for application, 123, 130–132
- longitude and latitude, detecting, 309–312
- loop attribute
 - <audio> element, 41
 - <video> element, 43

M

- manifest file, 124, 129–130
 - enabling Internet capability, 271–272
 - enabling Pictures Library access, 298–299
 - enabling webcam capability, 293–294
- margin property, 63, 64–65
- <mark> element, HTML, 33
- master-detail view. *See* SemanticZoom widget

- Math object, JavaScript, 75
- max-height property
 - for HTML elements, 66
 - for media queries, 71
- max-width property
 - for HTML elements, 66
 - for media queries, 71
- media attribute, 70–71, 306–307
- Media functions, 101
- media queries, CSS, 70–71, 194–195
- Menu widget, 108
- Microsoft Bing SDK, 312–318
- min-height property
 - for HTML elements, 66
 - for media queries, 71
- min-width property
 - for HTML elements, 66
 - for media queries, 71
- MP3 files, 42
- MP4 files, 44
- ms-flex-XXX attributes, 191
- multimedia, 41–44
 - audio, 41–42
 - video, 42–44
- muted attribute, <video> element, 43

N

- named functions, JavaScript, 81–82
- NaN value, JavaScript, 75
- <nav> element, HTML, 31
- navigation
 - home screen for, 148–149, 150–151
 - hyperlinks for, 148
 - model for, 147–149
 - Navigation App template for, 11, 122, 149–151
 - page fragments for, 148–149, 151
 - persisting information with, 160–161
- Navigation App template, 11, 122, 149–151
- navigator.js file, 150
- .NET Framework, 5
 - compared to WinJS library, 102
 - history of, 1
 - version of, 5
- Networking functions, 101
- new constructor, JavaScript, 80, 84
- New Project link, 10
- notification object, 324–326, 333
- NOT keyword, media queries, 71

novalidate attribute, <form> element, 40
 null values, JavaScript, 76
 numberButtonClick() function example, 19
 Number object, JavaScript, 75
 number type, JavaScript, 75

O

object literals, JavaScript, 80
 objects

- application, persisting, 235–247
 - appending data to files, 240
 - deleting files, 240
 - reading files, 243–247
 - serialization formats for, 240–243, 245–247
 - writing files, 237–240
- JavaScript, 79–80
 - adding members to, 79–80, 83
 - built-in objects, 74–75
 - creating new instances of, 80
 - custom objects, creating, 83–85
 - global object, 75–76
 - immediate objects, 82–83
 - prototype for, 80, 83
 - wrapper objects, 74, 78

OGG files, 42, 44
 onactivated event, app object, 115, 117, 119, 134
 oncheckpoint event, app object, 118
 onclick attribute, <input> element, 90
 on-demand elements in UI, 108
 oninvalid attribute, <input> element, 39
 onready event, app object, 165
 onresize event, window object, 184, 188, 195
 onsettings event, app object, 224
 operating system kernel, 100
 orientation property, media queries, 71

P

<p> element, HTML, 15
 Package.appxmanifest file, 124, 129–130
 packaging applications, 342–349
 padding property, 63, 64
 PageControlNavigator object, 150
 PageControl widget, 108
 page layout, HTML5 elements for, 24–31
 pages folder, 149
 Paint.NET tool, 130–131
 parameters for functions, 81–82

parentheses (()), enclosing immediate functions or objects, 82
 parseFloat function, JavaScript, 76
 parseInt function, JavaScript, 76
 path (location) for projects, 12
 pattern attribute, <input> element, 39–40
 pattern matching. *See* regular expressions
 period (.)

