Microsoft

Programming in HTML5 with JavaScript and CSS3

Training Guide

Glenn Johnson

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2013933428
ISBN: 978-0-7356-7438-7

Printed and bound in the United States of America.

Fourth Printing: September 2015

Acquisitions Editor: Devon Musgrave
Developmental Editor: Devon Musgrave
Project Editor: Carol Dillingham
Editorial Production: nSight, Inc.
Technical Reviewer: Pierce Bizzaca; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Copyeditor: Kerin Forsyth
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Contents at a glance

	Introduction	*xxi*
CHAPTER 1	Getting started with Visual Studio 2012 and Blend for Visual Studio 2012	1
CHAPTER 2	Getting started with HTML5	29
CHAPTER 3	Getting started with JavaScript	65
CHAPTER 4	Getting started with CSS3	137
CHAPTER 5	More HTML5	205
CHAPTER 6	Essential JavaScript and jQuery	261
CHAPTER 7	Working with forms	311
CHAPTER 8	Websites and services	341
CHAPTER 9	Asynchronous operations	393
CHAPTER 10	WebSocket communications	415
CHAPTER 11	HTML5 supports multimedia	437
CHAPTER 12	Drawing with HTML5	459
CHAPTER 13	Drag and drop	507
CHAPTER 14	Making your HTML location-aware	539
CHAPTER 15	Local data with web storage	555
CHAPTER 16	Offline web applications	581
	Index	*621*

Contents

Introduction **xix**

Backward compatibility and cross-browser compatibility *xx*

System requirements *xx*

Practice exercises *xxi*

Acknowledgments *xxi*

Errata and book support *xxi*

We want to hear from you *xxii*

Stay in touch *xxii*

Chapter 1 Getting started with Visual Studio 2012 and Blend for Visual Studio 2012 **4**

Lesson 1: Visual Studio 2012 5

Visual Studio 2012 editions 5

Visual Studio 2012 support for HTML5 6

CSS3 support 7

JavaScript support 7

Exploring Visual Studio Express 2012 for Windows 8 8

Exploring Visual Studio Express 2012 for Web 12

Lesson summary 14

Lesson review 15

Lesson 2: Blend for Visual Studio 2012 16

Exploring Blend 16

Lesson summary 22

Lesson review 23

Practice exercises 23

Exercise 1: Hello World with Visual Studio Express 2012 for Windows 8 24
Exercise 2: Hello World with Visual Studio Express 2012 for Web 25
Exercise 3: Hello World with Blend 27

Suggested practice exercises 28

Answers. 29

Chapter 2 Getting started with HTML5 32

Lesson 1: Introducing HTML5 32
Understanding HTML, XHTML, and HTML5 33
Introducing semantic markup 34
Working with elements 35
Creating an HTML document 43
Lesson summary 45
Lesson review 46

Lesson 2: Embedding content. 47
Embedding HTML by using inline frames 47
Working with hyperlinks 49
Adding images to your HTML document 50
Embedding plug-in content 53
Lesson summary 55
Lesson review 56

Practice exercises 56
Exercise 1: Create a simple website by using Visual Studio Express for Web 56
Exercise 2: Create additional pages 59
Exercise 3: Embedding Content 61

Suggested practice exercises 64

Answers. 65

Chapter 3 Getting started with JavaScript 65

Lesson 1: Introducing JavaScript 66
Understanding JavaScript 66

Understanding the role of data 67
Using statements 71
Working with functions 73
Scoping variables 77
Nesting functions and nested local variable scoping 78
Converting to a different type 78
Conditional programming 80
Implementing code loops 84
Handling errors 87
Lesson summary 88
Lesson review 88

Lesson 2: Writing, testing, and debugging JavaScript 89
Hello World from JavaScript 90
Using the script tag 100
Handling browsers that don't support JavaScript 101
Inline JavaScript vs. external JavaScript files 102
Placing your script elements 102
Using the Visual Studio .NET JavaScript debugger 103
Lesson summary 107
Lesson review 107

Lesson 3: Working with objects 108
Working with arrays 108
Accessing DOM objects 112
Lesson summary 120
Lesson review 121

Practice exercises 121
Exercise 1: Create a calculator webpage 121
Exercise 2: Add the QUnit testing framework 123

Suggested practice exercises 133

Answers. 134

Chapter 4 Getting started with CSS3 137

Lesson 1: Introducing CSS3 137
Defining and applying a style 139

Adding comments within a style sheet 139
Creating an inline style 140
Creating an embedded style 140
Creating an external style sheet 141
Lesson summary 144
Lesson review 145

Lesson 2: Understanding selectors, specificity, and cascading 145
Defining selectors 146
Understanding the browser's built-in styles 159
Extending browser styles with user styles 159
Working with important styles 159
How do styles cascade? 160
Using specificity 161
Understanding inheritance 162
Lesson summary 163
Lesson review 164

Lesson 3: Working with CSS properties. 165
Working with CSS colors 166
Working with text 173
Working with the CSS box model 175
Setting the border, padding, and margin properties 176
Positioning <*div*> elements 178
Using the float property 186
Using the clear property 189
Using the box-sizing property 190
Centering content in the browser window 193
Lesson summary 193
Lesson review 194

Practice exercises 194
Exercise 1: Add a style sheet to the calculator project 195
Exercise 2: Clean up the web calculator 197

Suggested practice exercises 201

Answers. 202

Chapter 5 More HTML5 205

Lesson 1: Thinking HTML5 semantics 205

Why semantic markup? 206

Browser support for HTML5 206

Creating semantic HTML5 documents 207

Creating an HTML5 layout container 207

Controlling format by using the <*div*> element 213

Adding thematic breaks 213

Annotating content 213

Working with lists 221

Lesson summary 228

Lesson review 229

Lesson 2: Working with tables 229

Table misuse 230

Creating a basic table 230

Adding header cells 231

Styling the table headers 232

Declaring the header, footer, and table body 233

Creating irregular tables 238

Adding a caption to a table 241

Styling columns 241

Lesson summary 242

Lesson review 243

Practice exercises 243

Exercise 1: Add a page layout to the calculator project 244

Exercise 2: Add styles to the calculator layout 246

Exercise 3: Cleaning up the web calculator 252

Suggested practice exercises 257

Answers. 258

Chapter 6 Essential JavaScript and jQuery 261

Lesson 1: Creating JavaScript objects 262

Using object-oriented terminology 262

Understanding the JavaScript object-oriented caveat 263
Using the JavaScript object literal pattern 263
Creating dynamic objects by using the factory pattern 265
Creating a class 266
Using the prototype property 271
Debating the prototype/private compromise 274
Implementing namespaces 276
Implementing inheritance 278
Lesson summary 283
Lesson review 284

Lesson 2: Working with jQuery. 285
Introducing jQuery 285
Getting started with jQuery 286
Using jQuery 287
Enabling JavaScript and jQuery IntelliSense 291
Creating a jQuery wrapper for a DOM element reference 294
Adding event listeners 295
Triggering event handlers 295
Initializing code when the browser is ready 295
Lesson summary 296
Lesson review 296

Practice exercises 297
Exercise 1: Create a calculator object 297

Suggested practice exercises 307

Answers. 308

Chapter 7 Working with forms 311

Lesson 1: Understanding forms 311
Understanding web communications 312
Submitting form data to the web server 316
Sending data when submitting a form 316
Using the *<label>* element 318
Specifying the parent forms 319
Triggering the form submission 319

Serializing the form 321

Using the autofocus attribute 321

Using data submission constraints 322

Using POST or GET 322

Lesson summary 323

Lesson review 324

Lesson 2: Form validation 324

Required validation 325

Validating URL input 327

Validating numbers and ranges 329

Styling the validations 330

Lesson summary 330

Lesson review 330

Practice exercises 331

Exercise 1: Create a Contact Us form 331

Exercise 2: Add validation to the Contact Us form 335

Suggested practice exercises 337

Answers. 338

Chapter 8 Websites and services 341

Lesson 1: Getting started with Node.js. 341

Installing Node.js 342

Creating Hello World from Node.js 342

Creating a Node.js module 344

Creating a Node.js package 345

Fast forward to express 354

Starting with express 354

Lesson summary 363

Lesson review 363

Lesson 2: Working with web services 364

Introducing web services 364

Creating a RESTful web service by using Node.js 366

Using AJAX to call a web service 368

Cross-origin resource sharing 380

Lesson summary 381
Lesson review 382
Practice exercises 382
Exercise 1: Create a website to receive data 382
Exercise 2: Create a web service to receive data 386
Suggested practice exercises 390
Answers. 391

Chapter 9 Asynchronous operations 393

Lesson 1: Asynchronous operations using jQuery and WinJS 393
Using a promise object 394
Creating jQuery promise objects by using $.Deferred() 395
Handling failure 397
Handling completion cleanup 397
Subscribing to a completed promise object 398
Chaining promises by using the pipe method 398
Parallel execution using $.when().then() 400
Updating progress 400
Conditional asynchronous calls 401
Lesson summary 402
Lesson review 403
Lesson 2: Working with web workers 404
Web worker details 404
Lesson summary 405
Lesson review 406
Practice exercises 406
Exercise 1: Implement asynchronous code execution 406
Suggested practice exercises 412
Answers. 413

Chapter 10 WebSocket communications 415

Lesson 1: Communicating by using WebSocket 415
Understanding the WebSocket protocol 416
Defining the WebSocket API 416

Implementing the WebSocket object 417
Dealing with timeouts 420
Handling connection disconnects 422
Dealing with web farms 422
Using WebSocket libraries 423
Lesson summary 424
Lesson review 424

Practice exercises 425
Exercise 1: Create a chat server 425
Exercise 2: Create the chat client 429

Suggested practice exercises 435

Answers. 436

Chapter 11 HTML5 supports multimedia 437

Lesson 1: Playing video 437
Video formats 438
Implementing the <*video*> element 438
Setting the source 439
Configuring the <*video*> element 441
Accessing tracks 441
Lesson summary 442
Lesson review 443

Lesson 2: Playing audio 443
Audio formats 444
The <*audio*> element 444
Setting the source 445
Configuring the <*audio*> element 445
Lesson summary 446
Lesson review 446

Lesson 3: Using the HTMLMediaElement object. .447
Understanding the HTMLMediaElement methods 447
Using HTMLMediaElement properties 447
Subscribing to HTMLMediaElement events 449
Using media control 450

Lesson summary 451
Lesson review 451

Practice exercises 452
Exercise 1: Create a webpage that displays video 452

Suggested practice exercises 455

Answers. 456

Chapter 12 Drawing with HTML5 459

Lesson 1: Drawing by using the <*canvas*> element 460
The <*canvas*> element reference 460
CanvasRenderingContext2D context object reference 460
Implementing the canvas 462
Drawing rectangles 463
Configuring the drawing state 465
Saving and restoring the drawing state 474
Drawing by using paths 475
Drawing text 488
Drawing with images 490
Lesson summary 494
Lesson review 495

Lesson 2: Using scalable vector graphics 495
Using the <*svg*> element 496
Displaying SVG files by using the <*img*> element 499
Lesson summary 501
Lesson review 502

Practice exercises 502
Exercise 1: Create a webpage by using a canvas 502

Suggested practice exercises 505

Answers. 506

Chapter 13 Drag and drop 507

Lesson 1: Dragging and dropping 507
Dragging 509
Understanding drag events 510

Dropping 511
Using the DataTransfer object 513
Lesson summary 516
Lesson review 516

Lesson 2: Dragging and dropping files 517
Using the FileList and File objects 517
Lesson summary 521
Lesson review 521

Practice exercises 522
Exercise 1: Create a number scramble game 522
Exercise 2: Add drag and drop to the game 526
Exercise 3: Add scramble and winner check 530

Suggested practice exercises 535

Answers. 536

Chapter 14 Making your HTML location-aware 539

Lesson 1: Basic positioning 540
Geolocation object reference 540
Retrieving the current position 541
Handling errors 543
Addressing privacy 544
Specifying options 544
Lesson summary 545
Lesson review 546

Lesson 2: Monitored positioning 546
Where are you now? How about now? 546
Calculating distance between samples 548
Lesson summary 549
Lesson review 549

Practice exercises 550
Exercise 1: Map your current positions 550

Suggested practice exercises 553

Answers. 554

Chapter 15 Local data with web storage 555

Lesson 1: Introducing web storage 555

Understanding cookies 556

Using the jQuery cookie plug-in 556

Working with cookie limitations 557

Alternatives to cookies prior to HTML5 557

Understanding HTML5 storage 558

Exploring localStorage 560

Using short-term persistence with *sessionStorage* *562*

Anticipating potential performance pitfalls 563

Lesson summary 564

Lesson review 564

Lesson 2: Handling storage events 565

Sending notifications only to other windows 566

Using the StorageEvent object reference 566

Subscribing to events 567

Using events with *sessionStorage* *568*

Lesson summary 568

Lesson review 568

Practice exercises 569

Exercise 1: Create a contact book by using *localStorage* *569*

Suggested practice exercises 578

Answers. 579

Lesson 1 579

Lesson 2 580

Chapter 16 Offline web applications 581

Lesson 1: Working with Web SQL 582

Considering the questionable longevity of Web SQL 582

Creating and opening the database 582

Performing schema updates 583

Using transactions 584

Lesson summary 588

Lesson review 589

Lesson 2: Working with IndexedDB 589

Using browser-specific code 590

Creating and opening the database 590

Using object stores 591

Using transactions 593

Inserting a new record 594

Updating an existing record 594

Deleting a record 595

Retrieving a record 595

Understanding cursors 596

Dropping a database 599

Lesson summary 599

Lesson review 600

Lesson 3: Working with the FileSystem API 600

Assessing browser support 601

Opening the file system 601

Creating and opening a file 602

Writing to a file 602

Reading a file 603

Deleting a file 604

Creating and opening a directory 604

Writing a file to a directory 605

Deleting a directory 605

Lesson summary 606

Lesson review 606

Lesson 4: Working with the offline application HTTP cache 607

Browser support 608

The cache manifest file 608

Updating the cache 609

Understanding events 610

Lesson summary 610
Lesson review 611
Practice exercises 611
Exercise 1: Modify a contact book to use IndexedDB 611
Suggested practice exercises 616
Answers. 617

Index *621*

Introduction

This training guide is designed for information technology (IT) professionals who develop or plan to develop HTML documents such as webpages or Windows Store applications. It is assumed that, before you begin using this guide, you are familiar with web development and common Internet technologies.

