Microsoft

Essentials of Developing Windows Store Apps Using HTML5 and JavaScript

Exam Ref 70-481

Wouter de Kort

Microsoft

Exam Ref 70-481 Essentials of Developing Windows Store Apps Using HTML5 and JavaScript

Wouter de Kort

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

Library of Congress Control Number: 2014940677
ISBN: 978-0-7356-8529-1

Printed and bound in the United States of America.

First Printing

Acquisitions Editor: Anne Hamilton
Developmental Editor: Karen Szall
Editorial Production: Box Twelve Communications
Technical Reviewer: Todd Meister
Cover: Twist Creative • Seattle

Contents at a glance

Introduction *xiii*

Preparing for the exam *xvii*

CHAPTER 1 **Design Windows Store apps** **1**

CHAPTER 2 **Develop Windows Store apps** **57**

CHAPTER 3 **Create the user interface** **125**

CHAPTER 4 **Program user interaction** **191**

CHAPTER 5 **Manage security and data** **243**

Index *295*

This page intentionally left blank

Contents

Introduction **xiii**

Microsoft certifications *xiii*

Acknowledgments *xiv*

Free ebooks from Microsoft Press *xiv*

Errata, updates, & book support *xv*

We want to hear from you *xv*

Stay in touch *xv*

Preparing for the exam **xvii**

Chapter 1 Design Windows Store apps **1**

Objective 1.1: Design the UI layout and structure 1

Evaluating the conceptual design 2

Deciding how the UI will be composed 7

Designing for the inheritance and reuse of visual elements 10

Designing for accessibility 11

Deciding when custom controls are needed 12

Using the Hub App template 13

Objective summary 15

Objective review 16

Objective 1.2: Design for separation of concerns 17

Planning the logical layers of your solution to meet application requirements 17

Designing loosely coupled layers 19

Incorporating WinMD Components 22

Objective summary 24

Objective review 25

Objective 1.3: Design and implement Process Lifetime Management (PLM). 26

Choosing a state management strategy 26

Handling the onactivated event 27

Handling the suspend event (oncheckpoint) 31

Preparing for app termination 32

Checking the ActivationKind and previous state 37

Objective summary 39

Objective review 39

Objective 1.4: Plan for an application deployment. 40

Planning a deployment based on Windows 8 application certification requirements 40

Preparing an app manifest (capabilities and declarations) 44

Signing an app 48

Planning the requirements for an enterprise deployment 49

Objective summary 50

Objective review 51

Answers. 52

Chapter 2 Develop Windows Store apps 57

Objective 2.1: Access and display contacts. 57

Calling the ContactPicker class 58

Filtering which contacts to display 59

Displaying a set number of contacts 61

Creating and modifying contact information 65

Selecting specific contact data 66

Objective summary 68

Objective review 68

Objective 2.2: Design for charms and contracts 69

Choosing the appropriate charms based on app requirements 69

Designing an application to be charm- and contract-aware 74

Configuring the application manifest for correct permissions 75

Objective summary 76

Objective review 77

Objective 2.3: Implement search 78

Providing search suggestions using the SearchPane and SearchBox control classes 78

Searching and launching other apps 81

Providing and constraining search within an app 81

Providing search result previews 82

Implementing activation from within search 84

Configuring search contracts 84

Objective summary 86

Objective review 86

Objective 2.4: Implement Share in an app 87

Using the DataTransferManager class to share data with other apps 87

Accepting sharing requests by implementing activation from within Share 90

Limiting the scope of sharing using the DataPackage object 92

Implementing in-app Share outside of the Share charm 93

Using web links and application links 93

Objective summary 95

Objective review 96

Objective 2.5: Manage application settings and preferences. 97

Choosing which application features are accessed in AppSettings 97

Adding entry points for AppSettings in the Settings window 98

Creating settings flyouts using the SettingsFlyout control 99

Adding settings options to the SettingsFlyout control 101

Storing and retrieving settings from the roaming app data store 105

Objective summary 108

Objective review 108

Objective 2.6: Integrate media features 109

Supporting DDS images 110

Implementing video playback 111

Implementing XVP and DXVA 112

Implementing TTS 113

Implementing audio and video playback using HTML5 DRM 115

Objective summary 116
Objective review 116
Answers. 118

Chapter 3 Create the user interface 125

Objective 3.1: Implement WinJS controls. 125
Using a FlipView control 126
Using a flyout 130
Using a Grid layout and a List layout 132
Using a menu object 135
Using a WebView control 138
Using an item container 140
Using the Repeater control 142
Objective summary 145
Objective review 146

Objective 3.2: Implement HTML layout controls. 147
Implementing layout controls to structure your layout 147
Implementing templates and bindings 154
Supporting scrolling and zooming with CSS3 156
Managing text flow and presentation, including overflow 159
Objective summary 161
Objective review 161

Objective 3.3: Create layout-aware apps to handle windowing modes . 162
Using CSS3 media queries to adapt to different devices 162
Responding to changes in orientation 165
Adapting to new windowing modes by using the View-Management namespace 167
Managing settings for an apps view 168
Objective summary 170
Objective review 170

Objective 3.4: Design and implement the app bar .171
Determining what to put on the app bar based on app requirements 171
Styling and positioning app bar items 173

Designing the placement of controls on the app bar 175
Handling AppBar events 176
Objective summary 177
Objective review 178

Objective 3.5: Apply CSS styling. 179
Implementing gradients 179
Implementing Grid layouts 181
Implementing zooming 182
Implementing scroll snapping 182
Implementing media queries 183
Objective summary 183
Objective review 184

Answers. 185

Chapter 4 Program user interaction 191

Objective 4.1: Manage input devices 191
Capturing gesture library events 192
Creating custom gesture recognizers 196
Listening to mouse events or touch gestures 198
Managing stylus input and inking 198
Handling drag-and-drop events 199
Objective summary 201
Objective review 202

Objective 4.2: Design and implement navigation in an app. 202
Handling navigation events, checking navigation properties, and calling navigation functions by using the WinJS.Navigation namespace 203
Designing navigation to meet app requirements 206
Using semantic zoom 207
Loading HTML fragments 210
Objective summary 212
Objective review 213

Objective 4.3: Create and manage tiles 214
Creating and updating tiles and tile contents 214

Creating and updating badges (the TileUpdateManager class) 219
Responding to notification requests 221
app requirements 224
Objective summary 226
Objective review 226

Objective 4.4: Notify users by using toast 227
Enabling an app for toast notifications 227
Populating toast notifications with images and text by using ToastUpdateManager 229
Playing sounds with toast notifications 230
Responding to toast events 231
Controlling toast duration 232
Configuring and using Microsoft Azure Mobile Services for push notifications 232
Objective summary 235
Objective review 236

Answers. 237

Chapter 5 Manage security and data 243

Objective 5.1: Choose a data access strategy 243
Choosing the appropriate data access strategy based on requirements 244
Objective summary 250
Objective review 250

Objective 5.2: Retrieve data remotely. 251
Using XHR or HttpClient to retrieve web services 252
Setting appropriate HTTP verbs for REST 255
Handling progress of data requests 258
Consuming SOAP/WCF services 259
Using WebSockets for bidirectional communication 260
Objective summary 262
Objective review 262

Objective 5.3: Implement data binding 263

Binding data to controls by using data-win-control and data-win-bind 263

Choosing and implementing data-bound controls 265

Binding data to item templates such as WinJS.Binding.Template 266

Configuring an iterator with data-win-options 266

Enabling filtering, sorting, and grouping data in the user interface 267

Objective summary 269

Objective review 270

Objective 5.4: Manage Windows authentication and authorization 271

Storing and retrieving credentials by using the PasswordVault class 271

Implementing the CredentialPicker class 273

Verifying credential existence by using Credential Locker 276

Storing account credentials in app settings 278

Objective summary 280

Objective review 281

Objective 5.5: Manage web authentication 282

Using the Windows.Security.Authentication.Web namespace 282

Setting up OAuth2 for authentication 283

Setting up Single Sign-On (SSO) 284

Implementing the CredentialPicker class 285

Implementing credential roaming 285

Implementing the WebAuthenticationBroker class 285

Supporting proxy authentication for enterprises 286

Objective summary 288

Objective review 288

Answers. 290

Index *295*

This page intentionally left blank

Introduction

Building apps for all kinds of devices is becoming more and more popular. If it's your goal to prove that you have the skills to build apps for the Microsoft ecosystem, this book is for you.

This book focuses on building Windows Store apps with HTML, JavaScript, and CSS. With experience in building web applications—be it on the Microsoft platform or on another platform—you can now use your existing skills to build Windows Store apps that run on millions of devices and leverage all the functionality that Windows offers you.

This book covers Exam 70-481, Essentials of Developing Windows Store Apps Using HTML5 and JavaScript, meaning that it closely follows the outline of the exam to help you quickly find the content you need to prepare yourself for the exam.

You will learn how to design and develop your app, and how to create both a great UI and user experience while making sure that everything is secure—both for the user and for your app.

After finishing this book, you will understand how to build Windows Store apps that prepare you for the ever-growing market of building apps.

This book covers every exam objective, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions themselves and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the links you'll find in the text to find more information and take the time to research and study the topic. Great information is available on MSDN, TechNet, and in blogs and forums.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to *http://www.microsoft.com/learning/en/us/certification/cert-default.aspx*.

Acknowledgments

I'd like to thank the following people:

- Jeff Riley, for your excellent role in managing the production of this book.
- Karen Szall, for helping me through the whole editing process. I learned a lot from your feedback and advice.
- Todd Meister, for your help in reviewing all the content and all your suggestions.
- Devon Musgrave, for helping me through the early stages of acquisitions and contracts.
- To my wife, Elise, for her support.
- And to all the other people who played a role in getting this book ready. Thanks for your hard work!

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

http://aka.ms/ER481R2

If you discover an error that is not already listed, please submit it to us at the same page.

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

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to http://support.microsoft.com.

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

This page intentionally left blank

Preparing for the exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. While there is no substitution for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. We recommend that you round out your exam preparation plan by using a combination of available study materials and courses. For example, you might use this Exam Ref and another study guide for your "at home" preparation and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you.

Note that this Exam Ref is based on publicly available information about the exam and the author's experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

This page intentionally left blank

CHAPTER 1

Design Windows Store apps

When you have a great idea for an app, it's hard to resist the temptation to start up Visual Studio and begin working on the new app. Resisting that temptation is necessary for building a great app, however.

Windows Store apps are more than just a clever implementation; they require upfront design, during which you consider the design goals that Microsoft describes for them. Thinking about the goal of your app, designing a UI, and making sure that you follow the Windows Store design principles are only some of the steps you need to consider.

This chapter helps you design your app, from initial idea, to coding, and then planning for deployment. This chapter covers the first objective in the Exam 70-481 objective domain (and this major topic area makes up about 20 percent of the exam's material). Although this chapter isn't code-heavy, there is a lot of discussion about the process of designing an app.

IMPORTANT

Have you read page xvii?

It contains valuable information regarding the skills you need to pass the exam.

Objectives in this chapter:

- Objective 1.1: Design the UI layout and structure
- Objective 1.2: Design for separation of concerns
- Objective 1.3: Design and implement Process Lifetime Management (PLM)
- Objective 1.4: Plan for an application deployment

Objective 1.1: Design the UI layout and structure

Designing your UI is the most important part of building a great app that appeals to users. When going through the Windows Store, most users skim through screen shots and a short list of features that your app has to offer. Making sure that you stand out, both graphically and in the features you offer, is the core requirement for building a great app.

This objective covers how to:

- Evaluate the conceptual design
- Decide how the UI will be composed
- Design for the inheritance and reuse of visual elements (for example, styles, resources)
- Design for accessibility
- Decide when custom controls are needed
- Use the Hub App template

Evaluating the conceptual design

The release of Windows Phone 7 started a new era for Microsoft because it released a version of Windows that showcases beautiful apps that follow a strict design philosophy. With Windows 8, this experience is expanded to other devices. Although the possibilities in your app and in the Windows ecosystem can be overwhelming, Microsoft offers clear design guidance that can help you transform your idea into a real app. Using these principles when evaluating your own design can help you develop and fine-tune your app until it becomes something that everyone wants to have.

Microsoft design principles

Microsoft's five foundational design principles include:

- Pride in craftsmanship
- Fast and fluid
- Authentically digital
- Do more with less
- Win as one

When designing your app, be prepared to devote time to the smallest details of all parts of your app. Take *pride in craftsmanship*; see your app as a real work of craft. Make sure that you are using the correct fonts and typography, are aligning all your pixels correctly, and are taking pride in what you are doing.

Fast and fluid comes down to responsiveness and using the right animations. You have probably seen Microsoft PowerPoint presentations in which an author creates too many animations. Every slide comes in from a different direction, all elements appear with some kind of animation, and you are distracted from the real goal of the presentation. Your apps will also use animations, but animations and transitions should support the user experience, not distract from it. Making sure that your app is responsive and uses simple but direct animations to guide users through your app is what fast and fluid is all about.

What’s the icon you use to save a document in Visual Studio? It’s a floppy disk! Depending on your age, you might never have used a real floppy disk in your life. Maybe you save your files to a regular hard drive, to a solid-state drive (SSD), or directly to the cloud. Should the icon change depending on the location in which you save the file? Or should you accept the fact that the digital world is different from the real world? That’s what being *authentically digital* means. It can be a principle when designing small things such as adding a shadow to an element. Why mimic the real world by adding a shadow when the user knows there isn’t actually a shadow there?

Extending this principle, you can create digital experiences that are not possible in the real world. Semantic zoom is one such area. If you open the Weather app on your Windows 8 PC, you can use Ctrl+scroll wheel to zoom in and out. But instead of showing the information on the screen smaller or larger, the app switches to a completely new view. Within this view, you can pick a different category or switch dates. Although the process is different from what you expect in the real world, it is perfectly possible in a digital app.

Doing more with less is another core principle of designing a Windows Store app. Compare the screen shot of OneNote 2013 running on your desktop in Figure 1-1 to the screen shot of OneNote 2013 running as a Windows Store app in Figure 1-2.

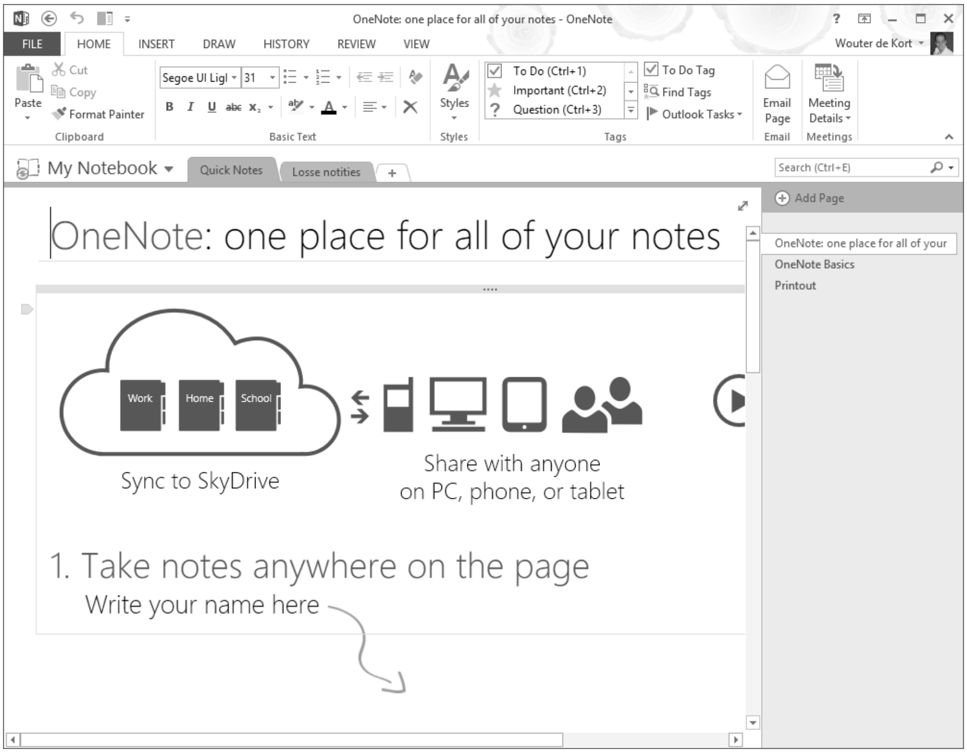


FIGURE 1-1 OneNote 2013 as a desktop app

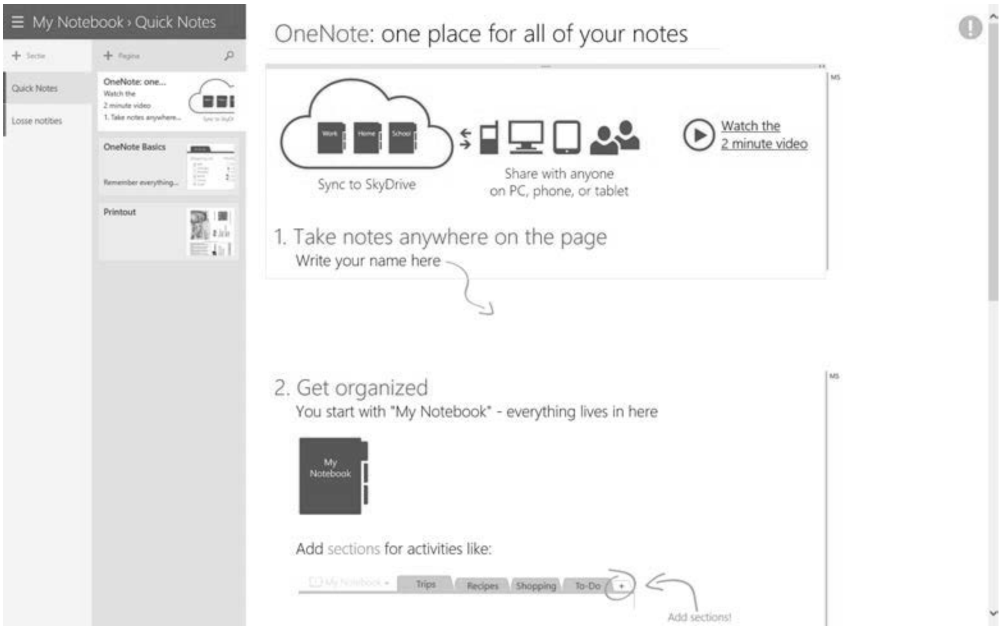

My Notebook › Quick Notes
Quick Notes
Losse notities
OneNote: one...
OneNote Basics
Printout
OneNote: one place for all of your notes
Sync to SkyDrive
Share with anyone on PC, phone, or tablet
Watch the 2 minute video
1. Take notes anywhere on the page
Write your name here
2. Get organized
You start with "My Notebook" - everything lives in here
My Notebook
Add sections for activities like:
Trips
Recipes
Shopping
To-Do
Add sections!

FIGURE 1-2 OneNote 2013 as a Windows Store app

The desktop app shows a lot more on the screen, but is that really beneficial? What is OneNote about? Isn't it about taking notes, viewing pages, and easily jotting down your thoughts? The Windows Store app version of OneNote focuses on those things. It shows options for formatting and other commands only when you need them, so a much larger portion of the screen can be used to accomplish the goal of your app: showing notes. Reducing chrome and navigation elements is key. Of course, reducing elements shouldn't be done so thoroughly that users are completely lost in your app. For example, Microsoft made the decision to add a search box directly to the interface of the Windows Store, as you can see in Figure 1-3. This search box immediately helps users find the right place to search for an app.

The last design principle, *Win as one*, is about integrating with the existing Windows 8 ecosystem. Users learn that they can change settings for applications by using the Settings charm, they expect to swipe up to open the app bar, and they can integrate with other apps that are installed on their device. Making sure that your app immediately feels familiar helps them use your app and ensures that they will return to it.

These five principles should guide you through the design of your app. Try to keep these design principles in mind and let them guide every decision you make during development.

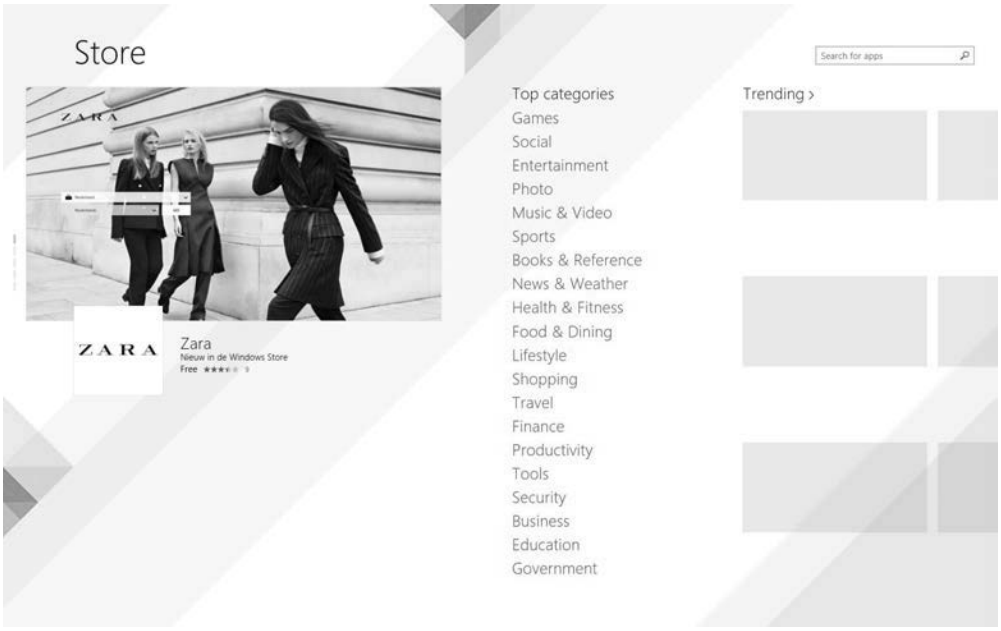


FIGURE 1-3 A Windows Store app showing the search box

EXAM TIP

Make sure you know the five design principles for the exam. Understand the difference between a regular desktop app and a Windows Store app.