- in compound selectors, 69
- preceding CSS classes, 53

persisting data

- application objects, 235–247
 - appending data to files, 240
 - deleting files, 240
 - reading files, 243–247
 - serialization formats for, 240–243, 245–247
 - writing files, 237–240
- application settings, 228–230, 248–249
- to database, 235, 251
- to isolated storage
 - in *ToDoList* project, 251–264
 - types of, 250–251
- to local disk, 249–250. *See also* File picker contract
- to localSettings object, 248
- to removable storage devices, 250

Pictures Library, 298–299
 placeholder attribute, <input> element, 37
 Play to contract, 205
 plus sign (+), in compound selectors, 69
 portrait mode, 178–179
 poster attribute, <video> element, 43
 pound symbol (#)

- preceding element IDs, 20, 52
- preceding hexadecimal RGB values, 55

preload attribute

- <audio> element, 41
- <video> element, 43

Presentation functions, 101
 primitive types, JavaScript, 74–75
 Print contract, 301–305
 printer, 301–309

- CSS media attribute for, 306–307
- Print contract for, 301–305
- printing specific content, 305–309
- templates for, 307–309
- user interface for, 304–305

PrintManager object, 303
 Print settings provider, 205

privacy page, for Settings

- privacy page, for Settings, 225
- product key, Visual Studio 2012, 5–7
- programming languages, 10, 99. *See also* JavaScript projects. *See also* example projects
 - building applications from, 15–16
 - creating, 10–16
 - CSS for. *See* CSS (Cascading Style Sheet)
 - files and folders in, 122–124
 - creating, 125
 - with Navigation App template, 149–150
 - opening, 14
 - HTML for. *See* HTML5
 - JavaScript for. *See* JavaScript
 - location (path) for, 12
 - templates for
 - choosing, 10–11
 - predefined, list of, 11
- Promise object, 118
 - error handling with, 268–269
 - timeouts with, 270
 - WinJS.xhr object returning, 268
- proportional design. *See* fluid layouts
- prototype common property, JavaScript, 80
- pseudo-classes, CSS, 66–69
- publishing applications, 335, 340–349
 - cost of, 335, 339
 - developer account for, 336–340
 - naming application for, 340–342
 - packaging application for, 342–349
 - for sideloaded applications, 349
 - submitting for certification, 346–349

Q

- querySelectorAll method, document object, 112
- querySelector method, document object, 112

R

- random numbers, generating, 18–19
- readability of code, 121
- ready event, app object, 165
- redundancy in UI, zero, 106
- References folder, 123
- RegExp object, JavaScript, 75
- regular expressions, for input validation, 39–40
- Release mode, 342–343
- remote data
 - JSON data, 278–290

- RSS data, 265–278
 - WinJS.xhr object for, 266–270
- remote machine, running applications on, 15
- removable storage devices, 250
- required attribute, <input> element, 38–39
- resize event, window object, 184, 188, 195
- resources. *See* books and publications; website resources
- responsiveness of UI, 105
- rgba function, 56
- RGB colors, 55
- rgb function, 55
- right angle bracket (>), in compound selectors, 69
- roaming isolated storage, 251, 259–264
- roamingSettings object, 249
- rotation of device, detecting, 179
- RSS data, 265–278
 - displaying RSS data, 275–278
 - downloading, 266–272
 - parsing data, 272–275
- RssReader project example, 266–278
 - displaying RSS data, 275–278
 - manifest file, 271–272
 - parsing data, 272–275
 - reading remote data, 266–272
- RTF format, for shared data, 217
- run time errors, 141. *See also* debugging

S

- Save picker contract. *See* File picker contract
- <script> element, HTML, 90–91
- Search contract, 203, 205
- <section> element, HTML, 30–31
- <select> element, HTML, 136
- selectors, 51–55, 69
- semantic elements, 31, 32
- SemanticZoom widget, 108, 165–172
- semi-colon (;)
 - in style attribute, 48
 - in style command, 51
- serialization formats, 240–243
- services, 204. *See also* contracts
- Settings contract, 203, 205, 224–233
 - about page for, 225–227
 - persisting application settings, 228–230
 - privacy page for, 225
 - Save button not recommended, 233
 - settings page for, 227–233