This book covers some of the topics and skills that are the subject of the Microsoft certification exam 70-480. If you are using this book to complement your study materials, you might find this information useful. Note that this book is designed to help you in the job role; it might not cover all exam topics. If you are preparing for the exam, you should use additional study materials to help bolster your real-world experience. For your reference, a mapping of the topics in this book to the exam objectives is included in the back of the book.

By using this training guide, you will learn how to do the following.

- Create a project using Visual Studio Express 2012 for Web.
- Create a project using Blend for Visual Studio 2012.
- Create a project using Visual Studio Express 2012 for Windows 8.
- Create an HTML document using semantic markup.
- Implement JavaScript functionality with your HTML documents.
- Use test-driven development techniques for writing JavaScript code.
- Create Cascading Style Sheets (CSS) that visually format your HTML document.
- Create HTML tables.
- Create JavaScript objects.
- Use jQuery to simplify JavaScript programming.
- Create HTML forms with validation.
- Create a Node.js website and web service.
- Call web services from JavaScript.
- Perform asynchronous JavaScript operations.
- Perform WebSocket communications.
- Play audio and video on a webpage.
- Draw with an HTML5 canvas.
- Use SVG image files.
- Perform drag and drop operations.
- Make your HTML location aware.
- Persist data on the browser client.

Backward compatibility and cross-browser compatibility

This book does not attempt to cover every difference between every version of every browser. Such a comprehensive discussion could easily yield a library of books.

Most of the code in this book is written using Internet Explorer 10, which is installed with Windows 8. In addition, many but not all the code examples were tested using the following browsers.

- Firefox 17.0.1
- Google Chrome 23.0.1271.97 m
- Opera 12.11
- Apple Safari 5.1.7

In most cases, if the other browsers were not compatible, there is a note stating so. This is especially true in the last chapters because web storage is still relatively new, and the requirements are still fluid.

The best way to see which features are available among browsers is to visit a website that is updated when new browser versions are released and HTML5 features are updated. The website *http://caniuse.com* is particularly good.

System requirements

The following are the minimum system requirements your computer needs to meet to complete the practice exercises in this book.

- Windows 8 or newer. If you want to develop Windows Store applications, you need Windows 8 on your development computer.

Hardware requirements

This section presents the hardware requirements for using Visual Studio 2012.

- 1.6 GHz or faster processor
- 1 GB of RAM (more is always recommended)
- 10 GB (NTFS) of available hard disk space
- 5400 RPM hard drive
- DirectX 9–capable video card running at 1024 × 768 or higher display resolution.
- Internet connectivity

Software requirements

The following software is required to complete the practice exercises.

- Visual Studio 2012 Professional, Visual Studio 2012 Premium, or Visual Studio 2012 Ultimate. You must pay for these versions, but in lieu of one of these versions, you can install the following free express versions.
    - Visual Studio Express 2012 for Web. Available from *http://www.microsoft.com/visualstudio/eng/downloads#d-express-web*.
    - Visual Studio Express 2012 for Windows 8. This installation also installs Blend for Visual Studio 2012. Available from *http://www.microsoft.com/visualstudio/eng/downloads#d-express-web*.

Practice exercises

This book features practices exercises to reinforce the topics you've learned. These exercises are organized by chapter, and you can download them from *http://aka.ms/TGProgHTML5/files*.

Acknowledgments

Thanks go to the following people for making this book a reality.

- To Carol Dillingham for your constructive feedback throughout the entire process of writing this book. Thanks for also having patience while the winter holiday months were passing, and my desire and ability to write was constantly interrupted.
- To Devon Musgrave for providing me the opportunity to write this book.
- To Kerin Forsyth for your hard work in making this book consistent with other Microsoft Press books and helping me with the delivery of this book.
- To Pierce Bizzaca for your technical reviewing skills.

To all the other editors and artists who played a role in getting my book to the public, thank you for your hard work and thanks for making this book venture a positive experience for me.

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 at Oreilly.com:

http://aka.ms/TGProgHTML5/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, send an email to Microsoft Press Book Support at *mspinput@microsoft.com*.

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

We want to hear from you

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

http://aka.ms/tellpress

The 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 at *http://twitter.com/MicrosoftPress*.

CHAPTER 1

Getting started with Visual Studio 2012 and Blend for Visual Studio 2012

Welcome to the world of HTML5, CSS3, and JavaScript! These technologies, commonly referred to as simply HTML5, can be used to develop applications for Windows and the web.

This chapter introduces you to the primary tools you need, Microsoft Visual Studio 2012 and Blend for Visual Studio 2012, which are used in the book's lessons. Visual Studio 2012 provides exciting new features. The chapters that follow introduce you to many features in Visual Studio 2012 and Blend.

Lessons in this chapter:

- Lesson 1: Visual Studio 2012 **2**
- Lesson 2: Blend for Visual Studio 2012 **13**

Before you begin

To complete this book, you must have some understanding of web development. This chapter requires the hardware and software listed in the "System requirements" section in the book's Introduction.

REAL WORLD **A CAREER OPPORTUNITY**

You're looking for a career in computer programming, but you don't know what technology to pursue. You want to learn a technology you can use at many companies to make yourself more marketable and to give you the flexibility to move between companies. What technology would you choose to give you this flexibility?

The Internet has exploded. Nearly every company has a website, so why not learn the web technologies?

HTML, CSS, and JavaScript are three closely coupled core web technologies that you can learn to increase your marketability and give you flexibility to choose the company for which you want to work. This is the beginning of your path toward your future career. Learn these technologies well, and you can expand into other programming technologies such as Silverlight, Flash, C#, and Visual Basic.

Lesson 1: Visual Studio 2012

Visual Studio 2012 is a highly useful tool for application development. Depending on the edition of Visual Studio you have, it can provide you with an integrated development environment (IDE) you can use for the entire project life cycle.

After this lesson, you will be able to:

- Identify the available versions of Visual Studio 2012 and the features of each.
- Start a project by using Visual Studio Express 2012 for Web.
- Start a project by using Visual Studio Express 2012 for Windows 8.

Estimated lesson time: 40 minutes

Visual Studio 2012 editions

The following is a list with short descriptions of the editions of Visual Studio that Microsoft offers.

- **Visual Studio Test Professional 2012** Provides team collaboration tools but not a full development IDE. This is ideal for testers, business analysts, product managers, and other stakeholders, but this is not an ideal edition for developers.
- **Visual Studio Professional 2012** Provides a unified development experience that enables developers to create multitier applications across the web, the cloud, and devices. This is an ideal edition for a lone developer or a small team of developers who will be developing a wide range of web, Windows, phone, and cloud applications.
- **Visual Studio Premium 2012** Provides an integrated application lifecycle management (ALM) solution and software development functions to deliver compelling applications for a unified team of developers, testers, and business analysts.
- **Visual Studio Ultimate 2012** Provides a comprehensive ALM offering for organizations developing, distributing, and operating a wide range of highly scalable software applications and services.
- **Visual Studio Team Foundation Server Express 2012** Provides the collaboration hub at the center of the ALM solution that enables small teams of up to five developers

to be more agile, collaborate more effectively, and deliver better software more quickly. Includes source code control, work item tracking, and build automation for software projects to deliver predictable results. This is free.

- **Visual Studio Express 2012 for Web** Provides the tools and resources to build and test HTML5, CSS3, ASP.NET, and JavaScript code and to deploy it on web servers or to the cloud by using Windows Azure. Best of all, it's free.
- **Visual Studio Express 2012 for Windows 8** Provides the core tools required to build Windows Store apps, using XAML and your choice of .NET language or HTML5, CSS3, and JavaScript. This is also free.

If you use Visual Studio Express 2012 for Web, you can work on web projects only, and you must choose a .NET language to start with, such as Visual Basic or C#. If you use Visual Studio Express 2012 for Windows 8, you can work on Windows Store applications only, but you can start with a JavaScript project, and you don't need to set up a website to create small applications. Blend for Visual Studio 2012 also provides the ability to create Windows Store applications with a JavaScript project.

The Visual Studio Express 2012 products are available free on the Microsoft website. You should download and install both Visual Studio Express 2012 for Windows 8 and Visual Studio Express 2012 for Web.

Visual Studio 2012 support for HTML5

Visual Studio .NET 2012 contains a new HTML editor that provides full support for HTML5 elements and snippets. Here is a list of some of the Visual Studio 2012 features that will make your development experience more enjoyable and productive. The new features will be demonstrated and explained later in this book when appropriate.

- **Testing** You can easily test your webpage, application, or site with a variety of browsers. Beside the Start Debugging button in Visual Studio 2012, you will find a list of all installed browsers. Just select the desired browser from the list when you are ready to test.
- **Finding the source of rendered markup** By using the new Page Inspector feature, you can quickly find the source of rendered markup. The Page Inspector renders a webpage directly within the Visual Studio IDE, so you can choose a rendered element, and Page Inspector will open the file in which the markup was generated and highlight the source.
- **Improved IntelliSense** Quickly find snippets and code elements. In the HTML and CSS editors, IntelliSense filters the display list as you enter text. This feature shows strings that match the typed text in their beginning, middle, or end. It also matches against initial letters. For example, "bw" will match "border-width."
- **Reusable Markup** You can easily create reusable markup by selecting the markup and extracting it to a user control.

- **Automatic Renaming** When you rename an opening or closing HTML tag, the corresponding tag is renamed automatically.

CSS3 support

Visual Studio .NET 2012 provides a new CSS editor that offers full support for CSS3, including support for cascading style sheets (CSS), hacks, and snippets for vendor-specific extensions to CSS.

- **Expandable Sections** Use the CSS editor to expand and collapse sections by clicking the plus or minus sign that is displayed to the left of each style entry.
- **Hierarchical Indentation** The CSS editor displays nested styles in a hierarchical fashion, which makes it much easier to read and understand the styles.
- **Add Comments** You can easily comment and uncomment blocks.
- **Color Selector** The CSS editor now has a color selector like the HTML editor.

JavaScript support

Visual Studio 2012 provides many new features to make the JavaScript developer experience more enjoyable and productive. The following is a list of some of the new features and enhancements.

- **Standards-based** Visual Studio 2012 incorporates the JavaScript features of ECMAScript 5 and the HTML5 document object model (DOM).
- **Improved IntelliSense** You can receive improved IntelliSense information for functions and variables by using new elements supported in triple-slash (///) code comments. New elements include *<var>* and *<signature>*. You can also view function signatures on the right side of the statement completion list.
- **Improved editor** JavaScript Editor implements smart indenting, brace matching, and outlining as you write code. For example, if you position your cursor to the left of an open curly brace, the open and closed curly braces are highlighted. This works with curly braces, square brackets, angle brackets, and parentheses.
- **Go To Definition** To locate function definitions in source code, you just right-click a function and then click Go To Definition. You can also put the cursor in the function, and then press the F12 key to open the JavaScript source file at the location in the file where the function is defined. (This feature isn't supported for generated files.)
- **IntelliSense from JavaScript comments** The new IntelliSense extensibility mechanism automatically provides IntelliSense when you use standard JavaScript comment tags (//).
- **Breakpoints** You now have more flexibility when setting a breakpoint. When a single line contains multiple statements, you can now break on a single statement.

- **Reference Groups** You can control which objects are available in global scope by using Reference Groups. Reference Groups is configured on the menu bar by navigating to Tools | Options | Text Editor | JavaScript | IntelliSense | References.
- **Drag-and-drop references** You can drag JavaScript files that have the .js extension from Solution Explorer to the JavaScript code editor, where they are added as references for Visual Studio to use to provide IntelliSense. When adding references by dragging and dropping, they are put at the top of the page in the code editor.

Exploring Visual Studio Express 2012 for Windows 8

When you start Visual Studio Express 2012 for Windows 8, the Start Page screen is displayed. Figure 1-1 shows the Start Page screen, which contains helpful information and links to learning and development resources. On the left side of the Start page are links to create a new project or open a new project. After you create at least one project, you'll see shortcut links to open any of your recent projects.