Creating a vision

Look at the design principles that should guide your app. Your app shouldn't be a monolithic, giant application that can do everything a user will ever want. Focus on the experience that you want to offer instead of the features. This is where a *vision* comes in. You should have a clear idea of what your app is great at. Imagine that someone comes to you with the idea of building a ToDo app. They visualize that users should be able to add tasks, mark them as complete, and see a quick overview of what they still have to do. Try to determine what the experience is that you want to give to users instead of only a list of features.

Why would a user use your app? What's your app all about? After some brainstorming, suppose that you come up with the following list:

- Add new tasks
- Mark tasks as completed

- Collaborate with others on some tasks
- Remove tasks
- Get a list of uncompleted tasks
- Search through tasks
- Specify deadlines for tasks

Take a step back from this list of features and think about what would make your app great. What scenario would make your app stand out from the competition?

One scenario that jumps out is the one about collaborating on tasks. Maybe you can think of other scenarios that could be even better, but try to pick one single scenario to guide your app.

Next, formulate your apps vision in a *great at statement*:

My ToDo app is great at letting people work together on a list of tasks.

Suddenly you have a clearer vision of your app. You can now check each feature that you think of against your vision. Does it help with your vision or distract users from it?

The second step is deciding what user activities to support. Which steps are important for users reaching the scenario of working together on a list of tasks? What is the ideal flow that they should go through? Maybe you come up with something like this:

- Create a new task list
- Add tasks
- Share it with another user
- Report progress

These are the basic steps that form the flow of your app.

Now that you know what you want and which user activities you want to support, you should start researching your platform. Windows 8 offers a broad set of capabilities that you can use—from different controls and animations to ways to integrate with other applications. For example, you can use the *Share contract* to implement the idea of sharing a task list with someone else. You can use built-in controls to show a list of tasks and use the standard touch gestures to let users select tasks and mark them as completed by clicking a button in the app bar.

And there are many more possibilities! This book is about all the options that Windows 8 gives you when building apps, from tiles to notifications, from form factors to contracts. Many things are possible when developing Windows Store apps, and you should be aware of those that can help you with your design.

Now your app idea is coming together. You have a clear vision, you have targeted activities that you want to support, and you understand the wealth of options that you can use when coding your app.

But before you start developing, you should create some prototypes of your app. You might start with simple sketches on paper, but you can also use tools such as Expression Sketchflow or even PowerPoint with the Storyboarding plug-in.

Figure 1-4 shows an example of what a prototype can look like in PowerPoint. The advantage of these types of prototypes is that you can have users test them and give you feedback without having to do any time-consuming development.

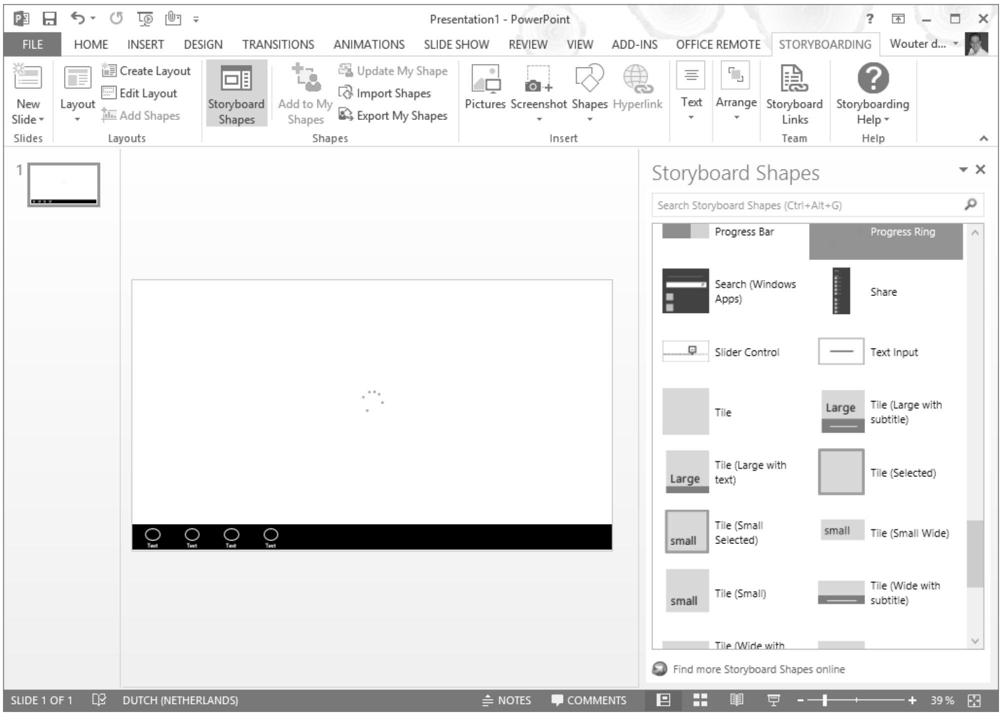


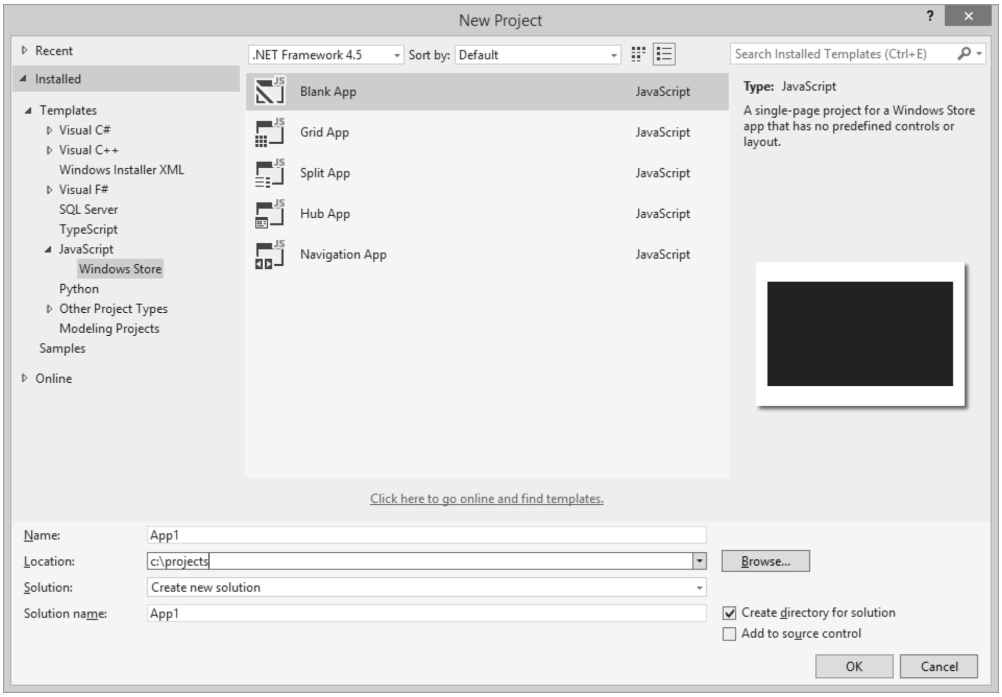
FIGURE 1-4 PowerPoint storyboarding for Windows Store apps

After following these steps, you will have some clear prototypes of your app and you can start thinking about implementation.

Deciding how the UI will be composed

When creating a new Windows Store app in Visual Studio, you have several different options. Figure 1-5 shows the project templates from which you can choose.

Deciding on the template to start with is an important step. If you have created some sketches, you probably have figured out what the flow of your application will be. Choosing a template that matches your flow closely can save you a lot of work.



Type: JavaScript

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

Name: App1

Location: c:\projects

Solution: Create new solution

Solution name: App1

☑ Create directory for solution

☐ Add to source control

FIGURE 1-5 Creating a new JavaScript Windows Store app

The available templates are the following:

- Blank App
- Grid App
- Split App
- Hub App
- Navigation App

Selecting the right template depends on the type of navigation that you want to support. You can choose a *flat navigation* if all content resides on the same hierarchical level (Internet Explorer or a game, for example). If you have content that is *hierarchical*, such as categories and items, you can choose for a hierarchical navigation pattern.

The *Blank App template* is the only template that comes without a navigation model. This template gives you an empty app with only HTML, cascading style sheets (CSS), and JavaScript files, together with some required files for a package manifest; a file to sign your app; and some default images.

Although it is not recommended to start with the Blank App template, you can use it if you need a simple environment to test some features or if you have layout requirements that don't fit in the standard Windows Store model.

The *Grid App template* is the basis for apps that let the user browse through categories and then navigate to individual items (a shopping or news app, for example). After creating this project, you have a default structure of categories with items in them shown in three layers: overview, category details, and item details. It implements navigation between pages and comes with some sample data that you can customize.

The *Split App template* combines showing a list of items with item details on one page. Think of it as a master-detail view of your data. It can be useful for building a blog reader app, for example.

The *Hub App template* was added to Visual Studio 2013 with the release of Windows 8.1. This template shows content in a horizontally panning view. The Hub App template is different from other project templates because it uses a mix of sections that represent the items in your app. Whereas the Grid App and Split App templates have a very specific way of showing items, the Hub App template mixes all those approaches to create a compelling UI.

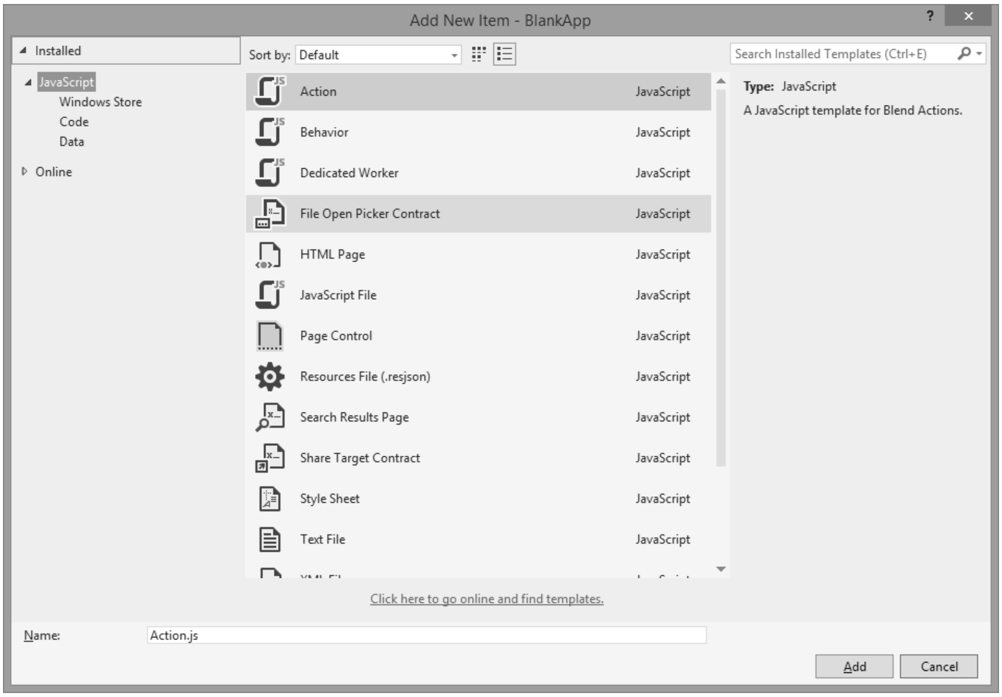
EXAM TIP

Be sure to try out the different project templates in Visual Studio. Become familiar with the content of each template and try sketching some simple app ideas (such as a shopping app or a chess game) with each so you understand how a template can get you started in the right direction.

One aspect of the templates in Visual Studio that might not be immediately obvious is that those templates implement a *single-page application (SPA)* structure. Regular websites use multiple pages and hyperlinks to navigate from one page to another. While the page is loading, your screen refreshes and a complete page is loaded.

An SPA uses a different architecture. Instead of having multiple pages, you load one page when it starts and then use JavaScript to enable and disable elements. This process avoids complete page reloads and gives your app a more fluid interface. When composing your UI, keep this important aspect in mind. You can reuse the skeleton of your page and switch different elements when navigating from "page to page."

After you select your project template, you can use several item templates, as shown in Figure 1-6. Those templates help you implement specific functionality, such as contracts, in your app.



Action	JavaScript
Behavior	JavaScript
Dedicated Worker	JavaScript
File Open Picker Contract	JavaScript
HTML Page	JavaScript
JavaScript File	JavaScript
Page Control	JavaScript
Resources File (.resjson)	JavaScript
Search Results Page	JavaScript
Share Target Contract	JavaScript
Style Sheet	JavaScript
Text File	JavaScript

Type: JavaScript

A JavaScript template for Blend Actions.

Click here to go online and find templates.

Name: Action.js

FIGURE 1-6 The Add New Item dialog box

Designing for the inheritance and reuse of visual elements

When building software, you can easily fall into the trap of simply copying and pasting elements (HTML, CSS, or some JavaScript) whenever you need them. However, although simply duplicating code can give you some velocity, it slows you down in the end.

The more code you have in your project, the harder it is to understand. *Maintainability* also becomes an issue; making a change to each copy of an element is not only cumbersome but can also lead to errors when you forget to change a copy.

During the design phase, you can easily see where certain elements will be repeated, so make sure from the start that you find a suitable solution instead of a plain copy-and-paste process.

Of course, remember principles such as *Keep It Simple (KIS)* and *You Aren't Gonna Need It (YAGNI)*, which remind you that you shouldn't go overboard planning to reuse elements that you will never reuse. Instead of wasting time on creating complex controls that can easily be reused but never will be, start with a simple solution and make sure that you adapt it whenever the requirements change.

Using HTML, CSS, and JavaScript for your app development gives you ample opportunities for reusing elements in your app.

CSS is probably the easiest to reuse. You should move styles that are used in multiple places in your app to a common file and then reference that file whenever needed. Especially because you are using the SPA architecture, you can load CSS elements once and then reuse them throughout your application.

JavaScript is also easily reusable. You can include certain utility functions in your JavaScript and call them from anywhere in your application. The next objective looks at structuring your JavaScript in a reusable way.

You can also reuse HTML by using JavaScript controls. For example, HTML PageControl enables you to reuse HTML pages by loading them through JavaScript. You can also build custom controls if you want to create a reusable unit of HTML, JavaScript, and CSS.

These options are discussed in more detail throughout the book.

Designing for accessibility

Imagine using your app if you are colorblind or completely blind. What if you are deaf or can’t use a mouse or touch device easily? Keeping users’ disabilities in mind is very important for the design process.

Maybe you are faced with legal requirements that force you to make your app accessible. Or maybe you are just thinking about all the possible users who can’t use your app if you don’t design for accessibility.

Fortunately, implementing accessibility in your app isn’t difficult. HTML already has good support for creating accessible websites, and your Windows Store app can expand on that foundation.

Make sure that your HTML not only looks nice but also specifies what it does. *Accessible rich Internet applications (ARIAs)* are defined for this purpose. An ARIA defines a set of special attributes that can be added to your HTML. Those attributes describe the description and role of elements that can be used by screen readers—for example, to help a user understand what an element does.

Next to using ARIA attributes, you should also make sure that your app is accessible only by using a keyboard, which means thinking about the tab index of elements, making sure that a user can use the arrow keys to navigate and implement accelerator keys.

You should also test your app under different conditions. The testing could be with varying resolutions and when using high-contrast teams or a larger font.

The Windows software development kit (SDK) comes with two tools that you can use to test the accessibility of your app: Inspect and UI Accessibility Checker (AccChecker). Of course, you can also do some manual testing: Unplug your mouse, change your color theme, and adjust font size. Narrator is an application that verifies your app can be used with a screen reader.

If you have followed the guidelines for accessibility, you can submit your app to the Windows Store as being accessible, which helps users with disabilities find your app more easily. Of course, Microsoft checks to see whether your app is accessible before allowing it.

MORE INFO **MAKING YOUR APP ACCESSIBLE**

To learn more about accessibility in Windows Store apps using HTML and JavaScript, see *http://msdn.microsoft.com/en-us/library/windows/apps/hh452681.aspx*. This page gives you links to examples of using ARIAs, checklists, and tools that you need to make your app accessible.

Deciding when custom controls are needed

When building your app, you'll use standard HTML elements and prebuild Windows Library for JavaScript (WinJS) controls created by Microsoft. With only these controls, you can build most apps without many problems. They can be styled with CSS, and you can often attach JavaScript to extend the behavior.

Sometimes you might need a custom control. Perhaps you want a calendar, some graph controls, or something else that's completely specific to your app.

First, see whether someone has already built your custom control. Companies such as Telerik create control suites for all types of applications. A good starting point is *http://services.windowsstore.com/*. You can find custom controls in the Controls & Frameworks section.

A basic reusable piece of code can be created by using HtmlControl or PageControl. More-complex controls can be implemented the same way as WinJS controls.

EXAM TIP

The exam requirements for Exam 70-481 don't state that you should be able to create a custom control. Exam 70-482 (Advanced Windows Store App Development Using HTML5 and JavaScript) requires you to create custom controls.

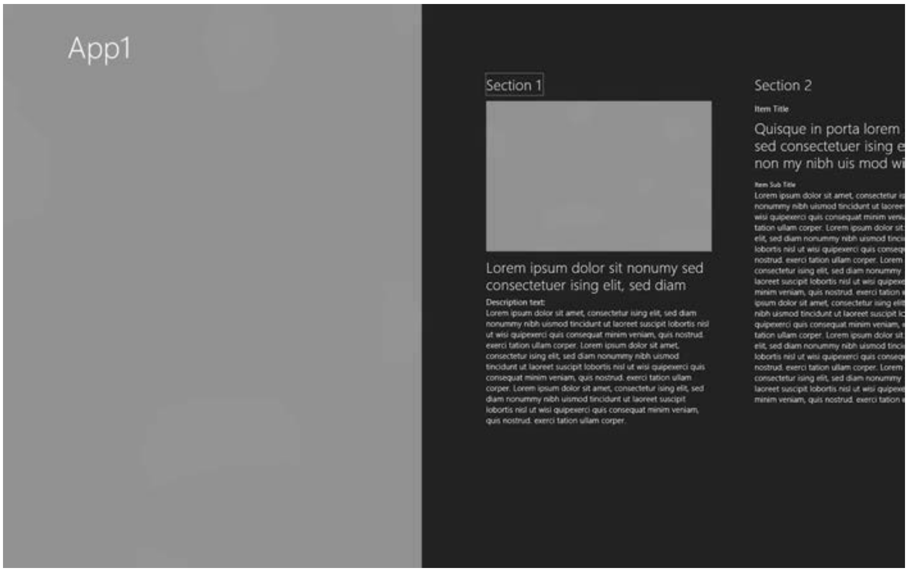
Using the Hub App template

You have learned about composing your UI and choosing a template. The exam specifically focuses on the Hub App template, so make sure that you understand how it works.

The Hub App template is an implementation of the hierarchical navigation pattern. The Start screen is shown in Figure 1-7.

The template has a horizontal layout that shows the sections you defined. Within those sections, you can show individual items. A user can select sections marked with > to view all the items in that section. When you select an individual item, a details page displays.

Figure 1-8 shows the initial files created by the template. The most interesting part is the Pages folder, which contains the Hub, Item, and Section pages. The data for Section and Item pages is loaded from the static data.js file in the js folder.

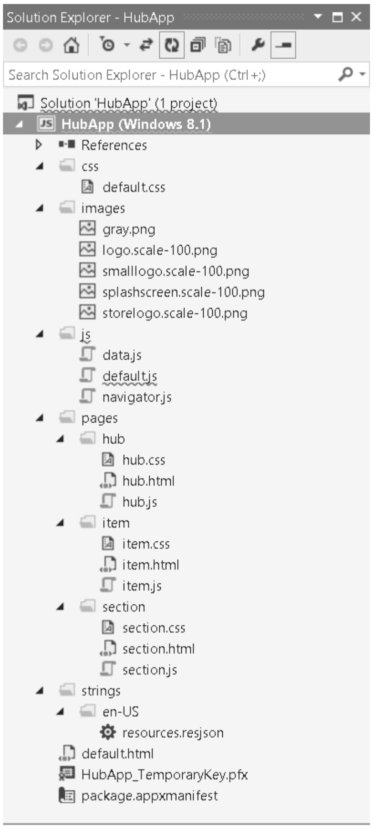


Lorem ipsum dolor sit nonumy sed consectetuer ising elit, sed diam

Description text:
Lorem ipsum dolor sit amet, consectetur ising elit, sed diam nonummy nibh uismod tincidunt ut laoreet suscipit lobortis nisl ut wisi quipexerci quis consequat minim veniam, quis nostrud. exerci tation ullam corper. Lorem ipsum dolor sit amet, consectetur ising elit, sed diam nonummy nibh uismod tincidunt ut laoreet suscipit lobortis nisl ut wisi quipexerci quis consequat minim veniam, quis nostrud. exerci tation ullam corper. Lorem ipsum dolor sit amet, consectetur ising elit, sed diam nonummy nibh uismod tincidunt ut laoreet suscipit lobortis nisl ut wisi quipexerci quis consequat minim veniam, quis nostrud. exerci tation ullam corper.