- settings event, app object, 224
- Settings panel, 203, 224
 - adding items to, 224–227
 - dismissing, 226
 - running code before and after, 232
- Share contracts, 203, 204, 216–224
 - application targets for, 224
 - conditional sharing, 223
 - data format for, 217
 - HTML format for, 221–222
 - if application doesn't support, 218
 - plain text format for, 219–220
 - programmatic sharing, 223
 - Share source contract, 204
 - Share target contract, 204
- sideloaded applications, 9, 349
- simulator, running applications in, 15, 16
- SmallLogo.png file, 123
- SnapMe project example, 182–195
- snapped view state, 179–182, 188–189
- Solution Explorer panel, 14
- <source> element, HTML, 42
- element, HTML
 - data-win-bind attribute, 114
 - innerText property, 115
- splash screen, 123, 132
- SplashScreen.png file, 123
- Split App template, 11, 122
- SQLite database, 251
- square brackets ([]), enclosing attribute pseudo-class, 68
- src attribute
 - <audio> element, 41
 - <script> element, 91
 - <video> element, 43
- stack, Windows 8, 100–101
- standard desktop mode, Windows 8, 177
- standard mode, Windows 8, 98–99
- Start Here! Learn HTML5 (Microsoft Press), 2
- Start Here! Learn HTML (Microsoft Press), 24
- Start Here! Learn JavaScript (Microsoft Press), 2
- start page. *See* home screen for application
- state (application), maintaining, 93
 - between pages, 160–161
 - between view states, 182–183
- states (lifecycle) of applications, 116–119, 161
- states (view) of application, 177–178
 - adapting content to, 187–200
 - behavior of application not affected by, 181, 188
 - detecting changes in, 184–187

- filled view state, 180–182
- full-screen view states, 178–179
 - landscape mode, 178
 - portrait mode, 178–179
 - rotation, detection of, 179
- maintaining application state between, 182–183
- snapped view state, 179–182, 188–189
 - restrictions on, 180
 - unsnapping programmatically, 188–189
- stopFloating element, 62
- StorageFile object, 212, 300
- Storage functions, 101
- StoreLogo.png file, 123
- String object, JavaScript, 75
- string type, JavaScript, 75
- style attribute, HTML, 48–49
- <style> element, HTML, 49–50
- style sheets. *See* CSS (Cascading Style Sheet)
- submit button, for forms submission, 40–41
- <summary> element, HTML, 32
- suspended state, 118–119
 - Background task provider for, 205
 - devices having, 161
- syntax errors, 141. *See also* debugging

T

- tags, HTML. *See* HTML5
- templates
 - choosing, 10–11, 121–122
 - predefined, list of, 11
- temporary isolated storage, 251
- <textarea> element, HTML, 135
- text fonts, setting, 58–60
- text format, for shared data, 217, 219–220
- theme for application UI, 124
- theme for Visual Studio, 13
- TileNotification object, 324–326
- TilesDemo project example
 - creating Live tiles, 323–324
 - notification object for, 324–326
- TimePicker widget, 108
- ToDoList project example, 122–146
 - about page for, 225–227
 - building application, 141
 - CSS files for, 127–129
 - debugging, 141–143
 - deleting tasks, 262–264
 - editing tasks, 260–262

ToggleSwitch widget

- file picker for, 206–213
- files and folders for, 122–124
- form for, 133–146
- header and footer for, 125–127
- Live tiles for, 326–333
- logos for, 130–132
- manifest file, 129–130
- persisting data, 236–247, 251–264
- saving tasks, 259–260
- settings page for, 227–233
- sharing data from, 216–224
- splash screen for, 132
- WinJS references in, 124–125
- ToggleSwitch widget, 108
- Tools menu, Options
 - default path, setting, 12
 - theme, setting, 13
- Tooltip widget, 108, 137, 138
- touch input, 103, 105
- transparency for colors, 56
- type attribute, `<input>` element, 35, 136, 162
- typeof operator, JavaScript, 76, 80
- types, JavaScript, 74–75