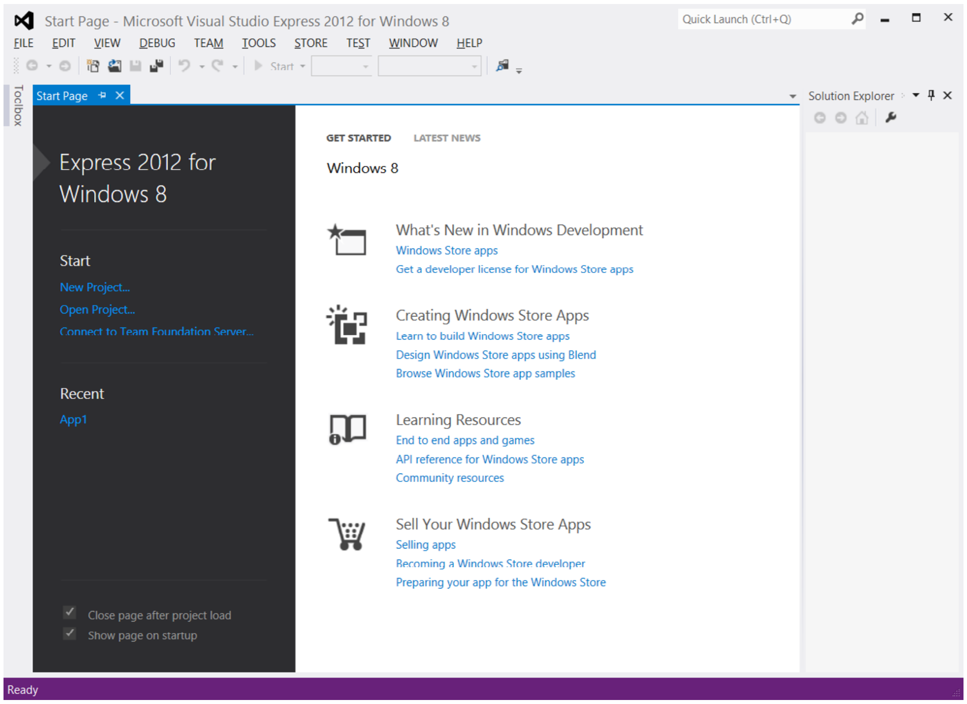
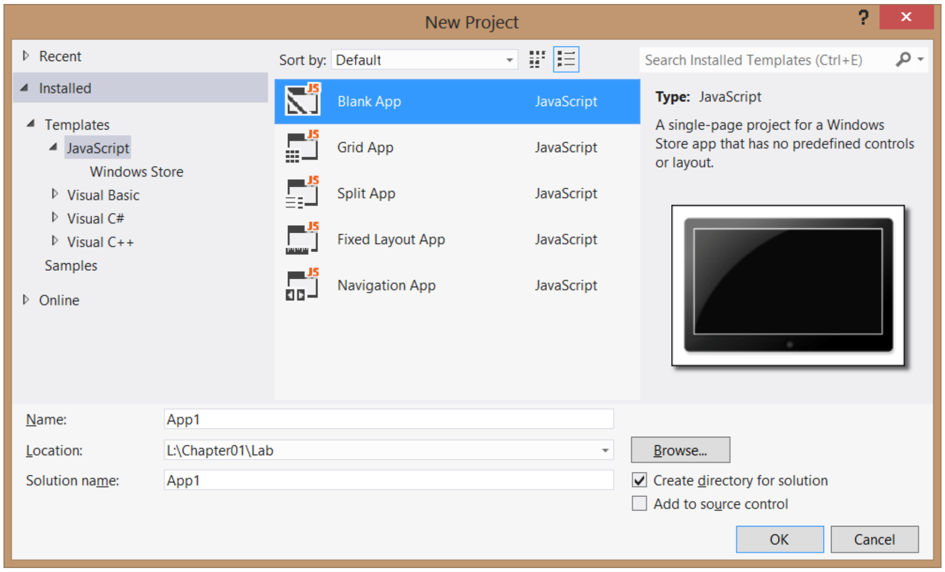

Start Page - Microsoft Visual Studio Express 2012 for Windows 8
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS STORE TEST WINDOW HELP
Start
Start Page
Toolbox
Solution Explorer
Express 2012 for Windows 8
Start
New Project...
Open Project...
Connect to Team Foundation Server...
Recent
App1
Close page after project load
Show page on startup
GET STARTED
LATEST NEWS
Windows 8
What's New in Windows Development
Windows Store apps
Get a developer license for Windows Store apps
Creating Windows Store Apps
Learn to build Windows Store apps
Design Windows Store apps using Blend
Browse Windows Store app samples
Learning Resources
End to end apps and games
API reference for Windows Store apps
Community resources
Sell Your Windows Store Apps
Selling apps
Becoming a Windows Store developer
Preparing your app for the Windows Store
Ready

FIGURE 1-1 Visual Studio Express 2012 for Windows 8 Start page

In addition to clicking the New Project link on the Start page, you can start a new project by clicking File and choosing New Project. Figure 1-2 shows the New Project screen, which displays a list of starting templates from which you can choose to start on your new application quickly. You can think of a template as a project on which someone completed the mundane tasks that would be common to all applications of that type and then saved as a framework that you can use to get started.

On the left side of the New Project screen, you can select from recent project templates that you've used or from the complete list of installed templates, or you can go online to select a template. You'll find that the installed templates are divided according to programming language. Figure 1-2 shows the templates that are installed for JavaScript.



Type: JavaScript

A single-page project for a Windows Store app that has no predefined controls or layout.

Name: App1

Location: L:\Chapter01\Lab

Solution name: App1

[✓] Create directory for solution

[ ] Add to source control

FIGURE 1-2 The New Project screen with the JavaScript project templates

NOTE **INCLUDED TEMPLATES**

In Visual Studio Express 2012 for Windows 8, all the included templates are for creating Windows 8 Windows Store applications. In Visual Studio Express 2012 for Web, all included templates are for creating web applications. Remember that you can use HTML5, CSS3, and JavaScript in Windows Store applications and web applications.

The following are short descriptions of each template.

- **Blank App** This template isn't quite empty. It provides a single-page project for a Windows app, but it has no predefined controls on the page.
- **Grid App** This template provides an application that contains multiple pages and enables you to navigate among groups of items.
- **Split App** This template creates a two-page project in which the first page contains a group of items, and the second page contains an item list with details of the selected item to the right of the list.

- **Fixed Layout App** This template provides a single page with a fixed aspect ratio layout that scales to fit the screen.
- **Navigation App** This template provides a single-page application containing controls for navigation.

Selecting a template causes a copy of the template to be opened in the Solution Explorer window. By default, the Solution Explorer window is on the right side, although windows can be dragged to any location on the screen. Solution Explorer contains a tree representation of all projects that are loaded into the current solution.

Under the Solution Explorer window is the Properties window, which is context-sensitive and contains the properties of the currently selected item. The properties are visible in this window, and most are also configurable.

On the left side of the screen is the toolbox. By default, the toolbox is a tab that you can point to to open the window. The toolbox is also context-sensitive, so different tools are available based on what is being displayed in the center window.

Quick check

- **You want to create a Windows Store application. Which edition of Visual Studio 2012 will you use, and is there an operating system requirement for your system to develop Windows Store application applications?**

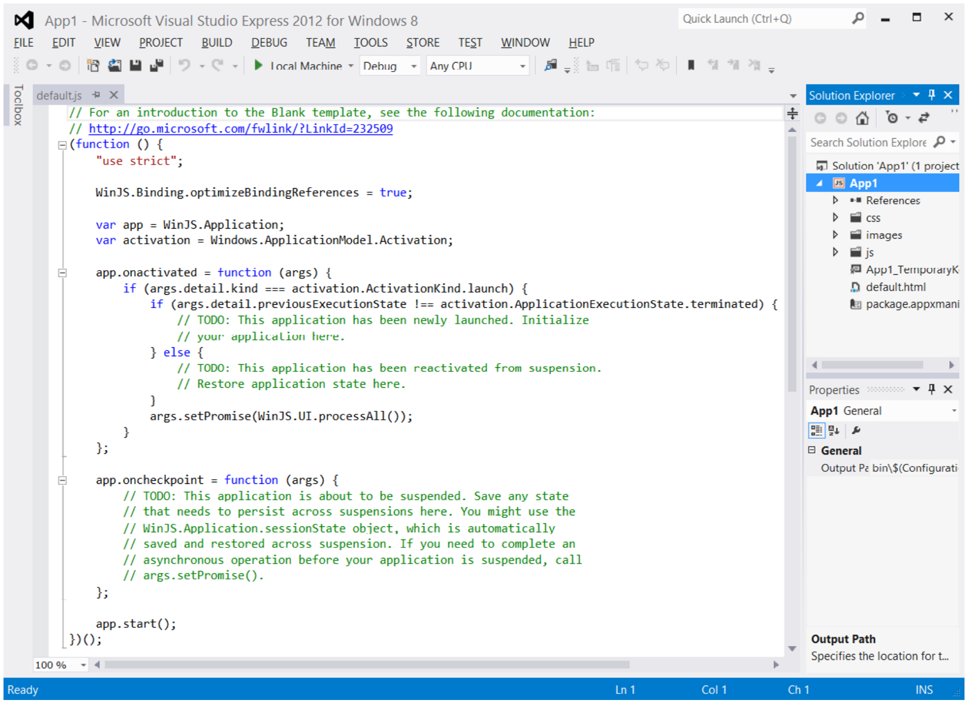
Quick check answer

- **You must use Visual Studio Express 2012 for Windows 8 and have Windows 8 installed to develop Windows Store applications.**

Examining the Blank App template

When the Blank App template is selected, a new solution containing one project is created. The new project won't be totally blank. As shown in Figure 1-3, there are several files and folders in this new project. At the outset, default.js was created, and it's currently displayed in the JavaScript editor window.

The default.js file is in the js folder, which is where you can add your own JavaScript files. This default.js file currently contains a small amount of code, which Chapter 3, "Getting started with JavaScript," revisits in more detail. Here is a general overview of what it does. The use of function on the third line creates a naming scope for variables and functions in your application. In the middle of the code are TODO comments that provide a place to put your own code to be executed when the application is launched or reactivated after being suspended or when the application is being suspended.



```
// For an introduction to the Blank template, see the following documentation:
// http://go.microsoft.com/fwlink/?LinkId=232509
(function () {
    "use strict";

    WinJS.Binding.optimizeBindingReferences = true;

    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;

    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.
            } else {
                // TODO: This application has been reactivated from suspension.
                // Restore application state here.
            }
            args.setPromise(WinJS.UI.processAll());
        }
    };

    app.oncheckpoint = function (args) {
        // TODO: This application is about to be suspended. Save any state
        // that needs to persist across suspensions here. You might use the
        // WinJS.Application.sessionState object, which is automatically
        // saved and restored across suspension. If you need to complete an
        // asynchronous operation before your application is suspended, call
        // args.setPromise().
    };

    app.start();
})();
```

FIGURE 1-3 Blank App template with preliminary coding

Blank App also contains other files that you will want to explore. If you open the default.html file, you'll see the following HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>App1</title>