Quisque in porta lorem
sed consectetuer ising e
non my nibh uis mod wi

FIGURE 1-7 Hub App template Start screen


Solution Explorer - HubApp
Search Solution Explorer - HubApp (Ctrl+;)

```
Solution 'HubApp' (1 project)
  HubApp (Windows 8.1)
    References
    css
      default.css
    images
      gray.png
      logo.scale-100.png
      smalllogo.scale-100.png
      splashscreen.scale-100.png
      storelogo.scale-100.png
    js
      data.js
      default.js
      navigator.js
    pages
      hub
        hub.css
        hub.html
        hub.js
      item
        item.css
        item.html
        item.js
      section
        section.css
        section.html
        section.js
    strings
      en-US
        resources.resjson
    default.html
    HubApp_TemporaryKey.pfx
    package.appxmanifest
```

FIGURE 1-8 Files created by the Hub App template

Adapting the template is easy. Get some meaningful data in for your app, which can be static test data that you put in the data.js file or that is asynchronously loaded by the app from an external data source.

Adding your own styling and behavior is just as important, of course. Using the Hub App template as a foundation for your app steers you in the right direction for an attractive app!

MORE INFO **CHANGING THE HUB APP TEMPLATE**

Microsoft published a complete example that shows you how to change the Hub App template to load data asynchronously. This is a good way to get started with the template! You can find it here: *http://code.msdn.microsoft.com/windowsapps/Hub-template-sample-with-4b70002d*.

Thought experiment

Designing your app

In this thought experiment, apply what you've learned about this objective. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are implementing an app for a popular restaurant, and the owners want users to be able to place orders through the app. The app should keep track of the users' preferences and make special offers based on their purchase history.

1. **Formulate a "great at" statement for your app.**
2. **Decide which template to use as the basis for your app.**
3. **Decide whether your app should be accessible.**

Objective summary

- When designing your app, make sure to follow the Microsoft design principles: pride in craftsmanship, fast and fluid, authentically digital, do more with less, and win as one.
- Make sure that you have a "My app is great at" statement to focus your app on one single user scenario and implement it fully.
- Visual Studio offers project templates that you can use as a starting point for your app: the Blank App, Hub App, Navigation App, Split App, and Grid App templates.
- You can easily reuse HTML, CSS, and JavaScript throughout your app to create a consistent look and feel and to ensure maintainability.
- Accessibility is important for creating an app that can be used by users with disabilities.
- Custom controls that you create can be used to reuse HTML, JavaScript, and CSS.
- The Hub App template is a new template in Windows 8.1 that you can use to create attractive apps that use a mixed mode of showing content to the user.

Objective review

Answer the following questions to test your knowledge of the information in this objective. 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 are developing the ToDo app in collaboration with designers who are new at Windows Store development. They encourage you to use animations in your app. What should you do?
   - **A.** Explain that animations are of the past and are distractions that Windows Store apps should avoid.
   - **B.** Agree with them and let them describe a list of animations that they want to use.
   - **C.** Refer them to the documentation and show them the list of animations used in Windows Store apps.
   - **D.** Meet with the designers and create custom animations that are useful for your application.
2. Why is the "great at" statement so important for an app?
   - **A.** A great at statement isn't important; apps are not allowed to become popular because the platform doesn't support it.
   - **B.** A great at statement isn't important because apps are only allowed to have limited functionality.
   - **C.** Without a "great at" statement, you can't follow the Microsoft design principles.
   - **D.** It creates a vision that you can use to guide the development of your app and make sure your app truly excels in your supported user goal.
3. Which of the following Microsoft design principles are important when designing your app? (Choose all that apply.)
   - **A.** Pride in craftsmanship
   - **B.** Integrate with the cloud
   - **C.** Fast and fluid
   - **D.** Authentically digital
   - **E.** Do more with less
   - **F.** Win as one

Objective 1.2: Design for separation of concerns

Software development is still a relatively young profession. Although other disciplines have established well-defined and accepted rules for their craft, the rules of software development are still evolving.

This objective discusses the principles that the software industry has established in the last couple of decades. These principles revolve around building maintainable software that can be easily understood and extended. You will learn how to use layers to build your application and how Windows Metadata (WinMD) Components fit in the picture.

This objective covers how to:

- Plan the logical layers of your solution to meet application requirements
- Design loosely coupled layers
- Incorporate WinMD Components

Planning the logical layers of your solution to meet application requirements

When doing some handiwork (or watching someone else do it) you probably use a variety of tools. Some tools, like a multi tool, can be used for a multiple scenarios. However, you can imagine scenarios where some parts of a multi tool become obsolete or broken. Changes to one element of your tool will affect other parts. Having dedicated tools with a specific goal often use fuller and requires less maintenance.

Maintenance and having a specific goal is also important for software design. But when building software, it's a lot harder to recognize and avoid mistakes in these types of designs.

Separation of concerns (SoC) entails splitting a computer program into logical sections so that each section addresses a specific concern. SoC is important for creating maintainable applications that can be easily tested.

Typical examples of SoC are HTML, JavaScript, and CSS. Each performs a unique role in creating a webpage or app. HTML is the semantic structure of your page, CSS is the styling, and JavaScript adds client-side functionality.

You can mix these three concerns. For example, although you can add styling directly on your HTML elements, these designs are unnecessarily complex, hard to maintain, and difficult to extend.

How does SoC apply to your app? A typical app has to do multiple things: fetch data from somewhere, extract the necessary fields from it, and maybe perform some other validations on it before showing it on the screen. These tasks should not be plunged into one single monolithic object that spans your whole app. Instead, you should divide those tasks into separate areas and make them work together.

This process is called *layering*, and an application can consist of several logical layers. An application typically includes a UI layer, service layer, business layer, and data layer.

The layers have specific tasks. The business layer doesn't have to know how to fetch data; that is the responsibility of the data layer. The data layer doesn't know how the data is displayed on the screen; that is the work of the UI layer.

A typical diagram of this architecture is shown in Figure 1-9.

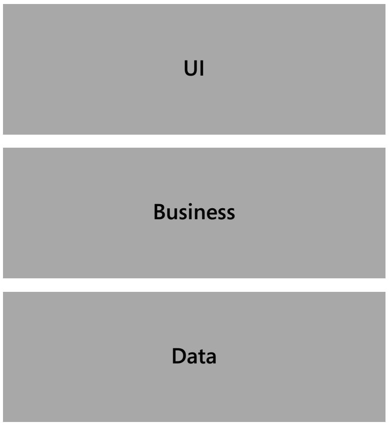


FIGURE 1-9 Diagram showing layering architecture

The diagram shows three typical layers stacked on top of each other. All layers also must address security and logging.

Before you start coding your app, you should have a reasonable idea of the different layers that you need in your application.

When you create a new app from one of the templates (the Hub App template, for example), you see that data.js is responsible for fetching data. In this way, you centralize all knowledge about data access to one location. Other parts of your app can call into the data object and use it without knowing anything about the specifics of your data storage mechanism.

Logical layers of an application differ from tiers. Maybe you have heard the term *N-tiered application*. A *tier* is a layer (or a couple of layers) that is physically separated from other layers. It might be a web service running somewhere in the cloud on Microsoft Azure that contains a piece of functionality from your app or from a database that stores the data for your app.

Logical layers can be placed on separate tiers, but it's not a requirement to do so to achieve a well-designed app.

Designing loosely coupled layers

JavaScript as a language has its own particular challenges for designing large-scale applications. Languages such as C++ or C# implement a paradigm called *object-oriented programming*, which enables you to create small classes that are targeted at doing one single task. You can configure scope for objects, group them, and build your application this way.

JavaScript is a powerful language, but it wasn't designed for building large applications. Ideas from object-oriented languages such as classes and modules are not built in to the JavaScript language.

Of course, you can just jump in and start developing your app without thinking too much about layering. However, as your app starts to grow, maintainability and the overall quality of your app will be negatively affected.

Avoiding global state

In JavaScript, everything you create is globally accessible by default, so every variable you declare can be modified anywhere in your code. Naming conflicts can occur when you declare the same variable twice, which can lead to unforeseen problems when a variable is modified somewhere in the application without an easy way to track the changes.

These conflicts don't occur for variables and functions declared inside a function. JavaScript limits the scope of those objects to the scope of the function. This limitation enables you to implement the concept of *private* (which you can find in languages such as C#) in JavaScript.

To avoid global state and create private data, the default JavaScript files created by the Visual Studio templates wrap their content like this:

```
(function () {
    ...
})();
```

What you see here is an *anonymous, self-invoking function*. The function is declared without a name, and it is immediately executed at the end of its declaration. This function allows you to scope all the items inside the function.

You don't want to keep all items private; some functions or variables should be exposed. To help you, WinJS uses the concept of namespaces.

By using the WinJS.Namespace.define method, you can set a name for your namespace and configure which items are accessible:

```
var namespacePublicMembers = { clickEventHandler: button1Click };
    WinJS.Namespace.define("startPage", namespacePublicMembers);
```

In this example, you define a namespace called startPage and expose a variable named clickEventHandler. This event handler points to the button1Click function defined inside your anonymous method.

Instead of having to use only plain functions and exposing them through namespaces, you can use WinJS to create classes. By using the WinJS.Class.define method, you can create a new class that has both behavior and data.

You can use the following plain JavaScript code to create a class-like object:

```
function Robot(name) {
    this.name = name;
}

Robot.prototype.modelName = '4500';
Robot.harmsHumans = false;
```

You create a class named Robot that expects a name on creation. It also has a modelName property that's unique for each instance. The harmsHumans property is *static*, meaning that it is shared across all instances.

Instead of using this syntax, WinJS exposes a helper method called WinJS.Class.define. You can use the following code to create your Robot class:

```
var Robot = WinJS.Class.define(
    // The constructor function.
    function(name) {
        this.name = name;
    },
    // The set of instance members.
    { modelName: "" },
    // The set of static members.
    { harmsHumans: false });

var myRobot = new Robot("Mickey");

myRobot.modelName = "4500";
Robot.harmsHumans = false;
```

WinJS also gives you a helper method to implement *inheritance*, which is a concept of object-oriented development in which you define base classes and derived classes. You can create a hierarchy of classes that all share behavior and data, but can also add additional elements to their base class.

A classic example is found with animals. If you have a base class Animal, you can add elements to it such as IsAlive or Age. Now you can derive specific subtypes such as Mammal or Bird. They can add their own data such as IsWarmBlooded or Fly.

There are problems, however. Can all birds fly? How should you express that not all types of birds can fly? You can start adding checks to make sure you don't execute a method that's not implemented on the current class, but going down this road where not all subclasses support the methods defined on a base class leads to code that's unmaintainable. Discussing all the fine-grained details of developing class hierarchies is outside the scope of this exam. If you start building more-complex applications, it pays to be familiar with object-oriented design concepts.

MORE INFO **OBJECT-ORIENTED DESIGN**

Lots of books, articles, tutorials, and other material exist on the topic of object-oriented design. A good starting point is reading material produced by Robert Martin, who is considered to be one of the founders of object-oriented design: *http://www.objectmentor.com/omSolutions/oops_what.html*.

Using strict mode

JavaScript is usually very forgiving of the way you write your code. You can use a variable without ever declaring it, write to a read-only property, extend objects that are marked as not extensible, delete functions, or duplicate properties and other strange errors that won't be immediately obvious when they start producing errors in your code.

Strict mode is a feature in JavaScript that you must explicitly enable. Enabling it results in better error-checking in your code to avoid the types of errors mentioned previously. You enable strict mode by adding the following line to your programs:

```
"use strict";
```

This line can be scoped to global scope (which applies to all code in your whole application, even external code), or you can use it inside a function that scopes it to the function.

Using TypeScript

JavaScript needs some help to become suitable for building large applications. Microsoft also noticed this lack, so it started developing *TypeScript*.

TypeScript is a superset of JavaScript that still compiles to plain JavaScript that can be used to run your apps. At development time, it adds extra features such as typing, classes, modules, generics, and inheritance.

These features significantly improve working with JavaScript. Look at the following TypeScript code:

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
var greeter = new Greeter("world");
```

The class keyword, constructors, and property typing are elements that are added by TypeScript, which allows the compiler to give you much better support. It finds errors, helps you with IntelliSense, and works with other Visual Studio features that improve your workflow.

There is nothing that stops you from using TypeScript for building your Windows Store apps. There are even TypeScript definition files for the Document Object Model (DOM) and WinJS libraries. As a JavaScript developer, you should definitely consider using TypeScript.

MORE INFO **TYPESCRIPT**

For more information on TypeScript, see the TypeScript website at *http://www.typescript-lang.org/*. An open-source repository for type definitions can be found at *https://github.com/borisyankov/DefinitelyTyped*. You can download definition files to start working with WinJS libraries.

Incorporating WinMD Components

When working on your Windows Store apps with HTML, JavaScript, and CSS, you call in to libraries defined in WinJS, which are built on the native C++ WinRT run time.

Calling in to a native dynamic-link library (DLL) from JavaScript is normally not supported. Microsoft put a lot of effort into creating an infrastructure that supports the interoperability of different languages to create apps for the Windows platform.

A regular C++ native component does not include metadata, which is necessary to create the correct mapping between the native components and the other languages. To make this work, Microsoft created a new file type named Windows Metadata (WinMD).

If you are running Windows 8, you can find these files in C:\Windows\System32\WinMetadata. The format of these files is the same as used by the .NET Framework for the Common Language Infrastructure (CLI).

WinMD files can contain both code and metadata. Those in your System32 directory contain only metadata, however. This metadata is used by Visual Studio to provide IntelliSense at design time. At run time, the metadata is used to signal that the implementation of all the methods found there is supplied by the run time, which is why the files don't have to contain actual code; they make sure that the methods are mapped to the correct methods in WinRT.

When building JavaScript apps, you have the choice to implement part of your application in another language, such as C# or C++. Those projects can contain code that's hard to implement in JavaScript while still integrating nicely with your app.

If you want to create your own WinMD assembly, create a WinRT Component in Visual Studio. The WinRT Component compiles down to a .winmd file that you can then use.

The following example shows some code that you can have inside your WinRT project:

```
namespace MyComponent
{
    public sealed class MyClass
    {
        public int DoSomething(int x)
        {
            return x + 42;
        }

    }
}
```

If you add a reference to your WinRT Component project in your app project, you can use the MyClass class from the C# project in the following way from your JavaScript code:

```
    var myClass = new MyComponent.MyClass();
    var value = myClass.doSomething(42);
```

This code calls in to the C# code and executes a method. The function name starts with a lowercase “d” in the JavaScript code. C# has the convention that element names should start with an uppercase character; JavaScript follows a convention in which each element starts with a lowercase character. This is why a method starting with an uppercase character in C# starts with a lowercase letter in JavaScript.

EXAM TIP

Remember that you can mix JavaScript and C# or C++ code when building your app.

There are a couple of restrictions when you use WinRT Components:

- The fields, parameters, and return values of all the public types and members in your component must be WinRT types.
- Public classes and interfaces can contain methods, properties, and events. A public class or interface can’t do the following, however:
  - Be generic
  - Implement an interface that is not a WinRT interface
  - Derive from types that are not inside the WinRT
- Public classes must be sealed.
- Public structures can have only public fields as members, which must be value types or strings.
- All public types must have a root namespace that matches the assembly name and does not start with Windows.

MORE INFO **WINMD COMPONENTS**

For more info on creating WinMD Components, see the official documentation at *http://msdn.microsoft.com/library/windows/apps/hh779077.aspx*.

Thought experiment

Designing a large application

In this thought experiment, apply what you've learned about this objective. You can find answers to these questions in the "Answers" section at the end of this chapter.

Your company is starting a new app project, and you are asked to sketch the initial architecture. You have read about the advantages of using a layered application, but a colleague argues against that architecture.

How should you react to his following statements?

1. A layered application complicates development.
2. Layering slows down development because developers have to wait for a layer to be completed before they can continue.
3. JavaScript can't be used to build a layered application.

Objective summary

- Dividing your application into distinct layers helps you create maintainable applications that are easier to extend.
- Typical layers are the UI, business, and data layers.
- When working with JavaScript, pay attention to how you structure your code to avoid some of the inherent JavaScript problems such as global state. You can use self-invoking anonymous functions to apply some scoping to your code.
- TypeScript is a superset of JavaScript that helps you write application-scale JavaScript.
- WinMD Components can be written in other languages such as C++ and C#, and can then be used from JavaScript Windows Store apps.

Objective review

Answer the following questions to test your knowledge of the information in this objective. 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 are designing an app that connects to an external web service to load data for the app. This data is then processed and displayed on the screen. From which layer should the web service be called?
   - **A.** Data layer
   - **B.** Service layer
   - **C.** Business layer
   - **D.** UI layer
2. Should you avoid global state in JavaScript applications?
   - **A.** No. Global state is easy because you can share data between different parts of your app.
   - **B.** No. It's not possible to avoid global state in JavaScript.
   - **C.** Yes. Avoiding global state keeps your code free from unwanted side effects and aids maintainability.
   - **D.** Yes. Global state is not possible in Windows Store apps.
3. Which elements can you use to build a layered application? (Choose all that apply.)
   - **A.** WinJS.Class.define
   - **B.** WinJS.Namespace.define
   - **C.** WinMD Components
   - **D.** Web services

Objective 1.3: Design and implement Process Lifetime Management (PLM)

When working with regular desktop applications, you are used to launching and closing them yourself. When you switch to another application, other running applications stay in memory, and you can easily switch back to them. When your computer starts running slowly, you start closing applications and maybe even open Task Manager to check what's happening.

Windows Store apps behave differently. Microsoft doesn't want users to bother with actively closing applications, so it created *Process Lifetime Management (PLM)* for Windows Store apps that manages the lifetime of an app without any user intervention.

The life cycle of your app is the foundation on which you build. Make sure that you get it right. You can have a beautiful app, but when it doesn't behave as users expect, you will lose them.

This objective covers how to:

- Choose a state management strategy
- Handle the suspend event (oncheckpoint)
- Prepare for app termination
- Handle the onactivated event
- Check the ActivationKind and previous state

Choosing a state management strategy

Windows Store apps can be launched and terminated in a couple of different ways. Understanding the application life cycle and anticipating it in your app lead to a better user experience in which your app naturally behaves as a user would expect.

Apps start their lives in the not running state. You launch the app by clicking the tile on the Start screen; your app then displays its splash screen, loads data, and begins running.

That's the easy track. In reality, however, a lot more can happen. Figure 1-10 shows the typical life cycle of an app.

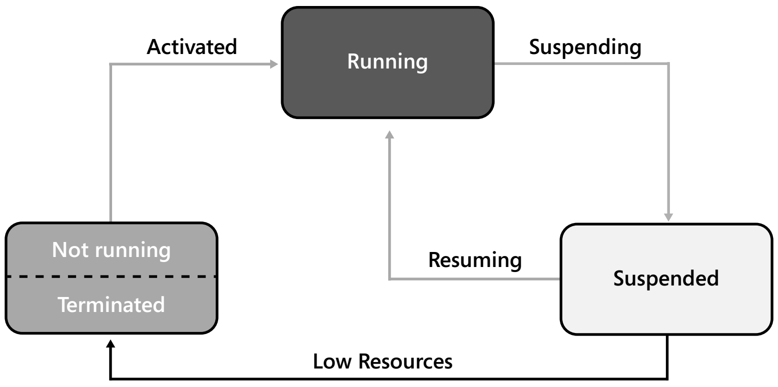


FIGURE 1-10 The life cycle of an app

When your app is in the *suspended state*, it consumes less memory than it consumes in the running state and it doesn't get scheduled for CPU time, which saves power to enable longer battery times on tablets and laptops. Although Windows tries to keep as many apps as possible in the suspended state, when the operating system is running low on resources (typically memory), Windows starts terminating apps that haven't been used for some time.

Choosing your state management strategy comes down to understanding the life cycle of your app and responding appropriately. What does this mean? When users (game or blog readers, for example) leave your app and then return to it, they expect to come back at the same point with the same settings as when they left.

In the meantime, maybe Windows suspended your app or even terminated it. However, the users don't know the situation and want to continue working with your app. So you have to respond to the events such as suspension or resumption and make sure that you save the correct state and restore it whenever necessary so users don't notice anything.

You can take this process one step farther. Because apps can be installed on multiple devices, you should accommodate a user switching between those devices when using your app. You need to save all the details of the users' actions to an external source and reload them whenever an app launches on a device. Windows helps you by automatically *roaming* data to all user devices so that you can share state across devices and provide a seamless experience.

Handling the onactivated event

Your app can be activated in a variety of ways. The most obvious one, of course, is a user directly launching it by clicking the app tile on the Start screen. There are also many more ways to activate apps. If you use toast notifications (which are discussed in more detail in

Chapter 4, “Program user interaction”), a user can launch your app by clicking a notification. If you are implementing contracts (see Chapter 2, “Develop Windows Store apps,” for more details), a user can launch your app with the Search or Share charms, by file type, or with URI associations.

These activation events require a different strategy. Fortunately, the Visual Studio templates give you some boilerplate code that you can use to react to those events. After you create a new app from the Blank App template, you see the following code in the default.js file:

```
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());
    }
};
```

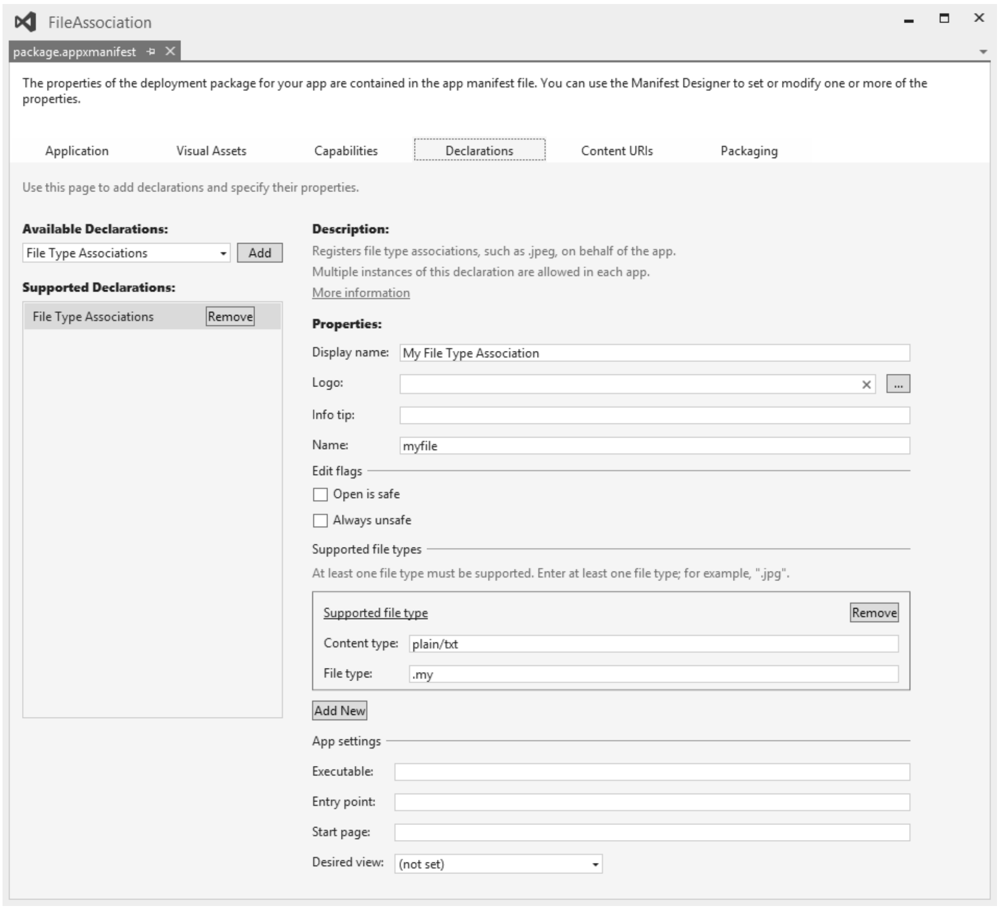
The code shows how to subscribe to the onactivated event of your WinJS application. Inside the event handler, you can see whether your app is newly launched or you are resuming from a suspended state.

This state affects the steps you need to take. If the app is newly launched, initialize the app and show its home screen to users. If users return to your app, make sure that they return to the exact same point.

If the UI content has changed since the app was suspended, you need to load the new data and update your UI accordingly. Your app’s activated event is running while Windows shows the splash screen, which is why you should make sure that your initialization is as fast as possible.

Your app can also be associated with a certain file type or a URI. Associating your app with a file type is configured in the application manifest.

Figure 1-11 shows the Manifest Designer with the File Type Association configured for files that have an extension of .my.



The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Use this page to add declarations and specify their properties.

Available Declarations: File Type Associations

Supported Declarations: File Type Associations

Description:
Registers file type associations, such as .jpeg, on behalf of the app.
Multiple instances of this declaration are allowed in each app.

Properties:

Display name: My File Type Association
Logo:
Info tip:
Name: myfile

Edit flags
☐ Open is safe
☐ Always unsafe

Supported file types
At least one file type must be supported. Enter at least one file type; for example, ".jpg".

Supported file type
Content type: plain/txt
File type: .my

App settings
Executable:
Entry point:
Start page:
Desired view: (not set)

FIGURE 1-11 The Manifest Designer dialog box showing the File Type Associations

After configuring these settings, launch the app from Visual Studio to register your new file type with Windows. You can then create a new text file and change the extension to .my. Double-click the new file to launch your app.

During the activated event of your app, you can see whether the app is launched from an associated file:

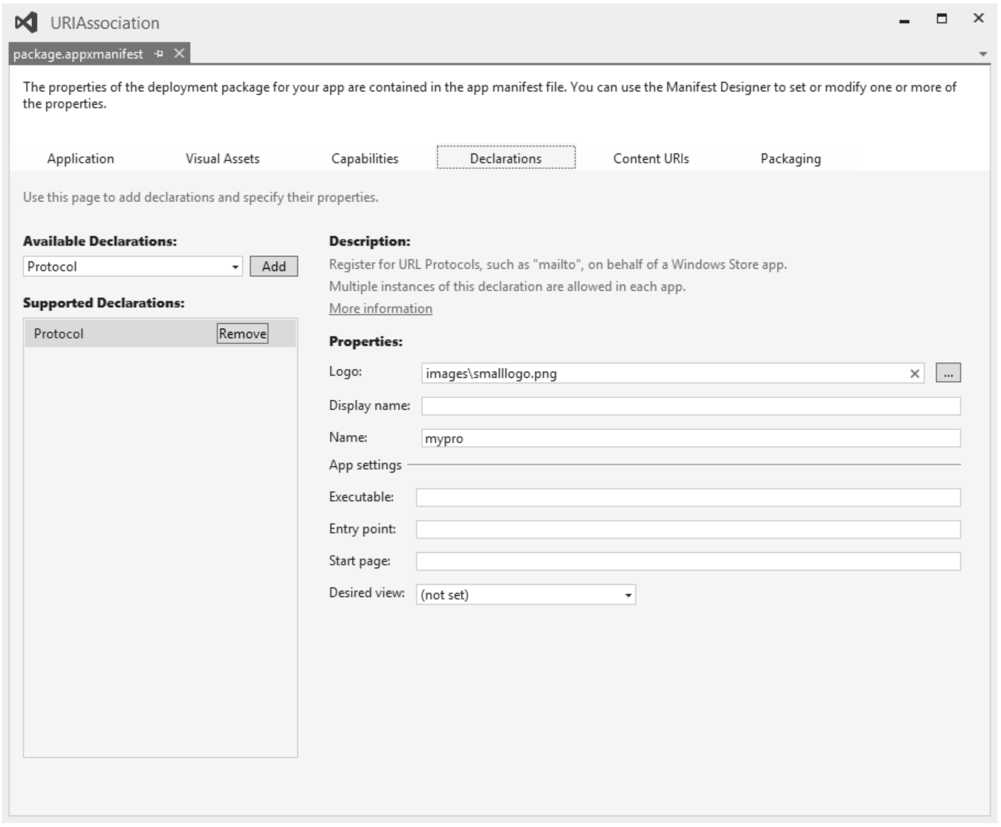
```
if (args.detail.kind === Windows.ApplicationModel.Activation.ActivationKind.file) {
    var file = args.detail.files[0];
    Windows.Storage.FileIO.readTextAsync(file).then(function (text) {
    });

    // The number of files received is eventArgs.detail.files.size
    // The first file is eventArgs.detail.files[0].name
}
```

This code checks to see whether ActivationKind is of type file. If so, the arguments passed to your activated event handler contain a details.files property that contains information about the file or files that a user selected when launching your app. In this example, you are dealing with a plain text file, so you can pass it to Windows.Storage.FileIO.readTextAsync and read the text content of the file.

Your app can also be activated from a URI. One example is the Windows Store. By navigating to a URI of the form ms-windows-store:PDP?PFN=, you launch the Windows Store and navigate to the specified Package Family Name. The ms-windows-store part of the URI is called the *protocol*.

You can add your own protocols to the app to associate it with specific URIs. Figure 1-12 shows the Manifest Designer with a newly added protocol of mypro.


URIAssociation

package.appxmanifest

The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Application | Visual Assets | Capabilities | Declarations | Content URIs | Packaging

Use this page to add declarations and specify their properties.

Available Declarations: Protocol [Add]

Supported Declarations: Protocol [Remove]

Description:
Register for URL Protocols, such as "mailto", on behalf of a Windows Store app.
Multiple instances of this declaration are allowed in each app.
More information

Properties:
Logo: images\smalllogo.png
Display name:
Name: mypro
App settings
Executable:
Entry point:
Start page:
Desired view: (not set)

FIGURE 1-12 The Manifest designer dialog box showing the Protocol declaration

Of course, it’s important to configure a logo and descriptive display name, after which you can launch the app to register your protocol with Windows. Opening Windows Explorer and navigating to mypro://content launches your app.

Just as with File Type Associations, you can see whether the app is launched from a URI in your activated event:

```
if (args.detail.kind === activation.ActivationKind.protocol) {
    var uri = args.detail.uri;
    var rawUri = uri.rawUri;
}
```

The args.detail.uri property contains information about the URI that launched your app. It is up to you to parse the URI and take appropriate action.

Remember that both the files and URIs that launch your app can be harmful. You should never trust the input a user gives you and always use security measures when dealing with external input.

Handling the suspend event (oncheckpoint)

When a user switches to another app, Windows suspends your app after a couple of seconds, which enables the user to immediately switch back to your app without it having to do any work.

Whenever Windows notices that the user isn't coming back right away, your app is suspended. Windows then raises the checkpoint event. In this event, you can save any user state that you want to restore when the app would be resumed from termination.

The syntax of subscribing to the checkpoint event is as follows:

```
    app.oncheckpoint = function (args) {
    };
```

Inside the function, you can save any state or other data that you want to restore when the app moves from terminated to running in the WinJS.Application.sessionState object. The content of this object is serialized to your local appdata folder. When the app is activated again from the terminated state, the sessionState object is rehydrated from your local appdata folder. You can then use the data inside the sessionState object to reinitialize your app.

This process can be as easy, as the following example shows:

```
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 {
            var value = WinJS.Application.sessionState.value;
        }
        args.setPromise(WinJS.UI.processAll());
    }
};

app.oncheckpoint = function (args) {
    WinJS.Application.sessionState.value = 42;
};
```

When your app goes into suspension, a value of 42 is saved inside your sessionState object. When the app launches from a terminated state, the value is retrieved from the object.

You can also use the WinJS.Application object directly to write and read state from the local, temp, or roaming folders. Writing directly to those folders can be useful if your data can't be directly serialized to a string or if you want specific control over the location of your data.

When using any asynchronous actions inside a checkpoint event, you have to signal it to the operating system. Windows assumes that you saved all your state when the checkpoint events returns, so it doesn't give your app any CPU time. To avoid losing CPU time with asynchronous operations, you can use the args.setPromise() method, as in the activated event.

Remember that you never get more than five seconds. If you don't return from the checkpoint method or finish all your asynchronous operations within five seconds, your app is terminated.

Preparing for app termination

When your app goes from running to suspended, you receive a notification from the Windows operating system. But when your app goes from suspended to terminated, your app doesn't receive a notification. This is by design and is actually quite logical.

Your app is terminated because the operating system is low on resources. Activating your app only to prepare itself for termination could become troublesome because the sole act of activating the app uses resources. And when your app tries to save some state to disk or call web services, even more memory is used.

To save resources, your app doesn't get called when your app terminates. Instead, you should do all your work in the checkpoint event discussed in the previous section. Then whenever your app goes from suspended to terminated, you have saved all the required state.

EXAM TIP

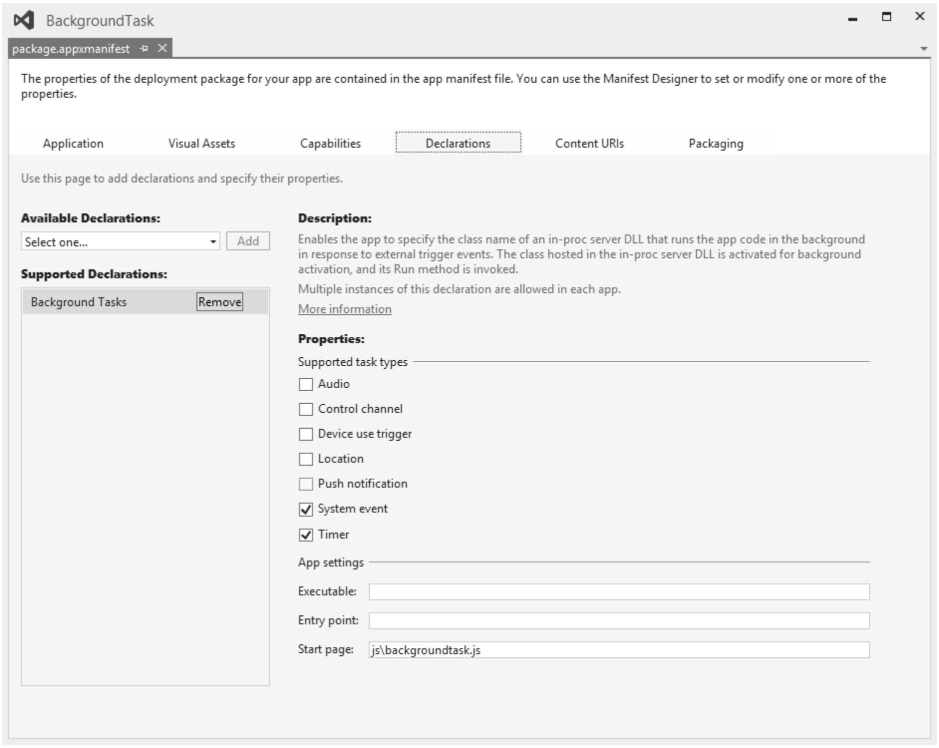
Remember that there is no separate event for termination. You should save all state in the checkpoint event and make sure that your app can completely recover from termination.

Using background tasks

But what if you want to keep running when the user closes your app? You can do so by using background tasks. You can request Windows to grant permission to execute code in the background by using the application Manifest Designer.

Figure 1-13 shows the application Manifest Designer with a BackgroundTask extension.

The background task is configured to trigger on a system event and on a timer. When the Background Task is triggered, it launches the JavaScript file js\backgroundtask.js. Your background task consists of two parts: the actual task and the code to register your task.



The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Use this page to add declarations and specify their properties.

Available Declarations: Select one...

Supported Declarations: Background Tasks

Description:
Enables the app to specify the class name of an in-proc server DLL that runs the app code in the background in response to external trigger events. The class hosted in the in-proc server DLL is activated for background activation, and its Run method is invoked.
Multiple instances of this declaration are allowed in each app.

Properties:
Supported task types
- [ ] Audio
- [ ] Control channel
- [ ] Device use trigger
- [ ] Location
- [ ] Push notification
- [x] System event
- [x] Timer

App settings
Executable:
Entry point:
Start page: js\backgroundtask.js

FIGURE 1-13 The Manifest Designer dialog box showing the Background Tasks declaration

Begin with registration. The following method lets you register a background task:

```
function registerTask(taskEntryPoint, taskName, trigger, condition) {

    var builder = new Windows.ApplicationModel.Background.BackgroundTaskBuilder();

    builder.name = taskName;
    builder.taskEntryPoint = taskEntryPoint;
    builder.setTrigger(trigger);

    if (condition !== null) {
        builder.addCondition(condition);
        builder.cancelOnConditionLoss = true;
    }

    var task = builder.register();
    task.addEventListener("progress", new BackgroundTaskSample.progressHandler(task).
onProgress);
    task.addEventListener("completed", new BackgroundTaskSample.completeHandler(task).
onCompleted)

    var settings = Windows.Storage.ApplicationData.current.localSettings;
    settings.values.remove(taskName);
};
```

This method takes a parameter that points to your JavaScript file that contains the actual task, a name, the trigger you want to use, and a condition that determines whether the task should run. You can call the method like this:

```
registerTask("js\\backgroundtask.js",
                            "SampleJavaScriptBackgroundTask",
                            new Windows.ApplicationModel.Background.SystemTrigger(
                                      Windows.ApplicationModel.Background.
SystemTriggerType.timeZoneChange, false),
                            null);
```

This code registers a background task that runs whenever users change their time zone without any other conditions.

Triggers can be any of the following:

- **SmsReceived** The background task is triggered when a new Short Message Service (SMS) message is received by an installed mobile broadband device.
- **UserPresent** The background task is triggered when the user becomes present.
- **UserAway** The background task is triggered when the user becomes absent.
- **NetworkStateChange** The background task is triggered when a network change occurs, such as a change in cost or connectivity.
- **ControlChannelReset** The background task is triggered when a control channel is reset.
- **InternetAvailable** The background task is triggered when the Internet becomes available.
- **SessionConnected** The background task is triggered when the session is connected.
- **ServicingComplete** The background task is triggered when the system has finished updating an app.
- **LockScreenApplicationAdded** The background task is triggered when a tile is added to the lock screen.
- **LockScreenApplicationRemoved** The background task is triggered when a tile is removed from the lock screen.
- **TimeZoneChange** The background task is triggered when the time zone changes on the device (for example, when the system adjusts the clock for daylight savings time [DST]).

- **OnlineIdConnectedStateChange** The background task is triggered when the Microsoft account connected to the account changes.
- **BackgroundWorkCostChange** The background task is triggered when the cost of background work changes.

Remember that for triggers such as user presence and others, your app must also be visible on the lock screen.

If you are not interested in every trigger change, you can add additional conditions to your background task:

- **UserPresent** Specifies that the background task can run only when the user is present. If a background task with the UserPresent condition is triggered and the user is away, the task doesn't run until the user is present.
- **UserNotPresent** Specifies that the background task can run only when the user is not present. If a background task with the UserNotPresent condition is triggered and the user is present, the task doesn't run until the user becomes inactive.
- **InternetAvailable** Specifies that the background task can run only when the Internet is available. If a background task with the InternetAvailable condition is triggered and the Internet is not available, the task doesn't run until the Internet is available again.
- **InternetNotAvailable** Specifies that the background task can run only when the Internet is not available. If a background task with the InternetNotAvailable condition is triggered, and the Internet is available, the task doesn't run until the Internet is unavailable.
- **SessionConnected** Specifies that the background task can run only when the user's session is connected. If a background task with the SessionConnected condition is triggered and the user session is not logged on, the task runs when the user logs on.
- **SessionDisconnected** Specifies that the background task can run only when the user's session is disconnected. If a background task with the SessionDisconnected condition is triggered and the user is logged on, the task runs when the user logs off.
- **FreeNetworkAvailable** Specifies that the background task can run only when a free (nonmetered) network connection is available.
- **BackgroundWorkCostNotHigh** Specifies that the background task can run only when the cost to do background work is low.

After configuring the triggers and conditions, the only thing you need is the actual task. In the previous example, you pointed to a specific JavaScript file: js/backgroundtask.js.

A simple background task can look like this:

```
(function () {
    "use strict";

    var cancel = false,
        progress = 0,
        backgroundTaskInstance = Windows.UI.WebUI.WebUIBackgroundTaskInstance.current,
        cancelReason = "";

    function onCanceled(cancelEventArg) {
        cancel = true;
        cancelReason = cancelEventArg.type;
    }
    backgroundTaskInstance.addEventListener("canceled", onCanceled);

    function onTimer() {
        var key = null,
            settings = Windows.Storage.ApplicationData.current.localSettings,
            value = null;

        if ((!cancel) && (progress < 100)) {
            setTimeout(onTimer, 1000);
            progress += 10;
            backgroundTaskInstance.progress = progress;
        } else {
            backgroundTaskInstance.succeeded = (progress === 100);
            value = backgroundTaskInstance.succeeded ? "Completed" : "Canceled with
reason: " + cancelReason;

            key = backgroundTaskInstance.task.name;
            settings.values[key] = value;

            close();
        }
    }
    setTimeout(onTimer, 1000);
})();
```

This code is a self-enclosing function that contains the task, which consists of the onTimer method that does the actual work. It also has a cancel event handler to see whether the task should be canceled.

The Windows.UI.WebUI.WebUIBackgroundTaskInstance.current property gives you access to the background task framework of Windows. You can check for cancellation and signal success or failure by using this object.

The call to close at the end of your task is required to signal that your task is done.

Background tasks are not meant to be used for long-running tasks. They should be used to respond to changes in the environment and run short tasks on a timer.

Checking the ActivationKind and previous state

For the exam, make sure that you understand the reasoning behind the ActivationKind enumeration and the value for the previous state of your app.

When you look at the activated event, you see both the kind and the previous state used:

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
        if (args.detail.previousExecutionState !==
                    activation.ApplicationExecutionState.terminated) {
        } else {
        }
        args.setPromise(WinJS.UI.processAll());
    }
};
```

ActivationKind can have a lot of different values:

- **Launch** The user launched the app or tapped a content tile.
- **Search** The user wants to search with the app.
- **ShareTarget** The app is activated as a target for share operations.
- **File** An app launched a file whose file type is registered to be handled by this app.
- **Protocol** An app launched a URL whose protocol is registered to be handled by this app.
- **FileOpenPicker** The user wants to pick files provided by the app.
- **FileSavePicker** The user wants to save a file and selects the app as the location.
- **CachedFileUpdater** The user wants to save a file for which the app provides content management.
- **ContactPicker** The user wants to pick contacts.
- **Device** The app handles AutoPlay.
- **PrintTaskSettings** The app handles print tasks.
- **CameraSettings** The app captures photos or video from an attached camera.
- **RestrictedLaunch** The user launched the restricted app.
- **AppointmentsProvider** The user wants to manage appointments provided by the app.
- **Contact** The user wants to handle calls or messages for the phone number of a contact provided by the app.
- **LockScreenCall** The app launches a call from the lock screen. If the user wants to accept the call, the app displays its call UI directly on the lock screen without requiring the user to unlock. A lock screen call is a special type of launch activation.