U

- `<u>` element, HTML, 34
- ui-dark.css file, 124
- ui-light.css file, 124
- UI (user interface), 102–107
 - App bar, 103, 109–112
 - binding data to, 112–116
 - Charms bar, 103
 - cloud use by, 107
 - dark or light theme for, 124
 - design principles for, 104–107
 - device-centric, 104
 - importing functions for, 106
 - Live tiles, 103
 - on-demand elements in, 108
 - platform influencing, 106
 - redundancy in, zero, 106
 - responsiveness of, 105
 - touch-enabled, 103, 105
 - visual widgets, 107–108
- undefined values, JavaScript, 76
- unescape function, JavaScript, 76
- URI format, for shared data, 217
- URL, accessing with WinJS.xhr object, 266–270
- user account, image for, 205
- user interface. *See* UI (user interface); *See* Windows 8 UI

V

- variables, JavaScript, 77–79
 - global variables, 78, 92–93
 - hoisting, 79
 - local variables, 77
- var keyword, JavaScript, 77, 78, 79
- `<video>` element, HTML, 42–44
- Video project example, 165–174
 - adapting to view states, 196–200
 - binding data, 167–170
 - playing videos, 172–174
 - SemanticZoom widget for, 165–172
- videos
 - choosing device to play to, 205
 - formats for, 44, 174
 - playing, 172–174
- ViewBox widget, 108
- view states, 177–178
 - adapting content to, 187–200
 - behavior of application not affected by, 181, 188
 - detecting changes in, 184–187
 - filled view state, 180–182
 - full-screen view states, 178–179
 - landscape mode, 178
 - portrait mode, 178–179
 - rotation, detection of, 179
 - maintaining application state between, 182–183
 - snapped view state, 179–182, 188–189
 - restrictions on, 180
 - unsnapping programmatically, 188–189
- visited pseudo-class, anchors, 67–68
- Visual Studio 2012
 - configuring, 5–9
 - developer account for, 7–9, 336–339
 - editions of, 2, 3
 - installing, 3–5
 - product key for, 5–7
 - programming language for, 10
 - projects in. *See* projects
 - theme for, setting, 13
 - Windows Store account for, 9

W

- W3C (World Wide Web Consortium), 48
- WAV files, 42
- web browsers. *See* browsers
- webcam, 291–301
 - capability for, setting in manifest file, 293–294
 - capturing photos from, 294–297
 - configuring, 294
 - displaying photos from, 298
 - saving photos from, 298–301
 - user permission for, 296
- website resources
 - Bing SDK, 313
 - CSS predefined colors, 55
 - flexible boxes, 190
 - jQuery library, 91
 - JSLint tool, 78
 - JSONLint, 286
 - SQLite database, 251
 - Visual Studio Express download, 3
 - Visual Studio features, 3
 - Windows 8 download, 3
- WideLogo.png file, 123
- widgets, WinJs library, 107–108
- width attribute, <video> element, 43
- width property, 65–66
- width property, media queries, 71
- window.print method, 309
- Windows 8
 - editions of, 2, 3
 - installing, 2–3
 - stack, 100–101
 - standard desktop mode, 98–99, 177
- Windows 8 applications, 98–99
 - background tasks for, 119
 - building, 15–16, 141
 - certificate for, 124
 - debugging, 141–143
 - exiting, 16
 - fonts for, changing, 58
 - home screen for, 124, 124–125, 129
 - income and taxes from, 340
 - installing
 - methods of, 336
 - requirements for, 8, 9
 - launching, 117–118, 165
 - lifecycle of, 116–119
 - localization for, 94
 - logos for, 123, 130–132
 - maintaining state, 93
 - naming, 340–342
 - packaging, 124, 342–349
 - platforms for, 344–345
 - programming languages for, 10, 99
 - projects for. *See* projects
 - publishing, 335, 340–349
 - cost of, 335, 339
 - developer account for, 336–340
 - naming application for, 340–342
 - packaging application for, 342–349
 - for sideloaded applications, 349
 - submitting for certification, 346–349
 - resuming, 119
 - running
 - in debug mode, 16–17
 - on local machine, 15, 16
 - on remote machine, 15
 - in simulator, 15, 16
 - splash screen for, 123, 132
 - states of, 116–119, 161
 - storing data in, 102
 - suspending, 118
 - tools required for, 1–5, 23–24
 - unpinning from Start screen, 322
 - user interface for. *See* Windows 8 UI
 - view states of, 177–178
 - adapting content to, 187–200
 - behavior of application not affected by, 188
 - detecting changes in, 184–187
 - filled view state, 180–182
 - full-screen view states, 178–179
 - maintaining application state between, 182–183
 - snapped view state, 179–182, 188–189
- Windows 8 SDK, 5
- Windows 8 UI, 102–107
 - App bar, 103, 109–112
 - binding data to, 112–116
 - Charms bar, 103
 - cloud use by, 107
 - dark or light theme for, 124
 - design principles for, 104–107
 - device-centric, 104
 - importing functions for, 106
 - Live tiles, 103
 - on-demand elements in, 108
 - platform influencing, 106
 - redundancy in, zero, 106
 - responsiveness of, 105