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

    <!-- App1 references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>
</head>
<body>
    <p>Content goes here</p>
</body>
</html>
```

The first line contains <!DOCTYPE html>, which is a declaration to the web browser that describes the version of HTML in which the page is written. It's not an HTML element; it's a special instruction. In HTML5, this special instruction must be the first thing the browser reads on the page. This instruction is not mandatory, but it is considered a best practice to have it.

Next is the *<html>* element that consists of the starting tag at the top and ending tag, </html>, at the bottom. This is considered the root element of the page, and all other elements are contained within the html element.

Inside the html element are the head and body elements. The head element typically contains metadata and page-related instructions. The body element contains content that will be displayed to the user. In this example, the head element contains a meta element that describes the character set being used (utf-8), a title that will be displayed in the browser title bar, links tags that reference CSS files, and script tags that reference JavaScript files. These references are instructions to the browser to load these files when the page loads. The body element contains a paragraph element with the message "Content goes here." This message appears when the application is executed.

The References folder contains a folder called Windows Library for JavaScript 1.0, which contains subdirectories that Microsoft provides and maintains. You should not modify files in this directory structure, but you should explore the files in this folder structure and learn as much as possible about these files. Of particular importance is the css folder that contains the ui-dark.css and ui-light.css files. These files set the primary theme for your application to either a light or dark theme.

MORE INFO **LIGHT AND DARK BACKGROUNDS**

The default.html file has a reference to the ui-dark.css file. If you run the application, the application displays the "Content goes here" message on a screen that has a dark background. If you change the reference to the ui-light.css file, you'll see a light background.

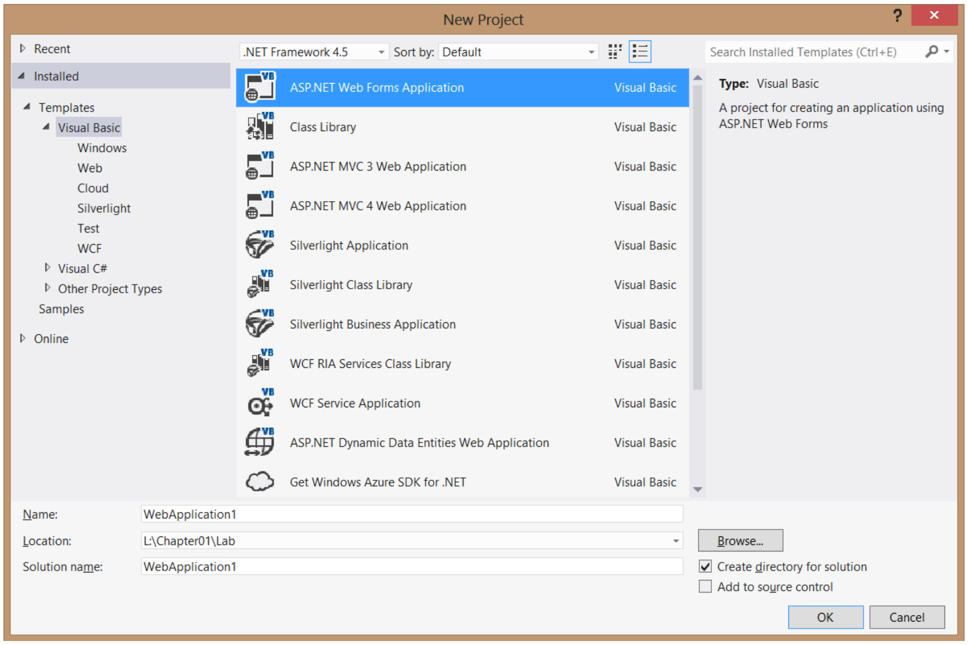
The css folder contains cascading style sheets for your application. Currently, the default.html file references a single file called default.css. The CSS file contains instructions for presenting your HTML file and will be covered in more detail in Chapter 4, "Getting started with CSS3."

The images folder contains blank images that are set to the best size for presentation to the user. You would typically edit these files as part of your finished application.

In the root directory of your application is a file with a .pfx extension that provides a security key for deployment and an appmanifest file that contains deployment metadata.

Exploring Visual Studio Express 2012 for Web

When you start Visual Studio Express 2012 for Web, the same Start Page screen is displayed as shown in Figure 1-1 and described in the previous section. In the installed templates, you'll find that they are divided according to .NET programming language, Visual Basic and Visual C#. Figure 1-4 shows the templates that are installed for Visual Basic.


New Project

ASP.NET Web Forms Application	Visual Basic
Class Library	Visual Basic
ASP.NET MVC 3 Web Application	Visual Basic
ASP.NET MVC 4 Web Application	Visual Basic
Silverlight Application	Visual Basic
Silverlight Class Library	Visual Basic
Silverlight Business Application	Visual Basic
WCF RIA Services Class Library	Visual Basic
WCF Service Application	Visual Basic
ASP.NET Dynamic Data Entities Web Application	Visual Basic
Get Windows Azure SDK for .NET	Visual Basic

.NET Framework 4.5 — Sort by: Default

Type: Visual Basic

A project for creating an application using ASP.NET Web Forms

Name: WebApplication1

Location: L:\Chapter01\Lab

Solution name: WebApplication1

[✓] Create directory for solution

[ ] Add to source control

FIGURE 1-4 The New Project screen with Visual Basic and Visual C# project templates

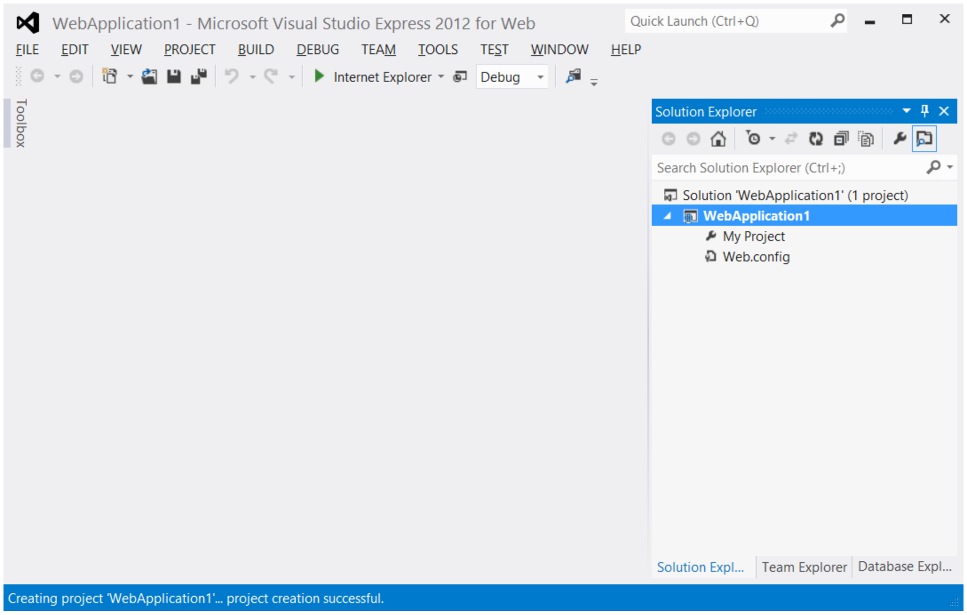
Your new project might differ based on the software installed on your computer. For example, the template shown here is the Get Windows Azure SDK For .NET template that was installed when the Azure SDK was installed.

All these templates are for web-related applications; none of them can be used to create a Windows 8 application. Note that none of the templates support the use of JavaScript as a server-side language, but you can select a Visual Basic or C# web project template and use client-side (on the browser) JavaScript. Remember that you can use HTML5, CSS3, and JavaScript as client-side technologies with any of the web application templates.

Under one of the languages, you can click the Web node to see a list of available web application templates. You can select a template called ASP.NET Empty Web Application to begin with a nearly empty startup project.

Examining ASP.NET Empty Web Application

After selecting ASP.NET Empty Web Application, a copy of the template is opened in the Solution Explorer window on the upper right, as shown in Figure 1-5. This window contains a node for the project (WebApplication1); a node that references the project settings, called My Project; and a node that references the project's configuration file, called Web.config. This project is almost empty. If you press F5 to build and run the web application, it won't run. You must add a webpage to the project first.


WebApplication1 - Microsoft Visual Studio Express 2012 for Web
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Internet Explorer
Debug
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebApplication1' (1 project)
WebApplication1
My Project
Web.config
Solution Expl...
Team Explorer
Database Expl...

Creating project 'WebApplication1'... project creation successful.

FIGURE 1-5 Almost empty ASP.NET Empty Web Application

By default, the Solution Explorer window is on the right side. Under the Solution Explorer window is the Properties window. The Properties window is context-sensitive and contains the properties of the currently selected item. The properties are visible in this window, and most are also configurable.

On the left side of the screen is the toolbox. By default, the toolbox is a tab that you can point to to open the window. The toolbox is also context-sensitive, so different tools are available based on what is being displayed in the center window.

You can add a webpage to the project by right-clicking the project node (WebApplication1) and then navigating to Add | New Item | HTML Page. If you name the page **default.html**, the web server automatically delivers the page to a browser that navigates to your website but doesn't specify the name of a webpage to retrieve. After adding the webpage, you can enter some text, such as a Hello World message, into the body of the webpage. When you press F5, you see the message in the browser.

Lesson summary

- The free editions of Visual Studio 2012 are the Express editions: Visual Studio Express 2012 for Web and Visual Studio Express 2012 for Windows 8. You can use the Express editions to work with HTML5, CSS3, and JavaScript.
- Use Visual Studio Express 2012 for Web to develop web applications. Use Visual Studio Express 2012 for Windows 8 to develop Windows 8 applications.

- Visual Studio Express 2012 for Windows 8 comes with Blend for Visual Studio 2012.
- Blend for Visual Studio 2012 drives the user interface design and must be run on Windows 8 to develop Windows 8 applications.
- You can change the style sheet reference from a dark theme to a light theme by changing the ui-dark.css reference in the default.html file to ui-light.css.

Lesson review

Answer the following questions to test your knowledge of the information in this lesson. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You would like to create a web application by using HTML5, JavaScript, and CSS3. Which of the following Visual Studio 2012 editions can you use? (Choose all that apply.)
   - **A.** Visual Studio Professional 2012
   - **B.** Visual Studio Premium 2012
   - **C.** Visual Studio Ultimate 2012
   - **D.** Visual Studio Express 2012 for Web
   - **E.** Visual Studio Express 2012 for Windows 8
2. You would like to create a Windows 8 application by using HTML5, JavaScript, and CSS3. Which of the following Visual Studio 2012 editions can you use? (Choose all that apply.)
   - **A.** Visual Studio Professional 2012
   - **B.** Visual Studio Premium 2012
   - **C.** Visual Studio Ultimate 2012
   - **D.** Visual Studio Express 2012 for Web
   - **E.** Visual Studio Express 2012 for Windows 8
3. You would like to create web applications and Windows 8 Windows Store applications by using HTML5, JavaScript, and CSS3, but while you're learning, you don't want to buy Visual Studio 2012. Which of the following Visual Studio 2012 editions can you use for free to accomplish your goal?
   - **A.** Visual Studio Professional 2012
   - **B.** Visual Studio Premium 2012
   - **C.** Visual Studio Ultimate 2012
   - **D.** Visual Studio Express 2012 for Web and Visual Studio Express 2012 for Windows 8

Lesson 2: Blend for Visual Studio 2012

Blend is included with Visual Studio 2012 Express for Windows 8 and helps you design your user interface. Blend is a design complement for Visual Studio and does for design what Visual Studio does for code. The following are some key features of Blend.

- **Visual design** Edit HTML, CSS, and Windows Store controls in a "what you see is what you get" (WYSIWYG) environment. What you see in Blend is what users will see in Windows 8.
- **Interactive mode** Design your app by changing states and setting styles. You don't need to compile and run continuously. Blend provides the ability to use interactive mode so the developer can run the application on the design surface until the desired state is reached. The developer can pause the application and then style the application for the new state.
- **App building** Windows Store controls can be dragged and dropped onto the design surface. After that, just set the properties and styles.
- **Powerful code generation** Blend takes care of all the syntax by generating concise, reliable, predictable code when you add a style or element to your application.
- **Debugging** Blend offers visual debugging of HTML and CSS. It has a virtual rule called Winning Properties that shows you how an element obtained its effective style from the CSS inheritance and cascade.

After this lesson, you will be able to:

- Identify the key features of Blend.
- Start a project by using Blend.

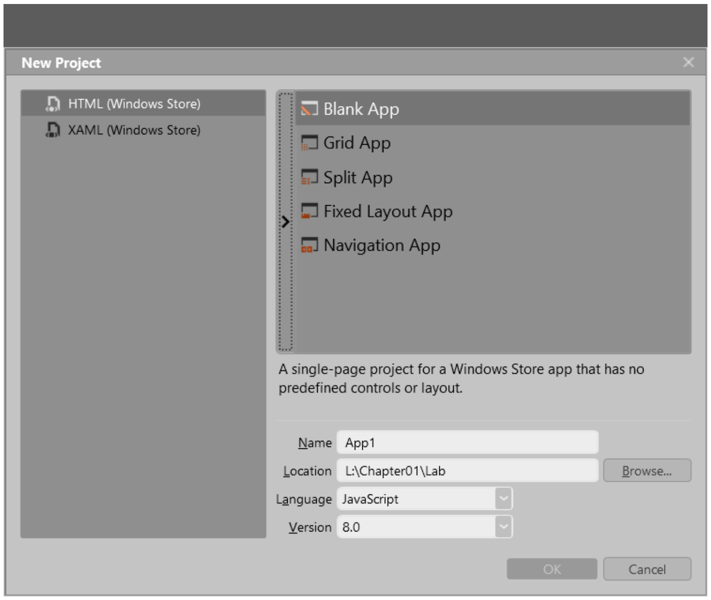
Estimated lesson time: 25 minutes

Exploring Blend

Blend is an exciting tool for designers and developers who will be using HTML5, CSS3, and JavaScript to develop Windows 8 applications. Blend also supports the creation of Windows 8 Windows Store applications by using XAML with your choice of .NET programming language. Figure 1-6 shows the New Project screen, which has the same new project templates as Visual Studio Express 2012 for Windows 8.

NOTE **RUN CONSTRAINT FOR BLEND**

Blend for Visual Studio 2012 must be run on Windows 8 to develop Windows 8 applications. If you run Blend for Visual Studio 2012 on Windows 7, you will not be able to develop Windows 8 applications.



A single-page project for a Windows Store app that has no predefined controls or layout.

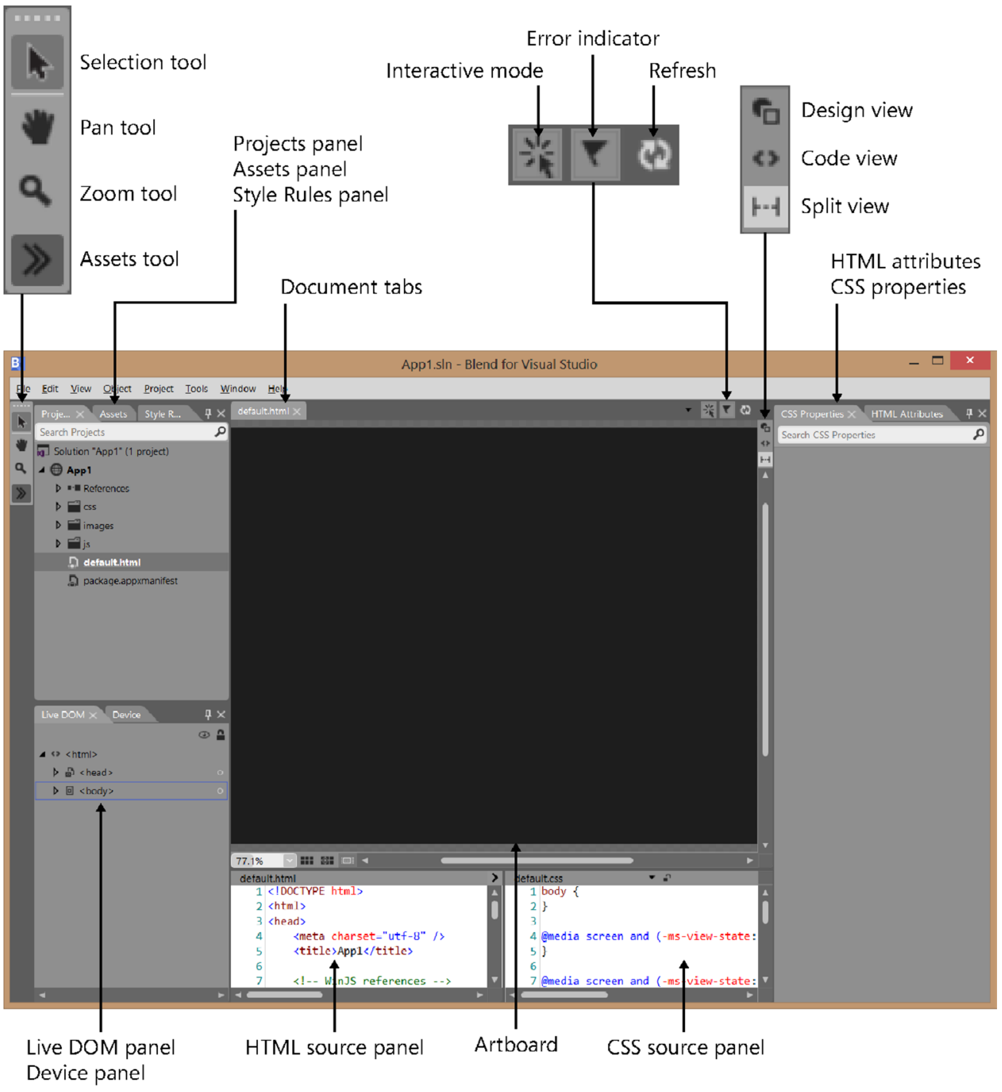
Name: App1
Location: L:\Chapter01\Lab
Language: JavaScript
Version: 8.0

FIGURE 1-6 The Blend New Project screen with the JavaScript project templates

Selecting Blank App creates the same Blank App that was discussed in the previous section. Note the screen layout. Figure 1-7 shows the Blend screen layout. On the left is the Tools panel, where you can point to each icon to see a tooltip that displays the name of the command. Just to the right of the Tools panel is a column with two windows, one over the other. These windows have tabs that can be selected to show more information. The upper-left window contains the following tabs.

- **Projects** Contains a tree-based representation of your solution, which can contain many projects, each project containing resources such as files and folders.
- **Assets** Contains a library of resources such as HTML elements, JavaScript controls, and media that you will use within your application.
- **Style Rules** Contains a list of cascading style sheets that are referenced in your project.

Under Style Rules is the Live DOM window, which shows a dynamic representation of your HTML page.



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>App1</title>

    <!-- WinJS references -->
```

```
body {
}

@media screen and (-ms-view-state:
}

@media screen and (-ms-view-state:
```