Most values come from integrating with the operating system. When you start implementing contracts, your app can be activated in a lot of different scenarios that you need to handle. Chapter 2 provides more detail on implementing contracts.

If the user explicitly closed your app, you can assume that there was some kind of error that the user wanted to correct. Restoring the state to the point where the user closed your app doesn't help. Instead, you should do a clean initialize of your app.

You can use the args.detail.previousExecutionState property to check the previous state of the app. It can be one of the following values:

- **NotRunning** The app is not running.
- **Running** The app is running.
- **Suspended** The app is suspended.
- **Terminated** The app was terminated after being suspended.
- **ClosedByUser** The app was closed by the user.

A previous state of NotRunning, which is the most common one, occurs whenever a user launches your app for the first time. It can happen after installing the app, but also after a computer reboot or when switching accounts.

A previous state of Running means that your app is already running, but one of its contracts or extensions is activated.

Suspended happens whenever Windows kept your app in memory but didn't assign any CPU to it. When this happens, you might want to update any on-screen content to make sure everything is up to date.

Terminated is the state you learned about in the previous sections. Whenever Windows determines that your app should be removed from memory, it terminates your app. When resuming from a terminated state, you need to reload all state, which can be done from the sessionState object or from an external web service (when you want to make sure that everything is up to date).

ClosedByUser has a different behavior. Whenever the user forcefully closes your app (through Alt+F4 or the close gesture) and returns within 10 seconds, you do a clean startup because Windows assumes that there was an error, and the user restarts the app. When the user takes longer to return, you need to restore state so the user can continue.

Thought experiment

Strengthening your foundation

In this thought experiment, apply what you've learned about this objective. You can find answers to these questions in the "Answers" section at the end of this chapter.

You have developed an app sharing task lists with other users. However, your first version is running, and users are reporting problems. What can you do to solve the following problems?

1. **Users want to create a task list on their main PC and then view it on their phone. However, the changes never sync between devices, making it impossible.**
2. **After coming back to their app, changes are lost, and they have to start over.**
3. **Users report that after launching their app, they sometimes see data that's out of date.**

Objective summary

- Windows Store apps go through a life cycle in which an app can be not running, running, suspended, or terminated.
- The activated event is important for initializing your app for users.
- The checkpoint event allows you to save any user state and data before your app gets suspended and possibly terminated.
- When your app gets activated, it is important to know the reason why your app is activated and its previous state.

Objective review

Answer the following questions to test your knowledge of the information in this objective. 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 are creating a game as a Windows Store app that features real-time action, and you are thinking about state management. Which of the following statements is true about state management? (Choose all that apply.)
   - **A.** For a game, you don't have to consider state management.
   - **B.** You need to implement the activated event.
   - **C.** You need to implement the checkpoint event.
   - **D.** You need to implement the terminated event.

2. A user closes your Windows 8.1 app by pressing Alt+F4. What should you do when the user returns the following day?
   - **A.** Do a fresh start of the app because the user forcefully closed the app.
   - **B.** Reload all user state and continue as if the user never left.
   - **C.** Show a dialog box that asks whether the user wants to continue or start over.
   - **D.** Restart the application behind the scenes to force a clean start.
3. You want to restore any saved state when the app resumes. Which event do you use?
   - **A.** Ready
   - **B.** Loaded
   - **C.** Checkpoint
   - **D.** Activated

Objective 1.4: Plan for an application deployment

No matter how good your app is, users won't use it if you don't deploy it. Planning your deployment is an important part of developing your application. Microsoft decided that not all apps are allowed in to the Windows Store. Knowing these requirements and understanding how to configure your app for deployment is the topic of this objective.

You also learn how to deploy an app for your enterprise when you don't use the public Windows Store.

This objective covers how to:

- Plan a deployment based on Windows 8 application certification requirements
- Prepare an app manifest (capabilities and declarations)
- Sign an app
- Plan the requirements for an enterprise deployment

Planning a deployment based on Windows 8 application certification requirements

To publish your app to the Windows Store, the first step is to acquire a developer account. The Express editions of Visual Studio require you to purchase such an account. MSDN subscribers already have an account that they can use for free.

After you have an account, register your new app with the Windows Store to reserve the name that you want to use. This name is then reserved for you for one year. After one year, the name will be free for other developers to use.

During this process, you also have to configure the way your app will be sold. Is it free? Are you using a trial? Or maybe in-app purchases? Answering these questions forces you to think about those steps before you start your app. You can plan your app around the business model that you want to support and make sure that your app fully supports it.

When you finish your app, you can submit it to the Windows Store. If you haven't done so, you can now supply the information on your app name, selling details, and services such as in-app purchases or trial support.

You also need a rating for your app, which can be an age rating (such as 3+ or 16+) or a rating board (such as ESRB or PEGI). Think carefully about this rating; when in doubt, choose the strictest one. Especially if your app uses some public Internet service (such as Twitter), you need to use a rating of at least 12+ or even 16+ because you never know what shows up on users' screens.

If your app uses some form of cryptography, you need to mention it. Other very important parts are the description, feature list, and screen shots of your app. Make sure to consider these options carefully. This information will show up in the Windows Store and it should convince a user to install your app.

Another important step is to ensure that the tester can fully use your app. If your app requires a web service to be available, make sure that the web service is running when your app goes through submission. If the tester needs to log on to your app, supply a demo account that gives the tester access to all features of your app.

Creating your app package

The most important part of your submission is the actual application data. Your app is submitted in what's called an *app package*.

The easiest way to create an app package is with Visual Studio. If you are running the Express edition of Visual Studio, you have a Store menu with an option to Create App Packages. If you run Visual Studio Professional or higher, you can find the Store menu as a submenu of Project.

The Create App Package allows you to build a package and upload it to the Windows Store or to create it locally.

The package that gets created is an .appx file, which is a zip file. You can open it in Windows Explorer and check out the files in it after changing the extension to .zip. The package contains everything that's required for your app, such as JavaScript, CSS, HTML, and images. It also contains some metadata in the form of a manifest (which you will learn about in the next section), a signature, and a block map.

The block map is a description of the data in your package, split into distinct blocks of data. By splitting your package into separate parts, the Windows Store can download only the parts that have changed when an update of your app is released. Downloading only the updated parts saves users a lot of bandwidth, which is becoming more and more important with mobile devices.

Certification requirements

After creating your package and submitting all the required information, you can submit your app to the Windows Store. Your app then goes through a series of tests to check your app thoroughly before it is allowed or disallowed from the Windows Store.

Microsoft released an official document (see the following More Info box) that outlines all certification requirements for the Windows Store. This document is frequently updated, so become familiar with it before publishing your app.

MORE INFO **APP CERTIFICATION REQUIREMENTS FOR THE WINDOWS STORE**

The complete description of app certification requirements can be found at *http://msdn.microsoft.com/en-us/library/windows/apps/hh694083.aspx*.

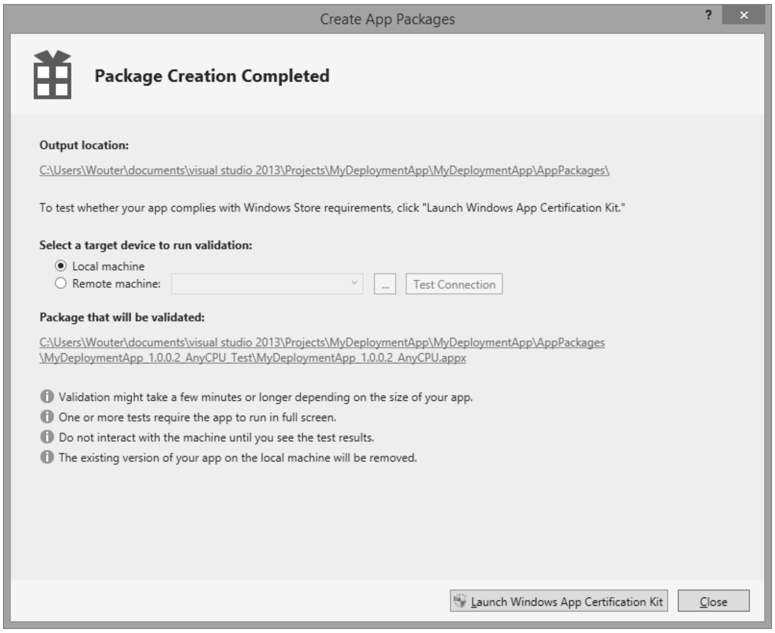
Some steps in this document are obvious. You shouldn't submit an app that doesn't work, doesn't add any value, or is not branded. You are not allowed to hack the system. Trying to communicate with other apps or loading remote scripts is forbidden. Be careful with privacy-related data by giving the user some consent options.

Windows App Certification Kit

The Windows App Certification Kit helps you test your app in an automated way similar to the way Microsoft will test your app upon submission. The easiest way to run the Windows App Certification Kit is to create your app package from Visual Studio. After the package is created, Visual Studio asks you whether you want to start the validation process (see Figure 1-14).

MORE INFO **COMPLETE LIST OF TESTS**

For a complete description of all the tests run by the Windows App Certification Kit and how to troubleshoot failures, see *http://msdn.microsoft.com/en-us/library/windows/apps/jj657973.aspx*.



Package Creation Completed

Output location:

C:\Users\Wouter\documents\visual studio 2013\Projects\MyDeploymentApp\MyDeploymentApp\AppPackages\

To test whether your app complies with Windows Store requirements, click "Launch Windows App Certification Kit."

Select a target device to run validation:

[✓ Local machine]
Remote machine:

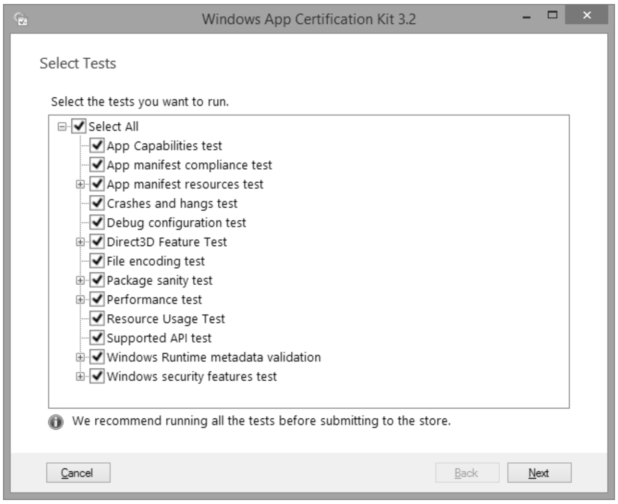
Package that will be validated:

C:\Users\Wouter\documents\visual studio 2013\Projects\MyDeploymentApp\MyDeploymentApp\AppPackages
\MyDeploymentApp_1.0.0.2_AnyCPU_Test\MyDeploymentApp_1.0.0.2_AnyCPU.appx

- Validation might take a few minutes or longer depending on the size of your app.
- One or more tests require the app to run in full screen.
- Do not interact with the machine until you see the test results.
- The existing version of your app on the local machine will be removed.

FIGURE 1-14 The Create App Packages dialog box

After starting the Windows App Certification Kit, some information is collected; then you see the dialog box shown in Figure 1-15. The Windows App Certification Kit runs many tests for you.



Select Tests

Select the tests you want to run.

- ✓ Select All
  - ✓ App Capabilities test
  - ✓ App manifest compliance test
  - ✓ App manifest resources test
  - ✓ Crashes and hangs test
  - ✓ Debug configuration test
  - ✓ Direct3D Feature Test
  - ✓ File encoding test
  - ✓ Package sanity test
  - ✓ Performance test
  - ✓ Resource Usage Test
  - ✓ Supported API test
  - ✓ Windows Runtime metadata validation
  - ✓ Windows security features test

We recommend running all the tests before submitting to the store.

FIGURE 1-15 The Windows App Certification Kit dialog box

Additional certification requirements

You shouldn’t depend solely on the results of the Windows App Certification Kit; you should also perform extensive manual testing of your app.

For example, you should test your app on multiple platforms. Maybe your app works great on your local PC, but that doesn’t mean it will work on a Microsoft Surface tablet or other less-resource-intensive systems. In some countries/regions, Microsoft helps by having special app development days during which you can come in and test your app on a series of devices. Microsoft also advises you about any issues your app may have.

MORE INFO **TEST CASES**

MSDN provides a list of possible test cases that you can run against your app at *http://msdn.microsoft.com/en-us/library/windows/apps/dn275879.aspx*.

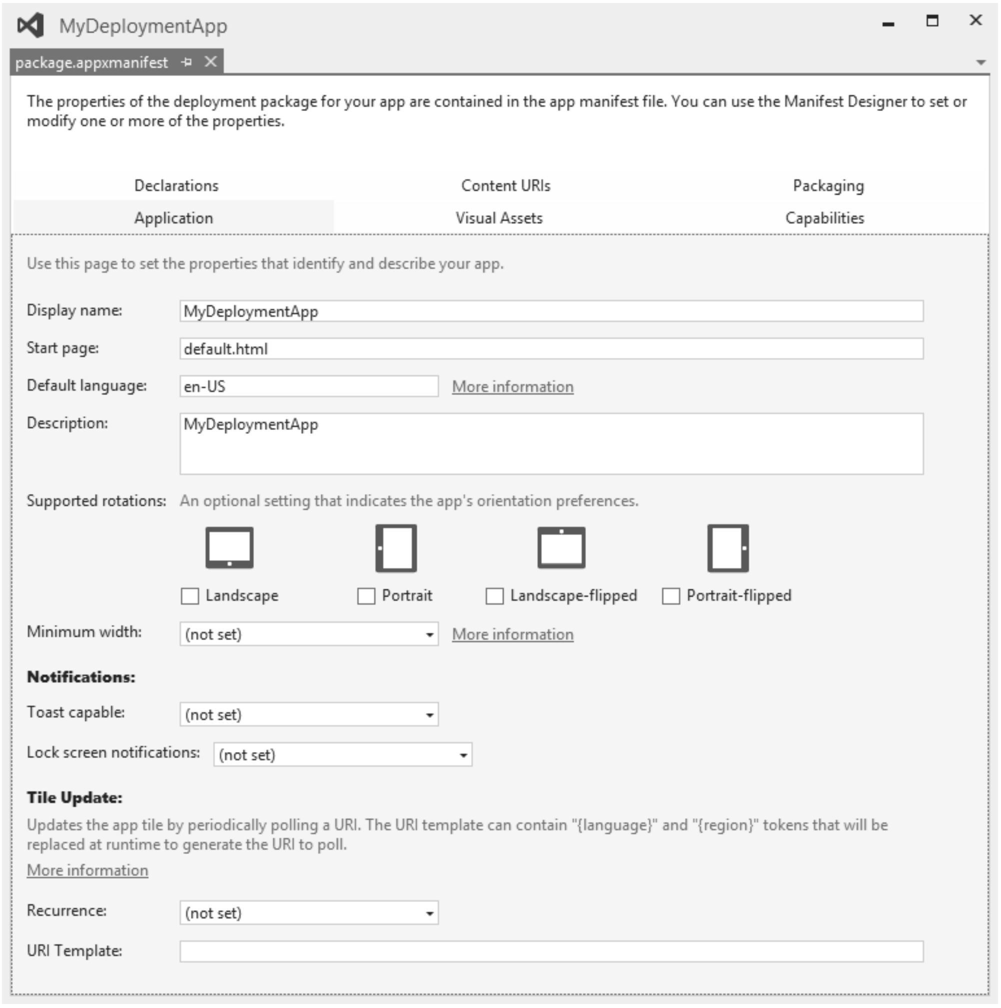
Preparing an app manifest (capabilities and declarations)

When you create a new Windows Store app, Visual Studio adds a file called package.apxxmanifest to your project.

This file is called an application manifest. When you open the manifest, the Manifest Designer loads (see Figure 1-16).

The Manifest Designer includes pages that you can use to configure your app:

- **Application** Allows you to set some application-wide settings such as a start page and supported rotations. You can also configure notifications and tile update settings.
- **Visual Assets** Allows you to configure different resolutions for images that are shown in the Windows Store. You can also configure tile images and your splash screen.
- **Capabilities** Specifies the features or devices that your app can use on the user’s system.
- **Declarations** You configure how your app integrates with Windows and other apps.
- **Content URIs** Your app can load web pages into an iframe. Those pages are normally restricted and have limited access to the user’s system. Here you can specify URIs that can access geolocation devices and the Clipboard, and can send script notifications to your app.
- **Packaging** Allows you to configure the display name of your package, version, and publisher information.



The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Use this page to set the properties that identify and describe your app.

Display name:	MyDeploymentApp
Start page:	default.html
Default language:	en-US
Description:	MyDeploymentApp
Supported rotations:	An optional setting that indicates the app's orientation preferences. ☐ Landscape ☐ Portrait ☐ Landscape-flipped ☐ Portrait-flipped
Minimum width:	(not set)

Notifications:

Toast capable:	(not set)
Lock screen notifications:	(not set)

Tile Update:

Updates the app tile by periodically polling a URI. The URI template can contain "{language}" and "{region}" tokens that will be replaced at runtime to generate the URI to poll.

More information

Recurrence:	(not set)
URI Template:	

FIGURE 1-16 The Manifest Designer, showing the Application page

The exam requires you to understand all elements of the manifest, but it pays special attention to the capabilities and declaration settings.

Why are those settings so important? They configure what your app is allowed to do on a user’s system. Instead of allowing every integration by default, Windows Store apps need to explicitly ask for permission. Users can see the list of required permissions when installing an app from the Windows Store and can decide whether they trust your app enough to allow those permissions.

For example, it would be strange if your RSS Reader app needed access to your webcam. It is a lot more likely for a chat application that features video chat. You can ask permission for the following capabilities:

- **Enterprise Authentication** Typically not needed for an app; it allows your app to connect to resources on an intranet that requires domain authentication.
- **Internet (Client)** Requests Internet action over public networks. This option is enabled by default and should be disabled if your app does not require the Internet.
- **Internet (Client & Server)** Allows both inbound and outbound connections.
- **Location** Requests access to the current location (from a GPS sensor or from network information).
- **Microphone** Requests access to the microphone's audio feed.
- **Music Library** Requests access to the music library to add, change, or delete files.
- **Pictures Library** Requests access to the pictures library to add, change, or delete files.
- **Private Network (Client & Server)** Requests access to inbound and outbound network access for a user's trusted places (such as home and work devices that are on the same network).
- **Proximity** Requests the capability to connect to other devices through Wi-Fi Direct or near field proximity radio.
- **Removable Storage** Requests access to removable storage devices. Allows you to add, change, or delete file types that you have defined in the Declarations page.
- **Shared User Certificates** Gives you access to application programming interfaces (APIs) for requesting the user to authenticate through a security card, certificate, and so on.
- **Videos Library** Requests access to the videos library to add, change, or delete files.
- **Webcam** Requests access to the webcam's video feed so you can take snapshots or movies.

Although you should never select more capabilities than are strictly required, trying to access a restricted area of the system without the required capability results in an error.

Next to capabilities, you also need to configure your declarations, which are required to support contracts and extensions. (A *contract* defines an agreement between apps; an *extension* is an agreement between your app and Windows.)

MORE INFO **CONTRACTS AND EXTENSIONS**

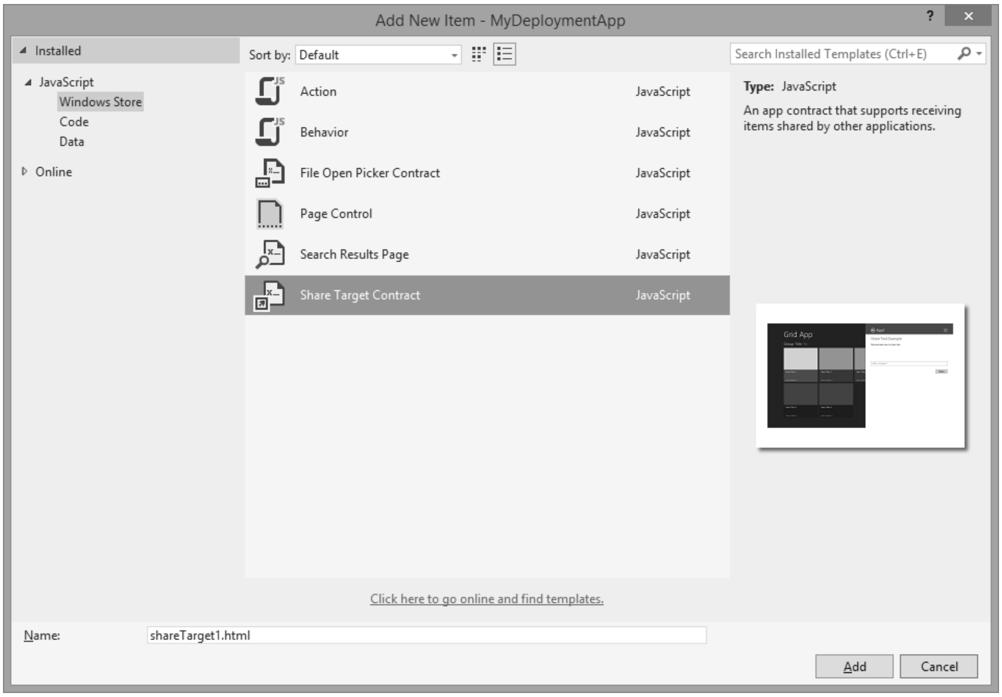
Contracts and extensions are discussed in more detail in Chapter 2.