Windows App Certification Kit

- touch-enabled, 103, 105
- visual widgets, 107–108
- Windows App Certification Kit, 346–347
- Windows clipboard, alternative to. *See* Share contracts
- Windows Live ID, 7
- Windows Runtime environment. *See* WinRT environment
- Windows Store account, 9
- Windows Store applications. *See* Windows 8 applications
- WinJS.Application object, 160
- WinJS.Binding.List object, 167
- WinJs library, 1, 23, 74
 - compared to .NET Framework, 102
 - references to, in default.html file, 124
 - visual widgets, 107–108
- WinJS logger, 112
- WinJS.Navigation object, 159
- WinJS.UI.AppBarCommand object, 110
- WinJS.UI.AppBar object, 110
- WinJS.UI.FlipView widget, 152–156
- WinJS.UI.SemanticZoom widget, 165–172
- WinJS.UI.Tooltip object, 137
- WinJS.xhr object, 266–270
 - downloading JSON data, 282–283
 - downloading RSS data, 266–272
 - errors in, handling, 268–269
 - manifest file settings for, 271
 - parameters for, 267–268
 - Promise object returned by, 268
 - timeouts with, 270
- WinRT (Windows Runtime) environment, 1, 97–102
 - API for, 100–102
 - stack, 100–101
 - system classes, WinJs library for, 1
- World Wide Web Consortium. *See* W3C
- wrapper objects, JavaScript
 - for global namespace, 78
 - for primitive types, 74

X

- XAML (Extensible Application Markup Language), 99, 100
- XHR, WinJS.xhr object, 266–267
- Xxx_TemporaryKey.pfx file, 124

Y

- YouTube videos, playing, 173–174

Z

- zooming images, 161–164

About the authors



Dino Esposito, a long-time trainer and top-notch consultant, is the author of many popular books for Microsoft Press that have helped the professional growth of thousands of .NET developers and architects. Dino is the CTO of a fast-growing company that provides software and mobile services to professional sports, and currently is also a technical evangelist for JetBrains, where he focuses on Android and Kotlin development, and is a member of the team that manages WURFL—the database of mobile devices used by organizations such as Google and Facebook. Follow Dino on Twitter at @despos and on <http://software2cents.wordpress.com>.



Even though he's still a teenager (he's only 15), **Francesco Esposito** has accumulated significant experience with mobile application development for a variety of platforms, including iOS with Objective C and MonoTouch, Android via Java, Windows Phone, and even BlackBerry. He wrote most of the code for IBI12—the official multi-platform app for the Rome ATP Masters 1000 tennis tournament.

When not writing apps, hanging out with friends, or practicing water polo, he likes going to school, where his secret goal is to achieve the highest marks ever so he can get a scholarship to Harvard or just buy his own Surface tablet.

What do you think of this book?

We want to hear from you!

To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Tell us how well this book meets your needs—what works effectively, and what we can do better. Your feedback will help us continually improve our books and learning resources for you.

Thank you in advance for your input!

Microsoft[®]
Press