FIGURE 1-7 The Blend screen layout

In the middle of the screen is your primary workspace, the *artboard*. At the top of the artboard is a tabbed list of documents that are open. By default, this window displays the rendered page. Note that there are buttons in the upper-right corner that can be used to change the view.

The bottom center displays the default.html and the default.css sources. This makes it easy for you to change the files and see the rendered output. Also, as you use the other windows to modify the rendered page, you see the changes reflected in these files.

The rightmost window contains the following two tabs.

- **HTML Attributes** Displays the properties for the currently selected HTML element. You can view or change these settings.
- **CSS Properties** Displays the style settings for the currently selected HTML element. You can set these properties.

Quick check

- **You want to be able to design your app by changing states and setting styles. Which mode provides this feature?**

Quick check answer

- **Interactive mode. You can run the application on the design surface until the desired state is reached. You can pause the application and then style the application for the new state.**

Projects panel

The Projects panel provides a file and folder view of the projects in your solution, as shown in Figure 1-8. You can use this panel to open files for editing by double-clicking the file. You can also right-click any file or folder to see options such as Copy, Delete, and Rename.

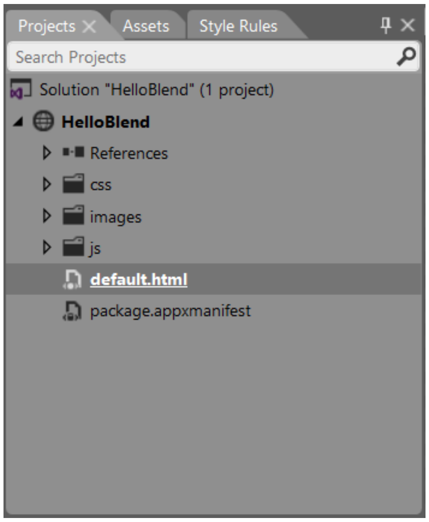


FIGURE 1-8 The Projects panel

Notice the different icons for the solution, project, references, folders, and files and note that the default.html file is underlined to indicate that it is the startup file when you run the application. At the top of the Projects panel is a Search Projects text box in which you can type the name of a file or folder you want to find. In the project is a virtual folder called

References. This is where you add references to CSS and JavaScript. The project also contains the package.manifest file, which contains all the settings for the project, including the setting for the Start page.

Assets panel

The Assets panel lists all the HTML elements, controls, and media that you can add to an HTML page that is open in the artboard, as shown in Figure 1-9. Although the Assets panel lists all the available controls in your Blend project, the most recently used controls appear in the Tools panel.

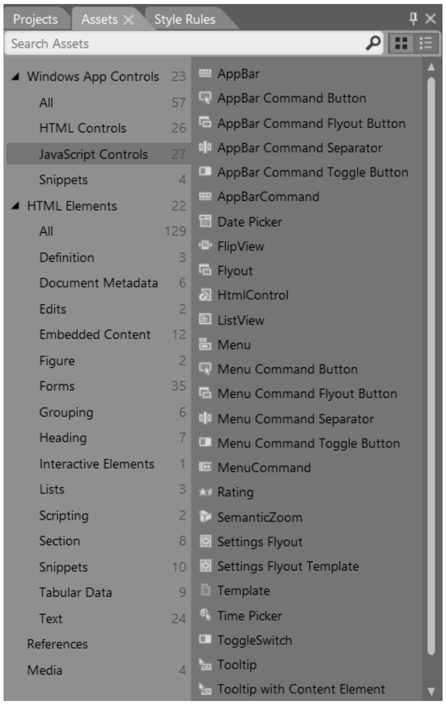


FIGURE 1-9 The Assets panel shown when building a Windows Store application with HTML

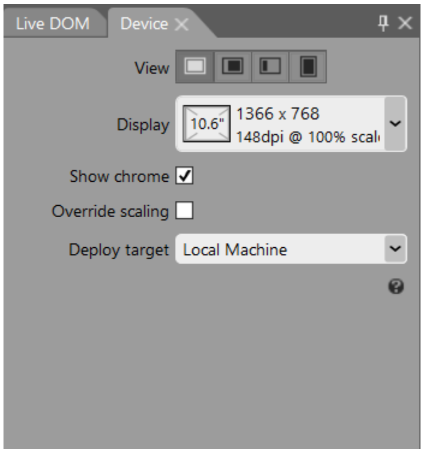
Open the Assets panel either by clicking the Assets icon at the bottom of the Tools panel or by clicking Assets in the Windows menu.

Device panel

Use the Device panel to configure your display so that you can visualize your application accurately on a variety of displays, as shown in Figure 1-10. You can select the following display options for your application.

- **View** The rendering mode when the application is run. Choices are landscape, filled, snapped, and portrait.

- **Display** The display size and resolution at which to render. Use this to simulate rendering on larger or smaller screens to see whether your application renders properly.
- **Show chrome** When selected, shows a simulated tablet screen around the edge of the application.
- **Override scaling** When selected, emulates the built-in display scaling of the device.
- **Deploy target** The device to which to deploy when the application is run.

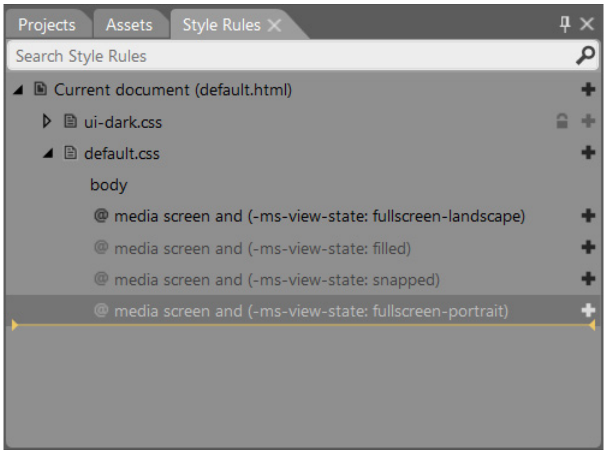


Display 10.6" 1366 x 768 148dpi @ 100% scal
Show chrome ✓
Override scaling ☐
Deploy target Local Machine

FIGURE 1-10 The Device panel

Style Rules panel

The Style Rules panel, shown in Figure 1-11, lists all the style sheets attached to the current document. Within each style sheet is a list of all the defined styles. In addition, the Style Rules panel contains a text box in which you can enter search criteria when locating a style.



Current document (default.html)
ui-dark.css
default.css
body
@ media screen and (-ms-view-state: fullscreen-landscape)
@ media screen and (-ms-view-state: filled)
@ media screen and (-ms-view-state: snapped)
@ media screen and (-ms-view-state: fullscreen-portrait)

FIGURE 1-11 The Style Rules panel containing a list of attached style sheets

You can click the plus signs on the right side of the panel to add a new style rule at that location. The yellow line indicates where new styles will be created if a location is not specified. Style rules that are dimmed are defined but not used in the current document.

Live DOM panel

The Live DOM panel displays the structure of the current document as a hierarchical representation, as shown in Figure 1-12. With the Live DOM panel, you can select elements and adjust their style rules. The Live DOM view provides automatic updating as the state of the app changes.

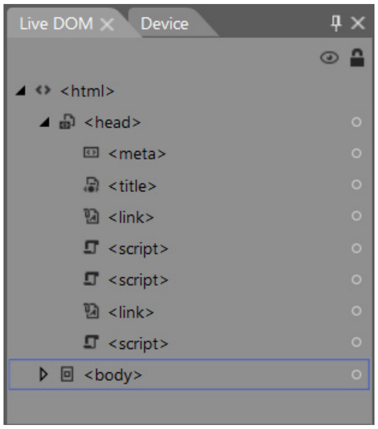


FIGURE 1-12 The Live DOM panel displaying a hierarchical representation of the current document

The Live DOM panel displays nodes by using their ID if an ID is assigned or by using the tag name if no ID is assigned. You can control the visibility of any node by clicking the eye icon on the right side of the panel. This can be helpful when you have layers of elements stacked on top of each other.

As with Visual Studio, you can press F5 to run the application. Blend has many features that you can learn about by using the built-in help.

Lesson summary

- The interactive mode enables you to run the application to build to the desired state and then pause the application and set the styles of the application based on the current state.
- The Assets panel contains a list of all available assets in the project.
- The Projects panel contains a file and folder view of the projects in the current solution.
- The Style Rules panel contains a list of all style sheets attached to the current document.

- The Device panel enables you to run the application by using simulations of different screen sizes and orientations.
- The Live DOM panel enables you to select an element and apply style rules to it.

Lesson review

Answer the following questions to test your knowledge of the information in this lesson. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You would like to create a Windows 8 application by using Blend and HTML5, JavaScript, and CSS3. Which feature of Blend enables you to pause an application when it reaches a desired state so you can set the style rules for the page and its controls while in this state? (Choose all that apply.)
   - **A.** Assets panel
   - **B.** Projects panel
   - **C.** Visual Design
   - **D.** Interactive mode
2. On which panel can you see a hierarchically structured view of the DOM?
   - **A.** Live DOM
   - **B.** Projects
   - **C.** Assets
   - **D.** Device
3. Which panel can you use to access a list of the HTML elements, controls, and media that can be added to an HTML page that is open in the artboard?
   - **A.** Projects
   - **B.** Assets
   - **C.** Device
   - **D.** Live DOM

Practice exercises

If you encounter a problem completing any of these exercises, the completed projects can be installed from the Practice Exercises folder that is provided with the companion content.

Exercise 1: Hello World with Visual Studio Express 2012 for Windows 8

In this exercise, you create a simple HTML5 and JavaScript Hello World application by using Visual Studio Express 2012 for Windows 8. This practice, like all Hello World applications, is intended to get you started by creating a minimal application. In later exercises, you get more exposure to Visual Studio. In this exercise, you use HTML5 to display "Hello Visual Studio for Windows 8" on the screen.

1. If you haven't installed Visual Studio Express 2012 for Windows 8, do so now. You can download this from the Microsoft website.
2. Start Visual Studio. Click File and choose New Project to display the New Project dialog box. Navigate to Installed | Templates | JavaScript | Windows Store. Select Blank App.
3. Set the name of your application to **HelloVisualStudioForWin8**.
4. Select a location for your application.
5. Set the solution name to **HelloVisualStudioForWin8Solution**.
6. Be sure to keep the Create Directory For Solution check box selected.
7. Click OK to create your new project.
8. When the application is created, the default.js file is displayed with some template code.

   This code is covered later, and there is no need to alter it now.
9. Open the default.html file.

   It contains HTML from the template.
10. To see the default content, press F5 to start debugging the application.

    You should see a black screen and, in the upper-left corner, a message stating, "Content goes here."
11. Switch back to Visual Studio by pressing Alt+Tab.

    The title bar says (Running).
12. Press Shift+F5 to stop debugging.

    Shift+F5 works only when Visual Studio has the focus; it does not work when the running application has the focus.
13. In the default.html file, replace the "Content goes here" message with **Hello Visual Studio for Windows 8**.
14. Replace the reference to ui-dark.css with ui-light.css.

    Your HTML should look like the following.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>HelloWorldForWin8</title>
```

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

        <!-- HelloWorld references -->
        <link href="/css/default.css" rel="stylesheet" />
        <script src="/js/default.js"></script>
    </head>
    <body>
        <p>Hello Visual Studio for Windows 8</p>
    </body>
    </html>
```

15. Press F5 to start debugging.

    The screen is white because you now reference the ui-light.css file instead of the ui-dark.css file. The screen also displays Hello Visual Studio For Windows 8. Congratulations—you have written your first Windows 8 application by using HTML5 technologies!