The declarations that you can configure are the following:

- **Account Picture Provider** Allows users to use your app to change their account pictures.
- **AutoPlay** Allows users to choose your app in the auto play dialog box.
- **Background Tasks** Apps can use background tasks to run app code even when the app is suspended. Background tasks are intended for small work items that require no interaction with the user.
- **Cached File Updater** If your app caches file is on local disk, you can subscribe to events such as the user opening the file (so you can check to see whether there is a newer version) or to download newer versions of a file as soon as they are available. OneDrive (formerly known as SkyDrive) is a good example of this kind of behavior.
- **Camera Settings** Allows you to customize the flyout that displays camera options.
- **Contact Picker** Enables your app to show up in the list of apps that can provide contact data whenever a user looks for a contact.
- **File Type Associations** Allows your app to register for handling certain file types (files with the same extension). The file type can be an existing file type or a new file type that's specific for your app.
- **File Open Picker** Allows users to directly select files from your app while using another app.
- **File Save Picker** Allows users to save files directly to your app from another app.
- **Print Task Settings** Allows you to customize the flyout that displays advanced print settings.
- **Search** Adds a search pane to your app that allows users to search in your app and in the data of other apps.
- **Share Target** Allows users to share data from your app with other apps.

There are many possibilities for integrating your app with other apps and with Windows. Whenever you want to implement any of these features, you should update your manifest accordingly.

An exception is when you add a Share or File Open contract. You can add these contracts in Visual Studio in the Add New Item dialog box (see Figure 1-17). When you use this dialog box, your manifest is updated accordingly.


Add New Item - MyDeploymentApp
Installed
JavaScript
Windows Store
Code
Data
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Action JavaScript
Behavior JavaScript
File Open Picker Contract JavaScript
Page Control JavaScript
Search Results Page JavaScript
Share Target Contract JavaScript
Click here to go online and find templates.
Add
Cancel

Type: JavaScript

An app contract that supports receiving items shared by other applications.

Name: shareTarget1.html

FIGURE 1-17 The Add New Item dialog box

EXAM TIP

Make sure that you understand how to use the manifest to configure what your app is allowed to do on a user's device. Remember that you need a privacy statement when your app communicates with the Internet.

Signing an app

To publish your app in the Windows Store, it has to be signed with a certificate. When locally testing your app, Visual Studio generates a certificate that can be used to install your app on a machine that has a developer license.

After creating your package through Visual Studio, you can find the package in the AppPackage folder. Inside this folder, you see a .cer file containing your certificate. When you publish your app to the Windows Store, a new certificate is generated that is linked to your publishers account. This certificate is then used by users to install the app.

MORE INFO CREATING A CERTIFICATE

If you want to manually create a certificate, you can use command-line tools that are part of the Windows Driver Kit (WDK). Documentation can be found at *http://msdn.microsoft.com/en-us/library/windows/apps/jj835832.aspx*.

Planning the requirements for an enterprise deployment

When deploying an enterprise app to users outside of your company, the easiest option is to use the Windows Store. By adding a sign-in page to your app, you can manage licenses and restrict access to your app.

However, if you want to deploy an app to internal users only, you probably don't want to use the Windows Store. This process is called *sideloading*.

Although your app is not validated for the Windows Store, you should make sure that you still follow the certification requirements for the Windows Store. Use the Windows App Certification Kit to validate your app before distributing it inside your company.

The Windows Store normally creates a trusted certificate for you, but you have to create it if you deploy your app without the Windows Store. Make sure that your app is signed with a certificate that's trusted by the PCs on which you will install your app. You can use a certificate that's already installed on your company's network or install a custom certificate specifically for your app.

When your company devices are domain-joined, you can easily configure a Group Policy that allows apps to be sideloaded. You can then install the apps by using the Deployment Image Servicing and Management (DISM) command-line tool or by running Windows PowerShell cmdlets.

Another option is to use Microsoft System Center Configuration Manager or Windows Intune. These commercial products that can manage Windows installations in a corporate environment are available from Microsoft.

MORE INFO SYSTEM CENTER AND INTUNE

For more information on System Center, see *http://blogs.technet.com/b/keithmayer/archive/2013/02/25/step-by-step-deploying-windows-8-apps-with-system-center-2012-service-pack-1.aspx#.UuuSxPldXxQ*. Documentation for Intune can be found at *http://blogs.technet.com/b/windowsintune/archive/2012/10/31/deploying-windows-8-apps-using-windows-intune.aspx*.

Thought experiment

Distributing your app

In this thought experiment, apply what you've learned about this objective. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are creating an app that will be distributed through the Windows Store with your branding and by several companies that have their own branding. With this in mind, answer the following questions:

1. **Do you need to follow the official certification requirements?**
2. **How can you easily change the branding of your app?**
3. **Can you use the Windows Store certificate to distribute your app in enterprise environments?**

Objective summary

- Microsoft has created specific requirements for apps that want to be distributed through the Windows Store.
- Use the Windows App Certification Kit to validate your app.
- An app manifest describes your app and states which integration with the operating system and other apps it supports.
- A certificate is required to distribute your app to other users. The Windows Store helps you generate a certificate.
- You can distribute your app within an enterprise by a process called sideloading, so you don't have to use the Windows Store.

Objective review

Answer the following questions to test your knowledge of the information in this objective. 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. What is contained inside your app package? (Choose all that apply.)
   - **A.** HTML, JavaScript, and CSS files
   - **B.** A block description
   - **C.** A certificate
   - **D.** A manifest
2. You want to access an external web service from your app. Which capability do you require?
   - **A.** Internet (Client & Server)
   - **B.** Internet (Client)
   - **C.** Private network (Client & Server)
   - **D.** Home Network (Client & Server)
3. You want to send your Windows 8 app to a group of testers. What should you do?
   - **A.** Ask them to install Visual Studio and send them the source code of your app.
   - **B.** Send them an app package with Windows PowerShell scripts.
   - **C.** Create a Windows Installer to install the app on their devices.
   - **D.** Submit the app to the Store so they can install it.

Answers

This section contains the solutions to the thought experiments and answers to the lesson review questions in this chapter.

Objective 1.1: Thought experiment

1. The My Restaurant app is great at helping users find the food they love. Of course, you can come up with other "great at" statements. The point is to choose one and use it to guide your app.
2. You can use the Grid App template if you want to center your app on categories of food or recommendations.
3. Yes. A restaurant attracts a variety of individuals, so you should anticipate that some might have disabilities.

Objective 1.1: Review

1. **Correct answer:** C
   - **A.** **Incorrect:** Animations should be used in a Windows Store app. You can use the animations already created for you by Microsoft.
   - **B.** **Incorrect:** You shouldn't design your own animations. You want a consistent feeling across all apps, which is why Microsoft created an animation library.
   - **C.** **Correct:** Animations should be used in your app. The documentation shows which animations are suggested by Microsoft, and your designer can use those animations in his design.
   - **D.** **Incorrect:** You should use the animation library instead of creating completely custom animations.
2. **Correct answer:** D
   - **A.** **Incorrect:** The platform supports apps of any size.
   - **B.** **Incorrect:** Apps can support multiple user scenarios. They should use the "great at" statement to make sure that all scenarios contribute to one great user experience.
   - **C.** **Incorrect:** Having an app that looks beautiful, is fast and fluid, and integrates with other apps doesn't require a "great at" statement. The statement is about what your app does and the user experience it delivers.
   - **D.** **Correct:** Your "great at" statement brings focus to your app and helps you to excel at what you want your app to do.

3. **Correct answers:** A, C, D, E, F
   - A. **Correct:** Pride in craftsmanship is a Microsoft design principle.
   - B. **Incorrect:** Cloud integration is not a Microsoft design principle.
   - C. **Correct:** Fast and fluid is a Microsoft design principle.
   - D. **Correct:** Authentically digital is a Microsoft design principle.
   - E. **Correct:** Do more with less is a Microsoft design principle.
   - F. **Correct:** Win as one is a Microsoft design principle.

Objective 1.2: Thought experiment

1. It is true that a layered application initially adds complexity to your code. Code is spread over several files, and not all code is allowed to call all other code. However, this does increase maintainability and helps you to centralize code around specific tasks, which pays off in the end.
2. This is not true. When a layer defines its interface, other layers can depend on those interface definitions. An implementation could then be faked (such as a static set of test data). You could also implement your application in vertical slices of functionality by developing all layers simultaneously instead of layer by layer.
3. Although JavaScript was not created with large-scale applications in mind, you can certainly create a good architecture in JavaScript. Another way is to use TypeScript to extend JavaScript and make it more suitable for large-scale applications.

Objective 1.2: Review

1. **Correct answer:** A
   - A. **Correct:** The data layer is responsible for fetching data from an external web service.
   - B. **Incorrect:** The service layer is not meant to fetch data from external resources. Instead, it should coordinate actions in your own app.
   - C. **Incorrect:** The business layer enforces business rules and functionality; it does not communicate with external web services.
   - D. **Incorrect:** The UI layer should not communicate with external data sources.

2. **Correct answer:** C
    - **A.** **Incorrect:** Sharing data should not be done through global state because it might create unwanted side effects.
    - **B.** **Incorrect:** You can avoid global state by scoping data to the containing function.
    - **C.** **Correct:** Avoiding global state creates a better maintainable app.
    - **D.** **Incorrect:** Global state is possible in every JavaScript project.
3. **Correct answers:** A, B, C, D
    - **A.** **Correct:** Creating classes is a core element of building a layered application.
    - **B.** **Correct:** Namespaces group classes, which helps you separate code into distinct areas.
    - **C.** **Correct:** WinMD Components can be used to create C# assemblies. This separation in assemblies automatically creates a separation in your code.
    - **D.** **Correct:** A web service is a distinct tier of your application that can host one or more layers.

Objective 1.3: Thought experiment

1. Windows Store apps can save data to a roaming folder, which is automatically synced between all user devices.
2. By handling the checkpoint event, you can save data before the app is suspended. When resuming from a terminated state, you can then restore this data.
3. When resuming your app, you need to restore data from the user’s folder, but you should also make sure that data is not out of date. By using your activated event, you can refresh your external data.

Objective 1.3: Review

1. **Correct answers:** B, C
    - **A.** **Incorrect:** A game needs state management just like every other app. Make sure that users can continue playing games from the moment they left.
    - **B.** **Correct:** The activated event should be used to restore any state after the app terminated.
    - **C.** **Correct:** The checkpoint event can be used to save state before the app is suspended and eventually terminated.
    - **D.** **Incorrect:** The terminated event does not exist.

2. **Correct answer:** B
   - **A.** **Incorrect:** Starting with Windows 8.1, you should completely refresh any data only if the user returns within 10 seconds after closing the app.
   - **B.** **Correct:** If a user doesn't launch the app within 10 seconds, treat the close action as a normal suspend and terminate.
   - **C.** **Incorrect:** This is never an option. You should not show unnecessary dialog boxes to the user.
   - **D.** **Incorrect:** Restarting the application is not an option. You can decide what you want to do with the current application state. However, you should try to follow the required action of resuming the app if the user does not return within 10 seconds.
3. **Correct answer:** D
   - **A.** **Incorrect:** This is a DOM event that you don't have to use inside your app. WinJS offers convenient events that map to your app life cycle.
   - **B.** **Incorrect:** This is a DOM event that you don't have to use inside your app. WinJS offers convenient events that map to your app life cycle.
   - **C.** **Incorrect:** The checkpoint event should be used to save any state before the app is suspended.
   - **D.** **Correct:** A web service is a distinct tier of your application that can host one or more layers.

Objective 1.4: Thought experiment

1. Yes, especially because your app will be deployed to the public Windows Store. For the internal distributed apps, it is also a good idea to follow all certification requirements.
2. The manifest enables you to specify different files for screen shots, tiles, and the splash screen. Other branding can be done like any HTML and CSS app. You can configure different images and CSS files to style your app.
3. No, this certificate is generated when publishing your app to the Windows Store. For internal distribution, you need to sign your app.

Objective 1.4: Review

1. **Correct answers:** A, B, C, D
   - **A. Correct:** All your content files are included in the package.
   - **B. Correct:** The block description is included, which is used to split your app into smaller chunks to make the update process easier.
   - **C. Correct:** A certificate is required for signing your app.
   - **D. Correct:** A manifest is required to describe your app. It is used to show your app in the Windows Store, and to install and run the app on a user's device.
2. **Correct answer:** B
   - **A. Incorrect:** The client and server requirement is used only if your app receives requests from external sources. You are connecting only from your app to an external web service.
   - **B. Correct:** It allows you to connect to your web service.
   - **C. Incorrect:** A private network is required only when you want to connect to other devices inside the network; for example, inside a domain or inside a home network.
   - **D. Incorrect:** The home network option does not exist.
3. **Correct answer:** B
   - **A. Incorrect:** Installing Visual Studio and having access to the source is not required, and it would probably be too much work for testers.
   - **B. Correct:** Windows PowerShell scripts can install the app locally and test it.
   - **C. Incorrect:** A Windows Installer is not required. An app package can be installed with the generated Windows PowerShell scripts.
   - **D. Incorrect:** If you want users to test your app, you should not submit it to the Windows Store. That way, everyone can access your app. And because your app is not yet ready, it will probably fail submission.

Index

Symbols

@font-face CSS rule, 159

A

AccChecker (UI Accessibility Checker) tool, 11
accepting share requests, 90–92
accessibility, UI layout, 11
accessible rich Internet applications (ARIAs), 11
accessing
  contacts,
    ContactPicker class, 58–59
    creating/modifying contact information, 65–66
    filtering contacts to display, 59–61
    selecting specific data, 66–67
    set number of contacts, 61–65
  data, 243–249
    local data, 244–247
    remote data, 247–249
Account Picture Provider declaration, 47
activated events, 64
ActivateKind enumeration, PLM, 37–39
activating apps, 27
ActivationKind enumeration, 84
add event handler, 64
Add New Item dialog box, 9–10, 47–48
Addresses property (Contact class), 65
AdjacentToLeftDisplayEdge property, 167
AdjacentToRightDisplayEdge property, 167
AJAX (Asynchronous JavaScript and XML), 252
AlwaysDisplayDialog property, 274
app bar, 171–176
  AppBar events, 176–177
  app requirements, 171–172
  placement of controls, 175–176
  styling and positioning items, 173–175
AppBar events, 176–177
appendResultSuggestion method, 83
ApplicationDataCreateDisposition enumeration, 280
ApplicationData.RoamingStorageQuota property, 106
applicationHidden event, 232
application links, sharing data, 93–95
Application page (Manifest Designer), 44–45
ApplicationView object, 167
applying CSS styling, 179–183
  gradients, 179–180
  Grid layouts, 181
  media queries, 183–184
  scroll snapping, 182
  zooming, 182
app manifest, 44–48, 75–76
App Manifest Designer
  configuring tile sizes, 214–215
  ContactPicker declaration, 62
  enabling toast notifications, 227–228
  list of declarations, 75
  Minimum Width property, 162–163
  Search declaration, 84–85
  Share declaration, 90
AppointmentsProvider value (ActivationKind enumeration), 37
app requirements
  app bar, 171–172
  data binding, 265–266
  navigation, 206–207
apps
  deployment planning, 40–49
    app manifest, 44–48
    requirements for enterprise deployment, 49
    signing an app, 48
    Windows 8 certification requirements, 40–43
  design
    planning app deployment, 40–49
    PLM (Process Lifetime Management), 26–38

separation of concerns, 17–23
UI layout and structure, 1–14
development
accessing and displaying contacts, 57–67
charms and contracts, 69–79
integration of media features, 109–115
managing application setting s and preferences, 97–107
Search charm, 78–85
sharing data, 87–95
People, 61
AppSettings
adding entry points, 98
choosing accessible features, 97–98
args.detail.previousExecutionState property, 38
ARIAs (accessible rich Internet applications), 11
Asynchronous JavaScript and XML (AJAX), 252
asynchronous methods, 58
AsyncOperation, 259
audio
playback, HTML5 DRM, 115–116
playing sounds with toast notifications, 230–231
authentically digital design principle, 3
authentication, 271–280
Credential Locker, 276–277
CredentialPicker class, 273–276
storing account credentials, 278–281
storing/retrieving credentials, 271–273
web authentication, 282–287
CredentialPicker class, 285
credential roaming, 285
OAuth2, 283–284
proxy authentication, 286–287
SSO (Single Sign-On), 284–285
WebAuthenticationBroker class, 285
Windows.Security.Authentication.Web namespace, 282–283
AuthenticationProtocol property, 274
authorization, 271–280
Credential Locker, 276–277
CredentialPicker class, 273–276
storing account credentials, 278–281
storing/retrieving credentials, 271–273
auto keyword (Grid layout columns and rows), 149
AutoPlay declaration, 47
avoiding global state, loosely coupled layers, 19–20

B

Background Task extension, Manifest Designer, 32–33
background tasks, 32–36
Background Tasks declaration, 47
BackgroundWorkCostChange trigger, 35
BackgroundWorkCostNotHigh condition, background tasks, 35
badges, 219–221
Basic authentication protocol, 275
beginZoom method, 210
bidirectional communication, WebSockets, 260–261
binding data, 263–268
configuring iterators with data-win-options, 266
data-win-bind attribute, 263–264
data-win-control attribute, 263–264
filtering, sorting, and grouping data, 267–268
HTML layout controls, 154–156
item templates, 266
meeting requirements with data-bound controls, 265–266
Blank App template, 8
Blob storage, 247–249
block compressed images sample, 110
break-after property, 153
break-before property, 153
break-inside property, 153
Button control, settings flyout, 102
buttons, app bar, 174

C

Cached File Updater declaration, 47
CachedFileUpdater value (ActivationKind enumeration), 37
CallerSavesCredential property, 274
Camera Settings declaration, 47
CameraSettings value (ActivationKind enumeration), 37
canGoBack property, 139
canGoForward property, 139
capabilities, app manifest, 45
Capabilities page (Manifest Designer), 44
Caption property, 275
capturing gesture library events, 192–195
certification requirements, 40–43
changes in orientation, responding to, 165–166
charms, 69–76

app requirements, 69–74
configuring app manifest for correct permissions, 75–76
designing app to be charm- and contract-aware, 74–75
Devices, 74
Search, 77–85
activation from within search, 84
configuring search contracts, 84–85
providing/constraining search within an app, 81
searching and launching other apps, 81–82
SearchPane and SearchBox control classes, 78–80
search result previews, 82–84
Settings, 95–107
adding entry points for AppSettings, 98
adding setting options to SettingsFlyout control, 101–104
choosing accessible features in AppSettings, 97–98
creating setting flyouts, 99–101
storing and retrieving settings, 105–107
Share, 86–95
accepting sharing requests, 90–92
DataTransferManager class, 87–90
implementing in-app Share outside of charm, 93
limiting scope of sharing using DataPackage object, 92
web links and application links, 93–95
charms bar, 69
checkpoint event, 31–32
classes
Contact, 65–66
ContactPicker, 58–59
CredentialPicker, 273–276, 285
DataTransferManager, 87–90
Horizontal, 157
InputPane, 167
PageControlNavigator, 204, 207
PasswordVault, 271–273
SimpleOrientationSensor, 165
SpeechSynthesizer, 113
TileUpdateManager, 219–221
ToastUpdateManager, 229–230
WebAuthenticationBroker, 285
Windows.UI.Input.Inking.InkManager, 198
Windows.UI.StartScreen.SecondaryTile, 218
WinJS.Binding.List, 267
ClosedByUser (previous state of app), 38
columns, Grid layout, 149
commands, app bar, 173
compareGroups method, 134
ComponentLoadFailed event, 115
conceptual design, UI layout
creating a vision, 5–7
Microsoft design principles, 2–5
conditions, background tasks, 35
configureForZoom method, 210
configuring
iterators, data-win-options, 266
Mobile Services for push notifications, 232–235
search contracts, 84–85
tile sizes, 214–215
ConnectedServiceAccounts property (Contact class), 65
consuming SOAP/WCF services, 259–260
Contact class, 65–66
ContactFieldType property, 59
ContactPicker class, 58–59
ContactPicker declaration, App Manifest Designer, 47, 62
contactPicker.js file, 63
contact pickers, 61–62
ContactPicker value (ActivationKind enumeration), 37
contacts, accessing and displaying, 57–67
ContactPicker class, 58–59
creating/modifying contact information, 65–66
filtering contacts to display, 59–61
selecting specific data, 66–67
set number of contacts, 61–65
Contact value (ActivationKind enumeration), 37
Content property, 253
contracts, 69–76
app requirements, 69–74
configuring app manifest for correct permissions, 75–76
designing app to be charm- and contract-aware, 74–75
ControlChannelReset trigger, 34
controls
custom, 12
flyouts, 131
HTML layout controls, 147–160
structuring layout, 147–154
templates and bindings, 154–156
ListView, 132–134
implementing data-bound controls, 265–266

implementing drag-and-drop, 199–201
placement on app bar, 175–176
SemanticZoom, 209
SettingsFlyout
adding settings options to, 101–104
creating setting flyouts, 99–101
WinJS, 125–145
FlipView, 126–130
flyout, 130–132
Grid and List layout, 132–135
item containers, 140–142
menu objects, 135–137
Repeater, 142–145
WebView, 138–140
WinJS.UI.AppBar, 173
CookieManager, 254
Create App Packages dialog box, 43
CreateCollisionOption parameter (createFileAsync method), 245
createFileAsync method, 245
createFiltered method, 267
createGrouped method, 134, 267
Create, Read, Update, and Delete (CRUD) operations, 249, 256
createSorted method, 267
creating
app packages, 41–42
contact information, 65–66
custom gesture recognizers, 196–197
setting flyouts, 99–101
tiles, 214–225
creating and updating badges, 219–221
periodic notifications, 222–223
push notifications, 223–224
responding to notification requests, 221
scheduled notifications, 221–222
tile update schedule, 224–225
updating tiles and tile content, 214–219
UI (user interface)
app bar, 171–176
CSS styling, 179–183
HTML layout controls, 147–160
layout-aware apps, 162–169
WinJS controls, 125–145
Credential Locker, 276–277
CredentialPicker class, 273–276, 285
CredentialPickerOptions object, 274
credential roaming, 285
credentials
storing and retrieving, 271–273
storing in app settings, 278–281
verifying with Credential Locker, 276–277
CredentialSaveOption property, 275
CredSsp authentication protocol, 275
crossSlideExact property, 196
crossSlideHorizontally property, 196
crossSlideThresholds property, 196
CRUD (Create, Read, Update, and Delete) operations, 249, 256
cryptography, 41
CSS3
media queries, 162–165
scrolling and zooming, 156–158
CSS styling, 179–183
Flyout control, 132
gradients, 179–180
Grid layouts, 181
media queries, 183–184
scroll snapping, 182
zooming, 182
Custom authentication protocol, 275
CustomAuthenticationProtocol property, 275
custom contact pickers, 61–62
custom controls, 12
custom formats, 89
custom gesture recognizers, 196–197
custom layout, app bar layout, 176