Exercise 2: Hello World with Visual Studio Express 2012 for Web

In this exercise, you create a simple HTML5 and JavaScript Hello World application by using Visual Studio Express 2012 for Web. This practice, like all Hello World applications, is intended to get you started by creating a minimal application. In later exercises, you get more exposure to Visual Studio. In this exercise, you create a new project in Visual Studio Express 2012 for Web and use HTML5 to display “Hello Visual Studio for Web” on the screen.

NOTE **NO SERVER CODE IN THIS EXERCISE**

You will not be writing any server code in this exercise, so it doesn’t matter whether you select Visual Basic or Visual C# when starting the new project.

1. If you haven’t installed Visual Studio Express 2012 for Web, do so now. You can download this from the Microsoft website.
2. Start Visual Studio. Click File and choose New Project to display the New Project dialog box. Navigate to Installed | Templates | Visual Basic | Web. Select the ASP.NET Web Form Application.
3. Set the name of your application to **HelloVisualStudioForWeb**.
4. Select a location for your application.
5. Set the solution name to **HelloVisualStudioForWeb Solution**.
6. Be sure to keep the Create Directory For Solution check box selected.
7. Click OK to create your new project.

8. When the application is created, the default.aspx page will be displayed with some template code.
9. In the Solution Explorer window, build the project by right-clicking the project node and choosing Build.
10. To see this template's default content, press F5 to start debugging the application.

    You should see a fancy screen with information on how to get started plus other useful information.
11. Switch back to Visual Studio by pressing Alt+Tab.

    The title bar says (Running).
12. Press Shift+F5 to stop debugging.

    Note that Shift+F5 works only when Visual Studio has the focus. Shift+F5 does not work when the running application has the focus.
13. Delete the default.aspx file by right-clicking this file in the Solution Explorer window, choosing Delete, and then clicking OK.
14. In the Solution Explorer window, add a default.html file by right-clicking the project node. Click Add and then choose HTML. Name the page **default.html**.
15. In the default.html file, place the text **Hello Visual Studio for Web** between the *<body>* and *</body>* tags.
16. In the default.html file, place the text **HelloVisualStudioForWeb** between the *<title>* and *</title>* tags.

    Your HTML should look like the following.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>HelloVisualStudioForWeb</title>
</head>
<body>
   Hello Visual Studio for Web
</body>
</html>
```

17. In the Solution Explorer window, set the default.html file as the startup file by right-clicking the default.html file and choosing Set As Start Page.
18. Press F5 to start debugging.

    The screen now displays Hello Visual Studio For Web. Congratulations—you have written your first web application using HTML5 technologies!

Exercise 3: Hello World with Blend

In this exercise, you create a simple HTML5 and JavaScript Hello World application by using Blend. This practice, like all Hello World applications, is intended to get you started by creating a minimal application. In later exercises, you get more exposure to Blend. In this exercise, you create a new project in Blend and use HTML5 to display “Hello World” on the screen.

1. If you haven’t installed Blend, do so now. Remember that Blend is installed automatically when you install Visual Studio Express 2012 for Windows 8. You can download this from the Microsoft website.
2. To start Blend, click New Project to display the New Project dialog box. Select the HTML (Windows Store) category in the left pane and select Blank App in the right pane.
3. Set the name of your application to **HelloBlend**.
4. Select a location for your application.
5. Click OK to create your new project.

   When the application is created, the default.html file is displayed.
6. To see the default content, press F5 to start debugging the application.

   You should see a black screen and, in the upper-left corner, a message stating, “Content goes here.”
7. Close the running application by pressing Alt+F4.
8. If Blend is not displayed, return to Blend by pressing Alt+Tab.
9. In the default.html file, double-click the “Content Goes Here” message and replace the text with **Hello from Blend**.

   You see the change in the default.html source view window at the bottom of the screen.
10. Replace the reference to ui-dark.css with ui-light.css.

    Your HTML should look like the following.

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

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

    <!-- HelloBlend references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>
</head>
<body>
```

```
    <p>Hello from Blend</p>
</body>
</html>
```

11. Press F5 to start the application.

    Notice that the screen is white because you now reference the ui-light.css file instead of the ui-dark.css file. The screen now displays Hello From Blend. Congratulations—you have written a Windows 8 application by using HTML5 technologies with Blend!

Suggested practice exercises

The following additional exercises are designed to give you more opportunities to practice what you've learned and to help you successfully master the lessons presented in this chapter.

- **Exercise 1** Learn more about Visual Studio Express 2012 for Web by creating new projects from each of the included project templates. After creating each project, try adding Hello World and run the application to see how the project looks and behaves.
- **Exercise 2** Learn more about Visual Studio Express 2012 for Windows 8 by creating new projects from each of the included project templates. After creating each project, try adding Hello World and run the application to see how the project looks and behaves.
- **Exercise 3** Learn more about Blend by creating new projects from each of the included project templates. After creating each project, try adding Hello World and run the application to see how the project looks and behaves.

Answers

This section contains the answers to the lesson review questions in this chapter.

Lesson 1

1. **Correct answers: A, B, C, and D**
   - **A. Correct:** Visual Studio Professional 2012 provides web templates for creating web applications.
   - **B. Correct:** Visual Studio Premium 2012 provides web templates for creating web applications.
   - **C. Correct:** Visual Studio Ultimate 2012 provides web templates for creating web applications.
   - **D. Correct:** Visual Studio Express 2012 for Web provides web templates for creating web applications only.
   - **E. Incorrect:** Visual Studio Express 2012 for Windows 8 provides templates for building Windows 8 applications only.
2. **Correct answers: A, B, C, and E**
   - **A. Correct:** Visual Studio Professional 2012 provides web templates for creating Windows 8 applications.
   - **B. Correct:** Visual Studio Premium 2012 provides web templates for creating Windows 8 applications.
   - **C. Correct:** Visual Studio Ultimate 2012 provides web templates for creating Windows 8 applications.
   - **D. Incorrect:** Visual Studio Express 2012 for Web provides web templates for creating web applications only.
   - **E. Correct:** Visual Studio Express 2012 for Windows 8 provides templates for building Windows 8 applications only.
3. **Correct answer: D**
   - **A. Incorrect:** Visual Studio Professional 2012 enables you to create web and Windows 8 applications, but it is not free.
   - **B. Incorrect:** Visual Studio Premium 2012 enables you to create web and Windows 8 applications, but it is not free.
   - **C. Incorrect:** Visual Studio Ultimate 2012 enables you to create web and Windows 8 applications, but it is not free.
   - **D. Correct:** Visual Studio Express 2012 for Web provides web templates for creating web applications, and Visual Studio Express 2012 for Windows 8 provides templates for creating Windows 8 applications. Both are free.

Lesson 2

1. **Correct answers: C and D**
   - **A. Incorrect:** The Assets panel enables you to access a list of the HTML elements, controls, and media that can be added to an HTML page that is open in the artboard.
   - **B. Incorrect:** The Projects panel provides a file and folder view of the projects in the current solution.
   - **C. Correct:** With Visual Design, what you see in Blend is what users will see in Windows 8.
   - **D. Correct:** Interactive mode enables you to run the application on the design surface until the desired state is reached. You can pause the application and then style the application for the new state.
2. **Correct answer: A**
   - **A. Correct:** Live DOM displays the structure of the current document as a hierarchical representation. You can use the Live DOM panel to select elements and adjust their style rules.
   - **B. Incorrect:** The Projects panel provides a file and folder view of the projects in your solution.
   - **C. Incorrect:** The Assets panel lists all the HTML elements, controls, and media that you can add to an HTML page that is open in the artboard.
   - **D. Incorrect:** The Device panel enables you to configure your display so that you can visualize your application accurately on a variety of displays.
3. **Correct answer: B**
   - **A. Incorrect:** The Projects panel provides a file and folder view of the projects in your solution.
   - **B. Correct:** The Assets panel lists all the HTML elements, controls, and media that you can add to an HTML page that is open in the artboard.
   - **C. Incorrect:** The Device panel enables you to configure your display so that you can visualize your application accurately on a variety of displays.
   - **D. Incorrect:** The Live DOM panel displays the structure of the current document as a hierarchical representation. You can use the Live DOM panel to select elements and adjust their style rules.

CHAPTER 2

Getting started with HTML5

Welcome to the world of HTML5, JavaScript, and CSS3! This chapter gets you started with HTML5. The next chapter does the same with JavaScript, and the following chapter familiarizes you with CSS3.

Now that you've installed Visual Studio 2012 and Blend, you're ready to build your knowledge foundation by learning some basic HTML. This chapter presents a great deal of HTML content. Although much of the content in this chapter exists in previous releases of HTML, all content in this chapter is part of the HTML5 specification.

Lessons in this chapter:

- Lesson 1: Introducing HTML5 **29**
- Lesson 2: Embedding content **44**

Before you begin

To complete this book, you must have some understanding of web development. This chapter requires the hardware and software listed in the "System requirements" section in the book's Introduction.

Lesson 1: Introducing HTML5

Chapter 1, "Getting Started with Visual Studio 2012 and Blend for Visual Studio 2012," presented a very brief overview of the Visual Studio editions and Blend. This was necessary to introduce you to the tools that will be used in this book. This lesson presents a rather detailed overview of HTML5 and covers many of the fundamentals of HTML that existed prior to HTML5 but are still part of the HTML5 specification.

After this lesson, you will be able to:

- Understand the history of HTML5.
- Create an HTML5 document and add elements and attributes to it.
- Add comments to an HTML5 document.
- Use special characters in your HTML document.

Estimated lesson time: 30 minutes

Understanding HTML, XHTML, and HTML5

HTML is an acronym for Hypertext Markup Language, which is the language we have used to create webpages since the first webpages arrived on the web. HTML roots are from an older markup language that was used in document publishing, called SGML (Standard Generalized Markup Language). Like SGML, HTML uses *tags* to provide formatting features such as *<b>*this is bold*</b>*, which would cause the text within the starting b tag and ending b tag to render as bolded text. Notice the difference between the first and second tag; the second tag has a slash (/) to indicate that it's an ending tag. Many but not all HTML tags have a matching end tag. HTML tags such as *<br>* and *<img>* did not have ending tags because the *<br>* just rendered a line break, and the *<img>* tag just rendered an image.

One interesting aspect of HTML and its relationship with browsers was that browsers were designed to be backward compatible and forward compatible. Creating a browser that is backward compatible is relatively easy because the problem domain is known, but how is forward compatibility accomplished? Browsers were created to ignore tags that they didn't recognize. For example, if a browser came across a *<xyz>* tag that it didn't recognize, it would skip over the tag as though it didn't exist.

Although HTML served its purpose for quite some time, people wanted more. Another evolving technology, called XML (eXtensible Markup Language), was becoming popular. XML looks a lot like HTML because both languages use tags. However, there are several big differences. First, XML always requires matching end tags for every tag, although you can use a shortcut notation to denote a starting tag and ending tag together. Another difference is that HTML has a very specific set of tag names, and each of these tags denotes a formatting feature that is to be applied to the rendered webpage. XML doesn't have a defined set of tag names, so you create your own tag names, and the tags can represent anything. XML tags are typically metadata tags: tags that describe the data that is within the tag. Although there are many other differences, one other large difference is that XML uses XML Schema Definition (XSD) technology, which validates the format of an XML document to ensure that all aspects of a document are valid before processing the XML document. HTML's lack of rigid structure prevented the creation of a technology such as XSD that could validate HTML documents.

The *World Wide Web Consortium*, also known as *W3C* (*http://ww.w3c.org*), is responsible for developing open standards for the web. The W3C introduced XHTML to solve the problems in HTML, which was up to version 4. *XHTML* is an XML-based specification that tightened the HTML specification to make HTML adhere to the XML rules that describe a well-formed document, such as having a matching end tag for each starting tag. This meant that XHTML documents could be validated by using XSD files and could be edited by using XML tools.

Although XHTML solved some problems, other problems still needed a solution. There was a need for an increasing amount of multimedia on the web. Companies wanted the flashiest (pun intended) website. Cascading Style Sheets (CSS) provided support for adding styles such as colors and fonts consistently across a website, but companies wanted more. They wanted their webpages to be highly interactive, with video and animations. Browsers added programmable support by providing JavaScript, but early versions of JavaScript were slow and difficult to program. The browsers became extensible by providing an application programming interface (API) that would allow third parties to create *plug-ins* that could run in the browser's environment. One such plug-in is Flash, which has a very large installed base. Flash provides a development environment that can be used to create a rich user experience. Although third-party plug-ins solved the immediate need for technology to create flashy websites, there was still a need for tighter integration of multimedia with the browser, especially on small devices.

HTML5 does not originate from XHTML; HTML5 originates from HTML 4.01. As a rule, however, applying XHTML rules to your HTML5 will make your webpage more compliant with a wider variety of browsers and webpage readers, generators, and validators. This book attempts to be most compliant with the most technologies.

HTML5 represents a reinvented HTML, CSS, and JavaScript in a way that solves the need for rich, interactive websites that can play audio and video and support animations from within the browser without the need for plug-ins. HTML5 contains most of the tags from HTML 4.01, but many of the tags have been redefined to be semantic tags.

Introducing semantic markup

HTML5 stresses separating structure, presentation, and behavior, a good practice to adhere to. *Semantic* is defined as the study of meaning of linguistic expressions. In the context of HTML, that means that tags provide meaning to the content in the HTML document. Tags do not provide presentation; they provide meaning.

HTML tags provide a meaningful structure, but do not provide presentation. Remember that separation is accomplished by providing structure in your HTML5 document, maintaining presentation in your CSS3 style sheet, and maintaining behavior in your JavaScript file.

How can you maintain separation when tags such as *<b>* (bold) and *<i>* (italic) exist? After all, these tags have presentation in their definitions. The W3C now defines the *<b>* tag as "a span of text offset from its surrounding content without conveying any extra emphasis or importance, and for which the conventional typographic presentation is bold text; for example, keywords in a document abstract, or product names in a review." The W3C now defines the *<i>* tag as "a span of text offset from its surrounding content without conveying any extra emphasis or importance, and for which the conventional typographic presentation is italic text; for example, a taxonomic designation, a technical term, an idiomatic phrase from another language, a thought, or a ship name." Do these tags need to render as bold and italic? Not at all, and the new definitions of these tags attempt to specify this.

Chapter 5, "More HTML5," revisits the notion of semantic markup. For now, remember that your HTML tags should be used to provide structure, not presentation. Presentation is the cascading style sheet's job.

Working with elements

An *element* is composed of a beginning tag, an ending tag, and the content between the tags. Consider the following HTML fragment.

```
<div>
   The quick brown <b>fox</b> jumps over the lazy dog
</div>
```

In this sample, the *<div>* tag is just the beginning tag on the first line. The *<div>* element is the complete sample, which includes content that also contains a *<b>* element. The *<b>* element consists of the beginning *<b>* tag, the content, which is the word "fox," and the ending *</b>* tag.

The *<div>* element creates a section in your document. It's common to use *<div>* elements to denote a section to which you will attach a style. You'll see many uses of the *<div>* element in this book and on most websites.

HTML tag names are not case sensitive. If you're working on an older webpage, you might notice that it was written using uppercase tag names. Browsers will treat a *<b>* tag and a *<B>* tag the same.

To comply with as many standards as possible, consider using lowercase tag names for any webpages you create by convention because the W3C recommends lowercase tag names in HTML 4.01 and requires lowercase tag names in XHTML. Although HTML5 does not mandate lowercase tag names, lowercase tag names are recommended.

Element reference

HTML5 has more than 100 defined elements that you can use to create rich webpages and applications. The W3C defines the following list of these elements with a brief description. Note that brevity is a substitute for 100 percent accuracy in these descriptions.

- **<a>** Hyperlink
- **<abbr>** Abbreviation
- **<address>** Contact information
- **<area>** Image map region
- **<article>** Independent section
- **<aside>** Auxiliary section
- **<audio>** Audio stream

- **<b>** Bold text
- **<base>** Document base URI
- **<bb>** Browser button
- **<bdo>** Bi-directional text override
- **<blockquote>** Long quotation
- **<body>** Main content
- **<br>** Line break
- **<button>** Push button control
- **<canvas>** Bitmap canvas
- **<caption>** Table caption
- **<cite>** Citation
- **<code>** Code fragment
- **<col>** Table column
- **<colgroup>** Table column group
- **<command>** Command that a user can invoke
- **<datagrid>** Interactive tree, list, or tabular data
- **<datalist>** Predefined control values
- **<dd>** Definition description
- **<del>** Deletion
- **<details>** Additional information
- **<dfn>** Defining instance of a term
- **<dialog>** Conversation
- **<div>** Generic division
- **<dl>** Description list
- **<dt>** Description term
- **<em>** Stress emphasis
- **<embed>** Embedded application
- **<fieldset>** Form control group
- **<figure>** A figure with a caption
- **<footer>** Section footer
- **<form>** Form
- **<h1>** Heading level 1
- **<h2>** Heading level 2
- **<h3>** Heading level 3
- **<h4>** Heading level 4

- **<h5>** Heading level 5
- **<h6>** Heading level 6
- **<head>** Document head
- **<header>** Section header
- **<hr>** Separator
- **<html>** Document root
- **<i>** Italic text
- **<iframe>** Inline frame
- **<img>** Image
- **<input>** Form control
- **<ins>** Insertion
- **<kbd>** User input
- **<label>** Form control label
- **<legend>** Explanatory title or caption
- **<li>** List item
- **<link>** Link to resources
- **<map>** Client-side image map
- **<mark>** Marked or highlighted text
- **<menu>** Command menu
- **<meta>** Metadata
- **<meter>** Scalar measurement
- **<nav>** Navigation
- **<noscript>** Alternative content for no script support
- **<object>** Generic embedded resource
- **<ol>** Ordered list
- **<optgroup>** Option group
- **<option>** Selection choice
- **<output>** Output control
- **<p>** Paragraph
- **<param>** Plug-in parameter
- **<pre>** Preformatted text
- **<progress>** Progress of a task
- **<q>** Inline quotation
- **<rp>** Ruby parenthesis
- **<rt>** Ruby text

- **<ruby>** Ruby annotation
- **<samp>** Sample output
- **<script>** Linked or embedded script
- **<section>** Document section
- **<select>** Selection control
- **<small>** Small print
- **<source>** Media resource
- **<span>** Generic inline container
- **<strong>** Strong importance
- **<style>** Embedded style sheet
- **<sub>** Subscript
- **<sup>** Superscript
- **<table>** Table
- **<tbody>** Table body
- **<td>** Table cell
- **<textarea>** Multiline text control
- **<tfoot>** Table footer
- **<th>** Table header cell
- **<thead>** Table head
- **<time>** Date and/or time
- **<title>** Document title
- **<tr>** Table row
- **<ul>** Unordered list
- **<var>** Variable
- **<video>** Video or movie
- **<wbr>** Optionally break up a large word at this element

Many of these elements are discussed in more detail later in this book.

Adding attributes to elements

The begin tag can contain additional data in the form of an attribute. An *attribute* is a name="value" pair in which name is unique within the tag and value is always enclosed within either single quotes or double quotes. You can add many attributes to the begin tag. You can also alternate using single quotes and double quotes, which is especially beneficial when you need to embed single or double quotes within the value of the attribute. You can also have Boolean attributes that contain the attribute name but no value.

Here is an example of an element that has attributes.

```
<div id="main" class='mainContent'></div>
```

In this example, id and class are attributes. The id attribute uniquely identifies an element within an HTML document. The class attribute specifies a named CSS style that should be applied to the element.

Working with Boolean attributes

Some attributes are *Boolean* attributes, which means that the mere presence of the attribute indicates that an option is set.

Some examples of Boolean attributes are as follows.

- **checked** Used with the check box and option button to indicate selection
- **selected** Used to indicate which option is selected in a drop-down or select list
- **disabled** Used to disable input, text area, button, select, option, or opt group
- **readonly** Used to prevent the user from typing data into a text box, password, or text area

There are different ways to indicate a Boolean attribute. One way is to use the minimized form, by which you just add the attribute name into the starting tag but don't provide a value. Here is an example of minimized form when setting a check box to selected.

```
<input type="checkbox" name="fruit" value="Apple" checked />
```

Another way to indicate a Boolean attribute is to use quoted form, in which you provide either an empty value or the name of the attribute as its value. Here are examples of both.

```
<input type="checkbox" name="fruit" value="Apple" checked='' />
<input type="checkbox" name="fruit" value="Apple" checked='checked' />
```

The latter seems redundant but is usually considered to be the preferred way to represent the Boolean attribute. If you use jQuery, which is a third-party JavaScript toolset, you'll find that it works best with that redundant example.

Quick check

- **You are using a <*button*> element, and you want it to be disabled until some criteria is met. What is the best way to disable the <*button*> element when the page is loaded?**

Quick check answer

- **Write the <*button*> element using quoted syntax and assign the attribute name to the attribute as follows.**

```
<button type='button' id='myButton' disabled='disabled'>Button</button>
```

HTML5 global attribute reference

HTML5 defines a set of named attributes that can be applied to any HTML5 element. These elements are called global attributes, and each has a very specific meaning, as follows.

- **accesskey** Enables you to either specify a shortcut key to which to jump or to set focus to an element. As a rule, you shouldn't use this because it can cause problems with other technologies.
- **class** Used with CSS to specify one or more class names for an element.
- **contenteditable** Specifies that the content within the tag can be edited.
- **contextmenu** User can right-click an element to display a menu. At the time of this writing, no browser supports this attribute.
- **dir** Enables you to specify left-to-right or right-to-left text direction for the content in an element.
- **draggable** Specifies whether an element is draggable.
- **dropzone** Enables you to specify the behavior of the dragged data when it's dropped. Data can be copied, moved, or linked.
- **hidden** Specifies that an element is not relevant.
- **id** Specifies a unique id for an element.
- **lang** Specifies the language (English, French, German, and so on) of the element's content.
- **spellcheck** Used with the lang attribute to enable you to indicate whether the element is to have its spelling and grammar checked.
- **style** Specifies an inline CSS style for the element.
- **tabindex** Sets the tabbing order of the element.
- **title** Provides extra information about the element.

You'll see many examples of these global attributes in this book.

Working with self-closing tags

You can represent any element that contains no content as a self-closing tag. A *self-closing tag* is a beginning tag and an ending tag in one. You end the starting tag with a space, slash, and greater-than symbol. For example, the *<br>* element cannot have any content, so here is the beginning and ending tag in one: *<br />*.

In XML, any empty element can be written with a self-closing tag, but in HTML5, this can cause problems in different browsers. The rule of thumb is to use self-closing tags for tags that cannot have content, such as the *<br />* tag. Empty elements that are capable of having content but currently don't have content should have separate end tags. An example is *<div></div>*; there is no content, but the beginning and ending tags still exist.

NOTE **USE THE <SCRIPT> ELEMENT CAREFULLY**

When using the <*script*> element to reference an external JavaScript file, the element will not have content when used in this context, but you must always include a separate end tag as follows.

```
<script src="/Scripts/MyCode.js"></script>
```

A symptom of incorrect usage of the <*script*> element is when the browser renders a blank screen, but you can view the source, and all its content is there.

Quick Check

- **You want to use the <*script*> element to include a JavaScript file named MyCode.js in the scripts folder. What is the proper syntax?**

Quick Check Answer:

- <script src="/Scripts/MyCode.js"></script>

Working with void elements

Most but not all elements can have content, and the content can include elements with content. Elements are not required to have content, but some elements cannot have content. These are called *void elements*. For example, the <*br*> tag represents a line break and cannot have any content.

The following is a list of void elements in HTML5.

- **<area>** Defines a hyperlink area with some text in an image map
- **<base>** Specifies the document's base URL or target for all relative URLs in the document
- **<br>** Represents a line break
- **<col>** Defines the properties of one or more columns within a <*colgroup*> element
- **<command>** Defines a command that can be invoked by a user
- **<hr>** Specifies a thematic change in content
- **<img>** Defines an image
- **<input>** Defines a typed data field that allows the user to edit the data
- **<link>** Defines a relationship between a document and an external resource such as a cascading style sheet
- **<keygen>** Defines a key-pair generator control for forms that is used to encrypt data that will be passed to the server
- **<meta>** Defines metadata that describes the HTML document

- **<param>** Defines a parameter for an object
- **<source>** Defines a multimedia resource for a <*video*> or <*audio*> element
- **<wbr>** Optionally breaks up a large word at this element

In earlier versions of HTML, you just used the <*br*> tag with no ending tag to indicate that you wanted to start a new line on the webpage. With XHTML, this was a problem because all beginning tags are required to have matching end tags. HTML5 allows you to use a beginning tag with no end tag, but a better solution is to use self-closing tags.

Adding expando attributes

Expando attributes are attributes that you define. Expando attributes are also known as author-defined attributes or simply as custom attributes. Any time you want to attach data to an HTML tag, you can just create an attribute with the name of your choice and assign the data. However, the name you create might conflict with either an existing W3C-defined attribute name or a future W3C-defined attribute name. To ensure that you have no existing or future naming conflict, assign a name that is prefixed with "data-".

Quick check

- **You have a webpage with a <*span*> element that contains the customer's name. Along with the name, you want to include the customer number on the <*span*> element, but you don't want to display the customer number. How can you write the <*span*> element for a customer called Contoso Ltd with customer number 123?**

Quick check answer

- **Use an expando attribute to hold the customer number as follows.**

```
<span data-customerNumber='123'>Contoso Ltd</span>
```

Adding comments

You can add comments to your HTML source by using the following syntax.

```
<!--comment here -->
```

Comments are not displayed on the rendered browser page but are sent to the browser. Comments can help document your source.

No spaces are allowed between the <! characters and the -- characters at the beginning of the comment, but spaces are allowed between the -- characters and the > character at the end of the comment tag. This seemingly weird behavior means that you cannot have back-to-back dashes (--) in your comment because this combination causes HTML syntax errors. In addition, you cannot end a comment with three dashes, such as <!-- and then it happened---> because this also generates a syntax error.

Adding conditional comments

Only Internet Explorer recognizes *conditional comments*, which enable you to add a browser-specific source that executes if the browser is Internet Explorer but is treated as a comment by other browsers. You can add conditional comments to your HTML document by using the following syntax.

```
<!--[if lte IE 7]>  <html class="no-js ie6" lang="en"> <![endif]-->
<!--[if lt IE 7]>   <html class="no-js ie6" lang="en"> <![endif]-->
<!--[if IE 8]>      <html class="no-js ie8" lang="en"> <![endif]-->
<!--[if gt IE 8]>   <html class="no-js" lang="en">     <![endif]-->
<!--[if gte IE 9]>  <html class="no-js" lang="en">     <![endif]-->
<!--[if !IE]> -->   This is not Internet Explorer!<br />            <!-- <![endif]-->
```

The first conditional comment checks whether the browser is Internet Explorer and the version is earlier than or equal to 7. The next conditional comment checks whether the browser is Internet Explorer and the version is earlier than 7. The next conditional comment checks whether the browser is Internet Explorer and the version is 8. The next conditional comment checks whether the browser is Internet Explorer and the version is later than 8, followed by a check to see whether the browser is Internet Explorer and the version is later than or equal to 9. The last line checks whether the browser is not Internet Explorer. Note that the syntax of the last line is different from the others.