D

data
binding, 263–268
configuring iterators with data-win-options, 266
data-win-bind attribute, 263–264
data-win-control attribute, 263–264
filtering, srting, and grouping data, 267–268
HTML layout controls, 154–156
item templates, 266
meeting requirements with data-bound controls, 265–266
formats that can be shared, 88
management
data access strategy, 243–249
data binding, 263–268
retrieving data remotely, 251–260

roaming, 105–107
sharing, 87–95
accepting share requests, 90–92
DataTransferManager class, 87–90
implementing in-app Share outside of charm, 93
limiting scope of sharing using DataPackage object, 92
web links and application links, 93–95
temporary, 105
data-bound controls, 265–266
DataPackage object, 92
DataRequestedEventArgs, 88
datarequested handler, 88
dataSource property, 127
DataSuppliers property (Contact class), 65
DataTransferManager class, 87–90
data-win-bind attribute, 155, 263–264
data-win-control attribute, 126, 263–264
data-win-options, configuring iterators, 266
DDS (DirectDraw Surface) images, 110–111
declarations, app manifest, 45, 75
Declarations page (Manifest Designer), 44
DeleteAsync method, 257
delivery methods, tiles, 224
Deployment Image Servicing and Management (DISM) command-line tool, 49
deployment of apps, planning, 40–49
app manifest, 44–48
requirements for enterprise deployment, 49
signing an app, 48
Windows 8 certification requirements, 40–43
design
app bar, 171–176
AppBar events, 176–177
app requirements, 171–172
placement of controls, 175–176
styling and positioning items, 173–175
Microsoft design principles, 2–5
navigation, 202–212
loading HTML fragments, 210–212
meeting app requirements, 206–207
semantic zoom, 207–210
WinJS.Navigation namespace, 203–206
Windows Store apps
planning app deployment, 40–49
PLM (Process Lifetime Management), 26–38
separation of concerns, 17–23
UI layout and structure, 1–14
desiredFieldsWithContactFieldType property, 59
development, Windows Store apps
accessing and displaying contacts, 57–67
charms and contracts, 69–79
integration of media features, 109–115
managing application settings and preferences, 97–107
Search charm, 78–85
sharing data, 87–95
Devices charm, 74
Device value (ActivationKind enumeration), 37
dialog boxes
Add New Item, 9–10, 47–48
Create App Packages, 43
Manifest Designer, 29
Speech Properties, 114
Windows App Certification Kit, 43
Digest authentication protocol, 275
Digital Rights Management (DRM), 115–116
DirectDraw Surface (DDS) images, 110–111
DirectX Video Acceleration (DXVA), 112
DISM (Deployment Image Servicing and Management) command-line tool, 49
displaying
contacts, 57–67
ContactPicker class, 58–59
creating/modifying contact information, 65–66
filtering contacts to display, 59–61
selecting specific data, 66–67
set number of contacts, 61–65
DisplayName property (Contact class), 65
drag-and-drop events, 199–201
dragenter event, 141
dragenter event handler, 200
dragleave event, 141
dragleave event handler, 200
dragover event, 141
dragstart event, 141
DRM (Digital Rights Management), 115–116
drop event, 141
drop event handler, 200
DXVA (DirectX Video Acceleration), 112

E

Emails property (Contact class), 65
enableNotificationQueue method, 223

enabling apps for toast notifications, 227–229
encrypting data, 278
endZoom method, 210
Enterprise Authentication capability, 46
enterprise deployment, planning requirements, 49
entry points, 98–101
ErrorCode property, 275
error codes, 275
Error method, 204
event.detail.insertAfterIndex property, 201
event handlers, 64
event.preventDefault method, 199
Expression Sketchflow, 7

F

FailIfExists value (CreateCollisionOption parameter), 245
fast and fluid design principle, 2
FIFO (first-in, first-out) approach, 223
File Open Picker declaration, 47
FileOpenPicker value (ActivationKind enumeration), 37
files
  contactPicker.js, 63
  ui.js, 126
File Save Picker declaration, 47
FileSavePicker value (ActivationKind enumeration), 37
File Type Associations declaration, 47
File value (ActivationKind enumeration), 37
filtering
  contacts, 59–61
  data, 267–268
findAllByResource method, 273, 277
findAllByUserName method, 273
first-in, first-out (FIFO) approach, 223
FirstName property (Contact class), 65
FlexBox (Flexible Box) layout, 147–149
Flexible Box (FlexBox) layout, 147–149
FlipView control, 126–130
flyouts, 99-101, 130–132
formats, sharing data, 88
fraction unit (Grid layout columns and rows), 150
FreeNetworkAvailable condition, background tasks, 35
functions. *See also* methods
  pickContactsAsync, 59
  rgba, 180
  template, 129

G

GenerateUniqueName value (CreateCollisionOption parameter), 245
gesture recognizers, 196–197
gestures, 192–195
  custom gesture recognizers, 196–197
  touch, 198
gestureSettings property, 196
GetAsync(Uri, HttpCompletionOption) method, 256
GetAsync(Uri) method, 256
GetBufferAsync method, 256
getCommandById method, 176
getCurrentItem method, 210
getDeferral method, 89
getGroupKey method, 134
GetInputStreamAsync method, 256
getPanAxis method, 210
GetStockPrice method, 259
GetStringAsync method, 253, 256
global positioning of commands, app bar, 175
global searches, 70
global state, loosely coupled layers, 19–20
glyph badges, 220–221
gradients, CSS styling, 179–180
Grid App template, 9
Grid layout, 132–135, 149–151, 181
grouping data, 267–268

H

handlePointer method, 210
handwriting recognition, 198
hashing algorithms, 279
hashing data, 278
Headers property, 253
HonorificNamePrefix property (Contact class), 65
HonorificNameSuffix property (Contact class), 65
Horizontal class, 157
HTML5 DRM (Digital Rights Management), 115–116
HTML fragments, navigation, 210–212
HTML layout controls, 147–160
  scrolling and zooming with CSS3, 156–158
  structuring layout, 147–154
    Flexible Box (FlexBox) layout, 147–149
    Grid layout, 149–151
    multi-column layout, 152–154

Regions layout, 153–154
templates and bindings, 154–156
text flow and presentation, 159–160
HTML PageControl, 11
HttpBaseProtocolFilter, 254
HttpClient, retrieving web servicves, 252–255
HttpDelete requests, 256
HttpGet actions, 256–257
HttpPost, 257–258
HttpPut verb, 257
HttpResponseMessage object, 253
Hub App template, 12–15
Hyperlink control, settings flyout, 102

I

Id property, 65, 167
implementation
app bar, 171–176
AppBar events, 176–177
app requirements, 171–172
placement of controls, 175–176
styling and positioning items, 173–175
HTML layout controls, 147–160
structuring layout, 147–154
templates and bindings, 154–156
navigation, 202–212
loading HTML fragments, 210–212
meeting app requirements, 206–207
semantic zoom, 207–210
WinJS.Navigation namespace, 203–206
PLM (Process Lifetime Management), 26–38
ActivateKind enumeration, 37–39
onactivated event, 27–30
preparing for app termination, 32–36
previous state, 37–39
state management strategy, 26
suspended event, 31–32
Search charm, 78–85
activation from within search, 84
configuring search contracts, 84–85
providing/constraining search within an app, 81
searching and launching other apps, 81–82
SearchPane and SearchBox control classes, 78–80
search result previews, 82–84
Share charm, 87–95
accepting share requests, 90–92
DataTransferManager class, 87–90
implementing in-app Share outside of charm, 93
limiting scope of sharing using DataPackage object, 92
web links and application links, 93–95
WinJS controls, 125–145
FlipView, 126–130
flyout, 130–132
Grid and List layout, 132–135
item containers, 140–142
menu objects, 135–137
Repeater, 142–145
WebView, 138–140
ImportantDates property (Contact class), 65
in-app searches, 70, 78
in-app Share, 93
inheritance, designing UI layout, 10–11
Init method, 204
inking, 198–199
InkManager class, 198
InkStroke objects, 198
input devices, 191–201
capturing gesture library events, 192–195
custom gesture recognizers, 196–197
drag-and-drop events, 199–201
listening to mouse events or touch gestures, 198
stylus input and inking, 198–199
Input Ink sample (SDK), 199
InputPane class, 167
Input: Simplified ink sample (SDK), 199
Inspect tool, 11
integration
apps, charms and contracts, 69–76
configuring app manifest for correct permissions, 75–76
designing app to be charm- and contract-aware, 74–75
media features, 109–115
audio and video playback using HTML5 DRM, 115–116
DXVA, 112
supporting DDS images, 110–111
TTS, 113–115
video playback, 111
XVP, 112
InternetAvailable condition, background tasks, 35
InternetAvailable trigger, 34

Internet (Client) capability, 46
Internet (Client & Server) capability, 46
InternetNotAvailable condition, background tasks, 35
invokeScryptAsync method, 139
IsFullScreen property, 167
IsOnLockScreen property, 167
IsScreenCaptureEnabled property, 167
IsSuccessStatusCode property, 253
item containers, 140–142
itemDataSource property, 127
itemdragdrop event, 201
itemdragend event handler, 199
itemdragenter event, 201
itemdragstart event handler, 199
item templates, 9–10, 266
iterators, configuring with data-win-options, 266
IZoomableView interface, 209, 210

J

JobInfo property (Contact class), 65

K

Keep It Simple (KIS) principle, 10
Kerberos authentication protocol, 275
KeyModifiers (SearchBoxQuerySubmittedEventArgs), 79
KIS (Keep It Simple) principle, 10

L

Language (SearchBoxQuerySubmittedEventArgs), 79
LastName property (Contact class), 65
launching apps, Search charm, 81–82
Launch value (ActivationKind enumeration), 37
layout-aware apps, creating, 162–169
- CSS3 media queries, 162–165
- responding to changes in orientation, 165–166
- settings for an apps view, 168–170
- ViewManagement namespace, 167–168

layout controls, 147–160
- scrolling and zooming with CSS3, 156–158
- structuring layout, 147–154
  - Flexible Box (FlexBox) layout, 147–149
  - Grid layout, 149–151
  - multi-column layout, 152–154
  - Regions layout, 153–154
- templates and bindings, 154–156
- text flow and presentation, 159–160

layout, UI, 1–14
- conceptual design evaluation, 2–7
  - creating a vision, 5–7
  - Microsoft design principles, 2–5
- custom controls, 12
- designing for accessibility, 11
- Hub App template, 13–15
- inheritance and reuse of visual elements, 10–11
- project templates, 7–10

life cycle, apps, 26
limiting scope of sharing, DataPackage object, 92
linear gradients, 179
LinguisticDetails (SearchBoxQuerySubmittedEventArgs), 79
List layout, 132–135
List objects, 127
ListView control, 132–134
- implementing data-bound controls, 265–266
- implementing drag-and-drop, 199–201

live tiles, 214
loading HTML fragments, navigation, 210–212
local data, 105, 244–247
localSettings property, 279
Location capability, 46
LockScreenApplicationAdded trigger, 34
LockScreenApplicationRemoved trigger, 34
LockScreenCall value (ActivationKind enumeration), 37
logical layers, 17–18
long-duration toasts, 232
lookup method, 280
looping sounds, 230
loosely coupled layers, 19–22
- avoiding global state, 19–20
- strict mode, 21
- TypeScript, 21–22

M

Mail share target, sharing data, 73
main content (flyouts), 131
maintainable software, 17–23
- integrating WinMD Components, 22–23
- loosely coupled layers, 19–22

planning logical layers, 17–18
management
application settings and preferences, 97–107
adding entry points for AppSettings, 98
adding setting options to SettingsFlyout control, 101–104
choosing accessible features in AppSettings, 97–98
creating setting flyouts, 99–101
storing and retrieving settings, 105–107
data
data access strategy, 243–249
data binding, 263–268
retrieving data remotely, 251–260
input devices, 191–201
capturing gesture library events, 192–195
custom gesture recognizers, 196–197
drag-and-drop events, 199–201
listening to mouse events or touch gestures, 198
stylus input and inking, 198–199
security
authorization and authentication, 271–280
web authentication, 282–287
settings for an apps view, 168–170
tiles, 214–225
creating and updating badges, 219–221
periodic notifications, 222–223
push notifications, 223–224
responding to notification requests, 221
scheduled notifications, 221–222
tile update schedule, 224–225
updating tiles and tile content, 214–219
mandatory snapping, 158, 182
Manifest Designer
Application page, 44–45
Background Task extension, 32–33
Manifest Designer dialog box, 29
matrices, 194
max-content keyword (Grid layout columns and rows), 150
media features, integration, 109–115
audio and video playback using HTML5 DRM, 115–116
DXVA, 112
supporting DDS images, 110–111
TTS, 113–115
video playback, 111
XVP, 112
Media Foundation Transcode Video Processor (XVP), 112
MediaProtectionManager object, 115
media queries
creating layout-aware apps, 162–165
CSS styling, 183–184
menu objects, 135–137
Message property, 275
methods. *See also* functions
appendResultSuggestion, 83
beginZoom, 210
compareGroups, 134
configureForZoom, 210
createFileAsync, 245
createFiltered, 267
createGrouped, 134, 267
createSorted, 267
DeleteAsync, 257
delivery methods for tiles, 224
enableNotificationQueue, 223
endZoom, 210
Error, 204
event.preventDefault, 199
findAllByResource, 273, 277
findAllByUserName, 273
GetAsync(Uri), 256
GetAsync(Uri, HttpCompletionOption), 256
GetBufferAsync, 256
getCommandById, 176
getCurrentItem, 210
getDeferral, 89
getGroupKey, 134
GetInputStreamAsync, 256
getPanAxis, 210
GetStockPrice, 259
GetStringAsync, 253, 256
handlePointer, 210
Init, 204
invokeScryptAsync, 139
lookup, 280
myFunction, 139
navigateToString, 138
onTimer, 36
pickAsync, 274
pickContactsAsync, 58
positionItem, 210
PostAsync, 257
Processed, 204

PutAsync, 257
Ready, 204
response.ensureSuccessStatusCode(), 254
retrieveAll, 273
sensor.getCurrentOrientation(), 166
SetApplicationLink, 89, 92
SetBitmap, 89, 92
setCurrentItem, 210
SetData, 89, 92
SetDataProvider, 89, 92
SetHtmlFormat, 89, 92
SetRtf, 89, 92
SetStorageItems, 89, 92
SetText, 89, 92
SetUri, 89, 92
SetWebLink, 89, 92
showMenu, 136
swapDisplaysForViewsAsync, 169
synthesizeSsmlToStreamAsync, 114
synthesizeTextToStreamAsync, 114
Unload, 204
UpdateLayout, 205
WebAuthenticationBroker.authenticateAsync, 283
Windows.Devices.Sensors.SimpleOrientationSensor.getDefault(), 165
Windows.UI.ViewManagement.ApplicationView.getForCurrentView(), 167, 168
WinJS.Binding.as, 264
WinJS.Binding.as ', 155
WinJS.Binding.processAll, 155
WinJS.Class.define, 20
WinJS.Namespace.define, 19
WinJS.Navigation.back, 206
WinJS.Navigation.forward, 206
WinJS.Navigation.navigate, 206
WinJS.UI.Pages.define, 204
WinJS.UI.processAll, 126
WinJS.xhr, 252
writeBytesAsync, 245
Microphone capability, 46
Microsoft design principles, 2–5
MiddleName property (Contact class), 65
min-content keyword (Grid layout columns and rows), 149
Minimum Width property, 162–163
minmax(a, b) keyword (Grid layout columns and rows), 150
mipmaps, 110
Mobile Services, configuring for push notifications, 232–235
modifying contact information, 65–66
more with less design principle, 3
mouse events, 198
MrfCrf444 mode, 112
ms-appdata protocol, 139
ms-content-zooming property, 158
ms-content-zooming: zoom property, 182
MSCSSMatrix object, 194
ms-flex: 2 property, 149
ms-flex-align property, 149
ms-flexbox property, 148
ms-flex-pack property, 149
ms-flex-wrap property, 149
MSGestureChange event, 193
MSGestureEnd event, 193
MSGestureHold event, 193
MSGestureStart event, 193
MSGestureTap event, 193
ms-grid-column-align property, 150
ms-grid-column-span property, 150
ms-grid-row-align property, 150
ms-grid-row-span property, 150
ms-scroll-snap-type property, 158, 182
MSWebViewAsyncOperation object, 139
MSWebViewScriptNotify event, 140
multi-column layout, 152–154
Music Library capability, 46
myFunction method, 139

N

namespaces
ViewManagement, 167–168
Windows.Applicationmodel. Contacts, 58
Windows.Media.SpeechSynthesis, 113
Windows.Media.Transcoding, 112
Windows.Security.Authentication.Web, 282–283
Windows.Security.Credentials.UI, 274
Windows.UI.Input.GestureRecognizer, 196
Windows.UI.Notification, 217
WinJS.Navigation, 203–206
WinJS.UI.Fragments, 210
Narrator, 12
navigateToString method, 138
navigation, 202–212

loading HTML fragments, 210–212
meeting app requirements, 206–207
semantic zoom, 207–210
WinJS.Navigation namespace, 203–206
Navigation App template, 8
NCP (Notification Client Platform), 223
Negotiate authentication protocol, 275
NetworkStateChange trigger, 34
nonlooping sounds, 230
Notes property (Contact class), 65
Notification Client Platform (NCP), 223
notifications
tiles, 221
toast, 227–235
configuring Mobile Services for push notifications, 232–235
controlling toast duration, 232
enabling apps for, 227–229
populating with images and text, 229–230
responding to toast events, 231–232
NotificationsExtensions library, 217
NotRunning (previous state of app), 38
N-tiered applications, 18
Ntlm authentication protocol, 275
number badges, 220

O

OAuth2 authentication, 283–284
object-oriented design concepts, 21
objects
ApplicationView, 167
CredentialPickerOptions, 274
DataPackage, 92
HttpResponseMessage, 253
InkStroke, 198
List, 127
MediaProtectionManager, 115
MSCSSMatrix, 194
MSWebViewAsyncOperation, 139
searchSuggestionCollection, 82
shareOperation, 91
WinJS.Application, 32
WinJS.Application.sessionState, 31
XMLHttpRequest, 252
onactivated event, PLM, 27–30
onafterhide event, 176
onaftershow event, 176
onbeforehide event, 176
onbeforeshow event, 176
one-directional communication, 260
OnlineIdConnectedStateChange trigger, 35
onpagecomplete event, 129
onpageselected event, 129
onpagevisibilitychanged event, 129
onTimer method, 36
OpenIfExists value (CreateCollisionOption parameter), 245
openMenuButton HTML element, 136
OrientationChanged event, 165
orientation changes, responding to, 165–166
Orientation property, 167
overflow, presenting text, 160
overloads, CredentialPicker, 274