Creating an HTML document

Now that you've seen the various elements and attributes, it's time to group them in a meaningful way to create an HTML document. The HTML document contains an outer structure, metadata, and some content.

Basic document structure

Every HTML document should have a basic structure that consists of a *<!DOCTYPE html>* declaration, which historically has indicated the version of HTML to the browser. In HTML5, this indicates to the browser that it should be in no-quirks mode. *No-quirks mode* causes the browser to operate in an HTML5-compliant manner. Next is the root *<html>* element, which contains the *<head>* element and the *<body>* element.

The *<head>* element contains hidden information such as metadata that describes the HTML document and instructions. The following is an example of metadata in the *<head>* element.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>title here</title>
  </head>
  <body>
    content here
  </body>
</html>
```

In this example, the *<meta>* element describes the character set as *utf-8*, which is an efficient form of unicode in which the English language characters (ASCII) require only a single byte to be represented, and in which other languages can have characters that are represented with up to 4 bytes. This is the most common character set used in HTML and XML documents.

This example also contains a *<title>* element, which is important because it serves the following purposes.

- Displays in the browser toolbar
- Provides the default name for the page when it is added to favorites
- Displays the title when a search engine displays the page in the search results

The *<body>* tag contains the displayable contents.

Using special characters (HTML entities)

You might want to display the < and > characters on your webpage, but you've seen that the less-than and greater-than characters define tags. These characters can be displayed in your content by using either the entity name or entity number as follows.

&entity_name;

or

&#entity_number;

There are many HTML entities, but Table 1-1 lists the most common HTML entities you will use in your HTML document.

TABLE 1-1 Reference to common entities

Display	Entity Name	Entity Number	Description
&	&amp;	&#38;	Ampersand
>	&gt;	&#62;	Greater-than sign
<	&lt;	&#60;	Less-than sign
"	&quot;	&#34;	Double quotation
©	&copy;	&#169;	Copyright
®	&reg;	&#174;	Registered trademark
™	&trade;	&#8482;	Trademark
	&nbsp;	$#160;	Nonbreaking space

NONBREAKING SPACE

If you try to embed a series of spaces into your HTML document, the browser normalizes contiguous white-space characters (such as spaces, tabs, and line breaks) and renders only a single space. This is usually a desirable feature because it enables you to format your HTML source content in a manner that is most readable in source mode while eliminating white-space in the rendered output.

When you want to display several spaces, you can use the nonbreaking space character. Nonbreaking space is also known as nonbreak space, nonbreakable space, and hard space. In addition to preventing the collapse of contiguous whitespace, the *nonbreaking space* prevents the automatic line break between words that you want to keep together on the same line.

Consider an HTML document in which you want to display 10 mph, where there is a space between the number 10 and the mph. You want to ensure that mph will not be separated from the number 10 by being moved to the next line. In your HTML document, use 10&nbsp;mph to keep the number 10 and mph together.

Lesson summary

- An element is composed of a starting tag, inner content, and an ending tag.
- Browsers ignore tags that are not recognized.
- HTML5 originates from HTML 4.01, not from XHTML.
- The W3C is responsible for developing open standards for the web.
- HTML elements provide structure, CSS style sheets provide presentation, and JavaScript provides behavior.
- Use lowercase tag names.
- Attribute values should always be quoted using either single quotes or double quotes.
- Boolean attributes are attributes whose mere presence on the starting tag indicates that the option is set.
- HTML5 defines global attributes, which are the set of attributes that can appear on any HTML5 element.
- Self-closing tags are tags whose beginning and ending tags are together to create an element with no content. Self-closing tags should be used only with elements that cannot have content.
- Void elements cannot have content. They should be created by using self-closing tags.
- Expando attributes are attributes that you define and are also known as author-defined attributes or custom attributes. Prefix these attributes with “data-“.
- You can use conditional comments to add a browser-specific source that will work with Internet Explorer but be treated as a comment by other browsers.

- HTML entities are special characters and can be embedded in your HTML document by using the ampersand (&), the entity name, and a semicolon (;). You can also use the ampersand (&), the hash symbol (#), the entity number, and the semicolon (;).
- Nonbreaking spaces can be used to render several contiguous spaces. You can also use nonbreaking spaces to keep two words from being separated by a line break.
- The id attribute specifies a unique identifier for an element.

Lesson review

Answer the following questions to test your knowledge of the information in this lesson. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You want to create an expando attribute on several *<h3>* tags that display vehicles for sale. The expando attribute will store the VIN (vehicle identification number) of the vehicle for sale. Which of the following is the most appropriate example of creating the expando attribute?
   A. *<h3 vin=*'current VIN here'>1965 VW Beetle*</h3>*
   B. *<h3 id=*'current VIN here'>1965 VW Beetle*</h3>*
   C. *<h3 data-vin=*'current VIN here'>1965 VW Beetle*</h3>*
   D. *<h3 datavin=*'current VIN here'>1965 VW Beetle*</h3>*
2. Which technology is HTML5 preceded by and derived from?
   A. HTML 4.01
   B. SGML
   C. XHTML 1.0
   D. XML
3. How should you start each HTML5 document?
   A. *<html>*
   B. *<head>*
   C. *<title>*
   D. *<!DOCTYPE html>*
4. You want to use the disabled Boolean attribute on a text box. How can you accomplish this? (Choose all that apply.)
   A. <input name='firstName' type='text' disabled />
   B. <input name='firstName' type='text' disabled='' />
   C. <input name='firstName' type='text' disabled='true' />
   D. <input name='firstName' type='text' disabled='disabled' />

Lesson 2: Embedding content

Soon, you will want to embed content in your HTML document. The content might be from an existing webpage, or you might embed images in your HTML document. You might also embed Adobe Flash applications. You can embed many interesting elements, and this lesson covers many of the ways to embed content.

After this lesson, you will be able to:

- Embed HTML documents in another HTML document by using inline frames.
- Create hyperlinks to remote or local HTML documents.
- Add images and image maps to the current HTML5 document.
- Embed plug-in content.

Estimated lesson time: 30 minutes

Embedding HTML by using inline frames

You can use the <*iframe*> element to embed an inline frame that contains an HTML document within the current HTML document. This can be useful when you want to create reuse functionality on your site; for example, when you want to create a common header that will show on all pages of your website. This can also be useful when you want to include an HTML page from another website on your page.

The <*iframe*> element creates a nested browser context into which another HTML document can be loaded. Loading an HTML document creates a browsing context for that document. The document that contains an <*iframe*> is contained within the parent browser context, where the document that is loaded into the <*iframe*> element is within the nested browser context.

You can navigate nested browsing contexts by using the following properties of the window object.

- **window.top** A WindowProxy object representing the top-level browsing context
- **window.parent** A WindowProxy object representing the parent browsing context
- **window.frameElement** An element that represents the browsing context container but returns null if there isn't one

The <*iframe*> element has a src (source) attribute and a name attribute. The src attribute can be set to the absolute or relative URL of the HTML document that you want to include, as shown in the following sample.

```
<iframe src="menu.html"></iframe>
```

The name attribute sets the browsing context name, which is useful when you need to reference the <*iframe*> element, possibly as the target of a hyperlink, as described in the

"Working with hyperlinks" section that follows. A valid browsing context name is any string with at least one character that does not start with an underscore because the underscore is used for these special key names: _blank, _self, _parent, and _top.

Sandboxing embedded content

Sandboxing is a means for preventing malware and annoyances such as pop-ups from being introduced when the content is embedded on your HTML page. The <*iframe*> element provides the sandbox attribute for this purpose.

The sandbox attribute places a set of extra restrictions on any content hosted by the iframe. When the sandbox attribute is set, the content is treated as being from a unique and potentially dangerous origin. Forms and scripts are disabled, and links are prevented from targeting other browsing contexts. Consider the following example.

```
<iframe sandbox src="http://someOtherDomain.net/content">
</iframe>
```

In the example, the source is referencing potentially hostile content in a different domain. This content will be affected by all the normal cross-site restrictions. In addition, the content will have scripting, plug-ins, and forms disabled. The content cannot navigate any frames or windows other than itself.

The restrictions can be overridden by space-separating any of the following.

- **allow-forms** Enables forms
- **allow-same-origin** Allows the content to be treated as being from the same origin instead of forcing it into a unique origin
- **allow-scripts** Enables scripts except pop-ups
- **allow-top-navigation** Allows the content to navigate its top-level browsing context

In the following example, allow-same-origin, allow-forms, and allow-scripts are enabled. On the surface, it might seem that the sandbox is not providing any protection, but the sandbox still disabling plug-ins and pop-ups.

```
<iframe sandbox="allow-same-origin allow-forms allow-scripts"
     src="http://otherContent.com/content.html"></iframe>
```

Seamless content embedding

The <*iframe*> tag has a seamless attribute that indicates that the source content is to appear as though it's part of the containing document. This means that the <*iframe*> element will not have borders and scrollbars. The seamless attribute is a Boolean attribute, so its presence on the <*iframe*> tag indicates that you want this option, but there are three ways to set a Boolean attribute. Here are three ways to specify seamless embedding of content.

```
<iframe seamless="seamless" src="http://otherContent.com/content.html"></iframe>
<iframe seamless="" src="http://otherContent.com/content.html"></iframe>
<iframe seamless src="http://otherContent.com/content.html"></iframe>
```

At the time of this writing, the seamless attribute is not supported on any browsers, but its intent is to blend the external content into the current HTML document so the HTML page does not look like it has embedded content. The alternative is to use CSS to obtain a similar presentation.

Working with hyperlinks

The *<a>* element creates a link to an external HTML document (external link) or jumps to a location in the current HTML document (internal link). The content of the *<a>* element is displayed in the browser with the following default appearance.

- **Unvisited link** Underlined and blue
- **Visited link** Underlined and purple
- **Active link** Underlined and red

The *<a>* element has the href attribute, which you usually use to specify the link destination. If the link is external, the href can be populated with either a relative or absolute URL as follows.

```
<a href="ExpenseReports.html">Expense Report Page</a>
<a href="http://www.contoso.com/SalesReports.html">Sales Report Page</a>
```

If the link is internal, the href will contain the hash (#) symbol followed by the id of the tag that you want to jump to. If you use only the hash symbol, clicking the link takes you to the top of the HTML document. Here are two examples.

```
<a href="#">Top</a>
<a href="#BillingAddress">Go To Billing Address</a>
```

Specifying the hyperlink target

When you're on a webpage and you click a hyperlink to an external resource, the external resource opens in the current browser window. If the external link is to a page on your website, this behavior probably makes sense. If the external link is to a different website, you might want to open a new browser window. By using the target attribute, you can control the link behavior by assigning one of the following.

- **_blank** Open in a new browser window
- **_parent** Open in the parent frame or window
- **_self** Open in the current window or frame (default)
- **_top** Open in the topmost frame, thus replacing the contents of the window
- **<iframe_name>** Open in the *<iframe>* element with matching name attribute

When you have a menu with hyperlinks that shows on every page, you might want to create an *<iframe>* element with its name attribute set to content and then set the target of all menu links to be content so the pages load into the *<iframe>* element as shown in the following example.

```
Main Menu<br />
<a href="Calendar.html" target="content">Calendar</a><br />
<a href="HumanResources.html" target="content">Human Resources</a><br />
<a href="ExpenseReports.html" target="content">Expenses</a><br />
<a href="Commissions.html" target="content">Commissions</a><br />
<br />
<iframe name="content"></iframe>
```

Sending email with hyperlinks

You can use mailto protocol to send email messages. The mailto URL accepts the following parameters: subject, cc, bcc, and body. The parameters can be entered in any order by adding a question mark (?) after the email address and separating the parameters with the ampersand (&). Some examples of an email hyperlink are as follows.

```
<!-- basic mailto -->
<a href="mailto:sales@contoso.com">Contact Sales</a>

<!-- add the name, notice that email is wrapped with &lt; and &gt; -->
<a href="mailto:Joe&lt;sales@contoso.com&gt;">Contact Joe in Sales</a>

<!-- multiple recipients comma separated -->
<a href="mailto:sales@contoso.com,service@contoso.com">Contact Sales and Service</a>

<!-- add carbon copy -->
<a href="mailto:sales@contoso.com?cc=service@contoso.com">Contact Sales cc Service</a>

<!-- add blind carbon copy -->
<a href="mailto:sales@contoso.com?bcc=service@contoso.com">Contact Sales</a>

<!-- basic mailto with message -->
<a href="mailto:sales@contoso.com?body=call me.">Contact Sales with call me message</a>

<!-- basic mailto with multi line message -->
<a href="mailto:sales@contoso.com?body=call me.%0AThanks">Contact Sales with multi line
message</a>

<!-- basic mailto with subject and message -->
<a href="mailto:sales@contoso.com?subject=hi&body=call me.">Contact Sales with hi
subject
 and call me message</a>
```

Adding images to your HTML document

When you want to embed an image in your HTML document, use the *<img>* element. The *<img>* element does not have an ending tag; it’s a void element. The *<img>* element has required attributes of src (abbreviation for source) and alt (abbreviation for alternate). Use the