P

Package Family Name (PFN), 94
Packaging page (Manifest Designer), 44
PAC (Privilege Attribute Certificate), 286
PageControlNavigator class, 204, 207
panning, 156–158
PasswordVault class, 271–273
People app, 61
periodic notifications, tiles, 222–223
Permissions option (Settings charm), 97
PFN (Package Family Name), 94
Phones property (Contact class), 66
pickAsync method, 274
pickContactAsync callback, 61
pickContactsAsync method, 58–59
Pictures Library capability, 46
pinch gesture, 192
placement, app bar controls, 175–176
planning
app deployment, 40–49
app manifest, 44–48
requirements for enterprise deployment, 49
signing an app, 48
Windows 8 certification requirements, 40–43
logical layers, 17–18
playing sounds, toast notifications, 230–231
PLM (Process Lifetime Management), 26–38
ActivateKind enumeration, 37–39

onactivated event, 27–30
preparing for app termination, 32–36
previous state, 37–39
state management strategy, 26
suspended event, 31–32
pointer input, 192
poll notifications, 223
positioning app bar items, 173–175
positionItem method, 210
PostAsync method, 257
preferences, 97–107
adding entry points for AppSettings, 98
adding setting options to SettingsFlyout control, 101–104
choosing accessible features in AppSettings, 97–98
creating setting flyouts, 99–101
storing and retrieving settings, 105–107
presentation, HTML layout controls, 159–160
press and hold gesture, 192
previews, search results, 82–84
PreviousCredential property, 275
previous state, PLM, 37–39
pride in craftsmanship design principle, 2
Print Task Settings declaration, 47
PrintTaskSettings value (ActivationKind enumeration), 37
Private Network (Client & Server) capability, 46
Privilege Attribute Certificate (PAC), 286
Process Lifetime Management. *See* PLM
Processed method, 204
program user interaction
input devices, 191–201
capturing gesture library events, 192–195
custom gesture recognizers, 196–197
drag-and-drop events, 199–201
listening to mouse events or touch gestures, 198
stylus input and inking, 198–199
navigation, 202–212
loading HTML fragments, 210–212
meeting app requirements, 206–207
semantic zoom, 207–210
WinJS.Navigation namespace, 203–206
tiles, 214–225
creating and updating badges, 219–221
periodic notifications, 222–223
push notifications, 223–224
responding to notification requests, 221
scheduled notifications, 221–222
tile update schedule, 224–225
updating tiles and tile content, 214–219
toast notifications, 227–235
configuring Mobile Services for push notifications, 232–235
controlling toast duration, 232
enabling apps for, 227–229
populating with images and text, 229–230
responding to toast events, 231–232
progress updates, data requests, 258–259
ProjectionManager, 169
project templates, UI layout, 7–10
properties
AdjacentToLeftDisplayEdge, 167
AdjacentToRightDisplayEdge, 167
AlwaysDisplayDialog, 274
ApplicationData.RoamingStorageQuota, 106
AuthenticationProtocol, 274
break-after, 153
break-before, 153
break-inside, 153
CallerSavesCredential, 274, 275
canGoBack, 139
canGoForward, 139
Caption, 275
Contact class, 65–66
ContactFieldType, 59
Content, 253
CredentialSaveOption, 275
crossSlideExact, 196
crossSlideHorizontally, 196
crossSlideThresholds, 196
CustomAuthenticationProtocol, 275
dataSource, 127
data-win-bind, 155
desiredFieldsWithContactFieldType, 59
ErrorCode, 275
event.detail.insertAfterIndex, 201
gestureSettings, 196
Headers, 253
HttpResponseMessage object, 253
Id, 167
IsFullScreen, 167
IsOnLockScreen, 167
IsScreenCaptureEnabled, 167
IsSuccessStatusCode, 253

- itemDataSource, 127
- localSettings, 279
- Message, 275
- Minimum Width, 162–163
- ms-content-zooming, 158
- ms-content-zooming: zoom, 182
- ms-flex: 2, 149
- ms-flex-align, 149
- ms-flexbox, 148
- ms-flex-pack, 149
- ms-flex-wrap, 149
- ms-grid-column-align, 150
- ms-grid-column-span, 150
- ms-grid-row-align, 150
- ms-grid-row-span, 150
- ms-scroll-snap-type, 158, 182
- Orientation, 167
- PreviousCredential, 275
- ReasonPhrase, 253
- request.data, 89
- RequestMessage, 253
- responseData, 284
- roamingSettings, 279
- RoamingSettings, 105
- RoamingStorageQuota, 244
- scheme, 168
- selectionDisabled, 141
- selectionMode, 59
- Source, 253
- StatusCode, 253
- TargetName, 275
- TerminateAppOnFinalViewClose, 167
- text-overflow, 160
- Title, 167
- touch-action CSS, 158
- Value, 167
- Version, 253
- Windows.Media.SpeechSynthesis.SpeechSynthesizer.allVoices, 114
- WinJS.Binding.optimizeBindingReferences, 154
- z-index, 150

Protocol value (ActivationKind enumeration), 37
prototypes, 7
ProviderProperties property (Contact class), 66
Proximity capability, 46
proximity snapping, 158, 182
proxy authentication, 286–287
push notifications
- configuring Mobile Services for, 232–235
- tiles, 223–224

PutAsync method, 257

Q

querysubmitted event, 79–80
query suggestions, searches, 72, 79, 82
QueryText (SearchBoxQuerySubmittedEventArgs), 79

R

radial gradients, 179
Radio Button control, settings flyout, 102
RandomAccessStreamReference, 83
Rate And Review option (Settings charm), 97
ratings, 41
Ready method, 204
real-time communication, WebSockets, 260–261
ReasonPhrase property, 253
RebootNeeded event, 115
Regions layout, 153–154
remote data, access strategy, 247–249
remote retrieval of data, 251–260
- consuming SOAP/WCF services, 259–260
- handling progress of data requests, 258–259
- setting HTTP verbs for REST, 255–258
- using WebSockets for bidirectional communication, 260–261
- XHR or HTTPClient, 252–255

Removable Storage capability, 46
Repeater control, 142–145
ReplaceExisting value (CreateCollisionOption parameter), 245
Representational State Transfer (REST) services, 255–258
request.data property, 89
RequestMessage property, 253
requirements, apps
- app bar, 171–172
- charms and contracts, 69–74
- data binding, 265–266
- navigation, 206–207

responding to toast events, 231–232

responseData property, 284
response.ensureSuccessStatusCode() method, 254
REST (Representational State Transfer) services, 255–258
RestrictedLaunch value (ActivationKind enumeration), 37
restrictions, using WinRT Components, 23
resultsuggestionchosen event, 83
result suggestions, searches, 72, 79, 82
retrieveAll method, 273
retrieving
  credentials, 271–273
  data, 251–260
    consuming SOAP/WCF services, 259–260
    handling progress of data requests, 258–259
    setting HTTP verbs for REST, 255–258
    using WebSockets for bidirectional communication, 260–261
    XHR or HTTPClient, 252–255
  settings from roaming app data store, 105–107
reuse of visual elements, designing UI layout, 10–11
rgba function, 180
roaming app data store, storing and retrieving settings, 105–107
roaming folder, storing local data, 244
RoamingSettings property, 105, 279
RoamingStorageQuota property, 244
rows, Grid layout, 149

S

scheduled notifications, tiles, 221–222
scheme property, 168
scrolling, 156–158
scroll snapping, 182
SDK (software development kit), 11, 199
SearchBox control, 78–80
SearchBoxResultSuggestionChosenEventArgs, 83
Search charm, 70–72, 78–85
  activation from within search, 84
  configuring search contracts, 84–85
  providing/constraining search within an app, 81
  searching and launching other apps, 81–82
  SearchPane and SearchBox control classes, 78–80
  search result previews, 82–84
search contracts, configuring, 84–85
Search declaration, 47, 84–85
SearchPane control, 78–80
search result previews, 82–84
searchSuggestionCollection object, 82
Search value (ActivationKind enumeration), 37
secondary tiles, 218
security management
  authorization and authentication, 271–280
    Credential Locker, 276–277
    CredentialPicker class, 273–276
    storing account credentials, 278–281
    storing/retrieving credentials, 271–273
  web authentication, 282–287
    CredentialPicker class, 285
    credential roaming, 285
    OAuth2, 283–284
    proxy authentication, 286–287
    SSO (Single Sign-On), 284–285
    WebAuthenticationBroker class, 285
    Windows.Security.Authentication.Web namespace, 282–283
Segoe UI font family, 159
Select Control, settings flyout, 102
selectionDisabled property, 141
selectionMode property, 59
Selection option, positioning commands on app bar, 175
semantic zoom, 207–210
SemanticZoom control, 209
sensor.getCurrentOrientation() method, 166
separation of concerns. *See* SoC
separators, app bar, 175
ServiceRequested event, 115
ServicingComplete trigger, 34
SessionConnected condition, background tasks, 35
SessionConnected trigger, 34
SessionDisconnected condition, background tasks, 35
SetApplicationLink method, 89, 92
SetBitmap method, 89, 92
setCurrentItem method, 210
SetData method, 89, 92
SetDataProvider method, 89, 92
SetHtmlFormat method, 89, 92
SetRtf method, 89, 92
SetStorageItems method, 89, 92
SetText method, 89, 92
setting flyouts, 99–101
Settings charm, 74, 97–107
  adding entry points for AppSettings, 98

adding setting options to SettingsFlyout control, 101–104
choosing accessible features in AppSettings, 97–98
creating setting flyouts, 99–101
storing and retrieving settings, 105–107
SettingsFlyout control
adding settings options to, 101–104
creating setting flyouts, 99–101
Settings window, 98
SetUri method, 89, 92
SetWebLink method, 89, 92
Share charm, 72–73, 87–95
accepting sharing requests, 90–92
DataTransferManager class, 87–90
implementing in-app Share outside of charm, 93
limiting scope of sharing using DataPackage object, 92
web links and application links, 93–95
Share contract, 72–73
Shared Access Signatures, 247
Share declaration, App Manifest Designer, 90
Shared User Certificates capability, 46
shareOperation object, 91
Share Target declaration, 47
share targets, 73
ShareTarget value (ActivationKind enumeration), 37
sharing
contact information, 57–67
ContactPicker class, 58–59
creating/modifying contact information, 65–66
filtering contacts to display, 59–61
set number of contacts, 61–65
data, 87–95
accepting sharing requests, 90–92
DataTransferManager class, 87–90
implementing in-app Share outside of charm, 93
limiting scope of sharing using DataPackage object, 92
web links and application links, 93–95
showMenu method, 136
sideloading apps, 49
SignalR, 224
SignificantOthers property (Contact class), 66
signing an app, 48
Simple Object Access Protocol (SOAP) services, consuming, 259–260
SimpleOrientation enumeration, 166
SimpleOrientationSensor class, 165
SimpleOrientationSensorOrientation ChangedEventArg, 166
simulators
testing apps, 164
testing gestures, 195
single-page application (SPA) structure, 9, 203
Single Sign-On (SSO), 284–285
slide gesture, 192
SmsReceived trigger, 34
snapping, 158
SOAP (Simple Object Access Protocol) services, consuming, 259–260
SoC (separation of concerns), 17–23
integrating WinMD Components, 22–23
loosely coupled layers, 19–22
planning logical layers, 17–18
software development kit (SDK), 11, 199
software development, maintainability, 17–23
integrating WinMD Components, 22–23
loosely coupled layers, 19–22
planning logical layers, 17–18
solid-state drives (SSDs), 3
sorting data, 267–268
sound, playing sounds with toast notifications, 230–231
Source property, 253
SPA (single-page application) structure, 9, 203
Speech Properties dialog box, 114
Speech Synthesis Markup Language (SSML), 113
SpeechSynthesizer class, 113
Split App template, 9
SSDs (solid-state drives), 3
SSML (Speech Synthesis Markup Language), 113
SSO (Single Sign-On), 284–285
standard length unit (Grid layout columns and rows), 149
standard toasts, 232
state management, PLM (Process Lifetime Management), 26
static tiles, 214
StatusCode property, 253
storage
account credentials, 278–281
credentials, 271–273
data
local data, 244–246
remote data, 247–249
settings from roaming app data store, 105–107
Storyboarding plug-in, 7

strict mode, loosely coupled layers, 21
structure, UI, 1–14
 conceptual design evaluation, 2–7
  creating a vision, 5–7
  Microsoft design principles, 2–5
 custom controls, 12
 designing for accessibility, 11
 Hub App template, 13–15
 inheritance and reuse of visual elements, 10–11
 project templates, 7–10
structuring layout, HTML layout controls, 147–154
 Flexible Box (FlexBox) layout, 147–149
 Grid layout, 149–151
 multi-column layout, 152–154
 Regions layout, 153–154
styling, app bar, 173–175
stylus input, 198–199
suggestionsrequested event, 80, 82
suspended event, PLM, 31–32
Suspended (previous state of app), 38
suspended state, apps, 27
swapDisplaysForViewsAsync method, 169
swipe gesture, 192
synthesizeSsmlToStreamAsync method, 114
synthesizeTextToStreamAsync method, 114

T

tap gesture, 192
TargetName property, 275
Telerik, 12
template function, 129
templates
 HTML layout controls and, 154–156
 Hub App, 13–15
temporary data, 105
temporary folder, storing local data, 244
TerminateAppOnFinalViewClose property, 167
Terminated (previous state of app), 38
termination of apps, PLM, 32–36
testing gestures, 194
text flow, HTML layout controls, 159–160
Text Input Box control, settings flyout, 102
text-overflow property, 160
Text To Speech (TTS), 113–115
Thumbnail property (Contact class), 66
tiers, 18
tiles, 214–225
 creating and updating badges, 219–221
 periodic notifications, 222–223
 push notifications, 223–224
 responding to notification requests, 221
 scheduled notifications, 221–222
 tile update schedule, 224–225
 updating tiles and tile content, 214–219
TileTemplateType enumeration, 217
TileUpdateManager class, 219–221
TileUpdateManager.getTemplateContent, 217
TimeZoneChange trigger, 34
Title (flyout), 131
Title property, 167
toast events, 231–232
toast notifications, 227–235
 configuring Mobile Services for push notifications, 232–235
 controlling toast duration, 232
 enabling apps for, 227–229
 populating with images and text, 229–230
 responding to toast events, 231–232
ToastUpdateManager class, 229–230
toggles, app bar, 174
Toggle Switch control, settings flyout, 102
tools
 AccChecker (UI Accessibility Checker), 11
 DISM (Deployment Image Servicing and Management) command-line tool, 49
 Inspect, 11
touch-action CSS property, 158
touch gestures, 198
transcoding video, 112
triggers, background tasks, 34
TTS (Text To Speech), 113–115
turn gesture, 192
TypeScript, 21–22

U

UI Accessibility Checker (AccChecker) tool, 11
ui.js files, 126
UI (user interface)
 creating
  app bar, 171–176

CSS styling, 179–183
HTML layout controls, 147–160
layout-aware apps, 162–169
WinJS controls, 125–145
layout and structure, 1–14
conceptual design evaluation, 2–7
custom controls, 12
designing for accessibility, 11
Hub App template, 13–15
inheritance and reuse of visual elements, 10–11
project templates, 7–10
Unload method, 204
UpdateLayout method, 205
update schedule, tiles, 224–225
updating tiles and tile content, 214–219
UserAway trigger, 34
user interaction
input devices, 191–201
capturing gesture library events, 192–195
custom gesture recognizers, 196–197
drag-and-drop events, 199–201
listening to mouse events or touch gestures, 198
stylus input and inking, 198–199
navigation, 202–212
loading HTML fragments, 210–212
meeting app requirements, 206–207
semantic zoom, 207–210
WinJS.Navigation namespace, 203–206
tiles, 214–225
creating and updating badges, 219–221
periodic notifications, 222–223
push notifications, 223–224
responding to notification requests, 221
scheduled notifications, 221–222
tile update schedule, 224–225
updating tiles and tile content, 214–219
toast notifications, 227–235
configuring Mobile Services for push notifications, 232–235
controlling toast duration, 232
enabling apps for, 227–229
populating with images and text, 229–230
responding to toast events, 231–232
user interface. *See* UI (user interface)
UserNotPresent condition, background tasks, 35
UserPresent condition, background tasks, 35
UserPresent trigger, 34

V

Value property, 167
verifying credentials, Credential Locker, 276–277
Version property, 253
video playback, 111, 115–116
Videos Library capability, 46
ViewManagement namespace, 167–168
vision, UI layout, 5–7
Visual Assets page (Manifest Designer), 44
Visual Studio, creating app packages, 41

W

WCF (Window Communication Foundation) service, consuming, 259–260
Weather app
app bars, 172
Settings charm, 74
Share charm, 73
web authentication, 282–287
CredentialPicker class, 285
credential roaming, 285
OAuth2, 283–284
proxy authentication, 286–287
SSO (Single Sign-On), 284–285
WebAuthenticationBroker class, 285
Windows.Security.Authentication.Web namespace, 282–283
WebAuthenticationBroker, 283
WebAuthenticationBroker.authenticateAsync method, 283
WebAuthenticationBroker class, 285
Webcam capability, 46
WebGL, 110
web links, sharing data, 93–95
Web Proxy Auto-Discovery (WPAD), 286
web services, retrieving with XHR or HTTPClient, 252–255
Websites property (Contact class), 66
WebSockets, bidirectional communication, 260–261
WebView control, 138–140
win as one design principle, 4
Window Communication Foundation (WCF) service, consuming, 259–260

windowing modes, creating layout-aware apps, 162–169
    CSS3 media queries, 162–165
    responding to changes in orientation, 165–166
    settings for an apps view, 168–170
    ViewManagement namespace, 167–168
Windows 8 certification requirements, 40–43
Windows App Certification Kit, 42–44
Windows App Certification Kit dialog box, 43
Windows.Applicationmodel. Contacts namespace, 58
Windows Dev Center page (WebView control), 138
Windows.Devices.Sensors.SimpleOrientationSensor.getDefault() method, 165
Windows.Media.SpeechSynthesis namespace, 113
Windows.Media.SpeechSynthesis.SpeechSynthesizer.allVoices property, 114
Windows.Media.Transcoding namespace, 112
Windows Push Notification Services (WNS), 223
Windows.Security.Authentication.Web namespace, 282–283
Windows.Security.Credentials.UI namespace, 274
Windows Store apps
    design
        planning app deployment, 40–49
        PLM (Process Lifetime Management), 26–38
        separation of concerns, 17–23
        UI layout and structure, 1–14
    development
        accessing and displaying contacts, 57–67
        charms and contracts, 69–79
        integration of media features, 109–115
        managing application setting s and preferences, 97–107
        Search charm, 78–85
        sharing data, 87–95
Windows.UI.Input.GestureRecognizer namespace, 196
Windows.UI.Input.Inking.InkManager class, 198
Windows.UI.Notification namespace, 217
Windows.UI.StartScreen.SecondaryTile class, 218
Windows.UI.ViewManagement.ApplicationView.getForCurrentView() method, 167, 168
Windows.UI.WebUI.WebUIBackgroundTaskInstance.current property, 36
WinJS.Application object, 32
WinJS.Application.onsettings event, 98
WinJS.Application.sessionState object, 31
WinJS.Binding.as method, 155, 264
WinJS.Binding.List class, 267
WinJS.Binding.optimizeBindingReferences property, 154
WinJS.Binding.processAll method, 155
WinJS.Binding.Template, 266
WinJS.Class.define method, 20
WinJS controls, 125–145
    FlipView, 126–130
    flyout, 130–132
    Grid and List layout, 132–135
    item containers, 140–142
    menu objects, 135–137
    Repeater, 142–145
    WebView, 138–140
WinJS.Namespace.define method, 19
WinJS.Navigation.back method, 206
WinJS.Navigation namespace, 203–206
WinJS.Navigation.navigate method, 206
WinJS.UI.AppBar control, 173
WinJS.UI.BackButton, 205
WinJS.UI.Fragments namespace, 210
WinJS.UI.Pages.define method, 204
WinJS.UI.processAll method, 126
WinJS.xhr method, 252
WinMD Components, incorporation, 22–23
WNS (Windows Push Notification Services), 223
WPAD (Web Proxy Auto-Discovery), 286
writeBytesAsync method, 245

X

XHR, retrieving web services, 252–255
XMLHttpRequest object, 252
XML templates
    support for tiles, 216
    toasts, 229
XVP (Media Foundation Transcode Video Processor), 112

Y

YAGNI (You Aren't Gonna Need It) principle, 10
YomiDisplayName property (Contact class), 66
YomiFamilyName property (Contact class), 66

YomiGivenName property (Contact class), 66
You Aren’t Gonna Need It (YAGNI) principle, 10

Z

z-index property, 150
zoomedInListView, 209
zoomedOutListView, 209
ZoomFactor, 209
zooming, CSS3, 156–158, 182

About the author

WOUTER DE KORT was born in a little place in the Netherlands called Grootebroek (literal translation: large pants!) in 1986. He started playing with software development when he was seven years old, when his dad came home with their first computer. Wouter works with C# and .NET on a daily basis and has done so since their inception. He is now a software architect focusing on Application Lifecycle Management for everything that runs on the Microsoft stack. As a Microsoft Certified Trainer, he loves helping companies stay on the cutting edge of software development, solving complex problems, and teaching others how to become better developers.