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

This page intentionally left blank

Contents

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

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.

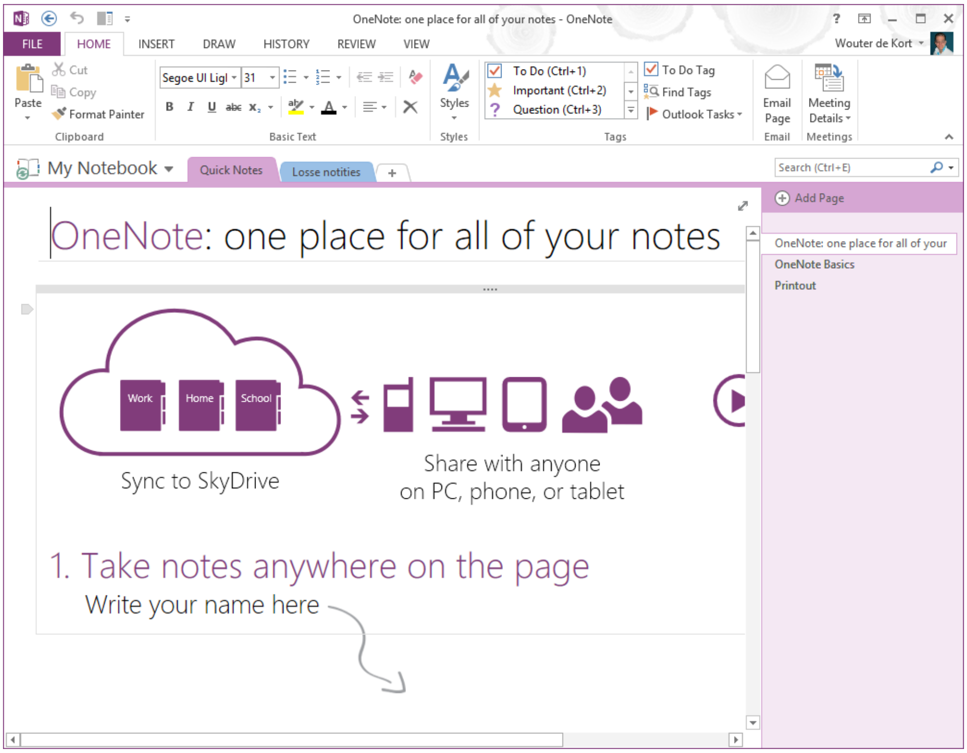
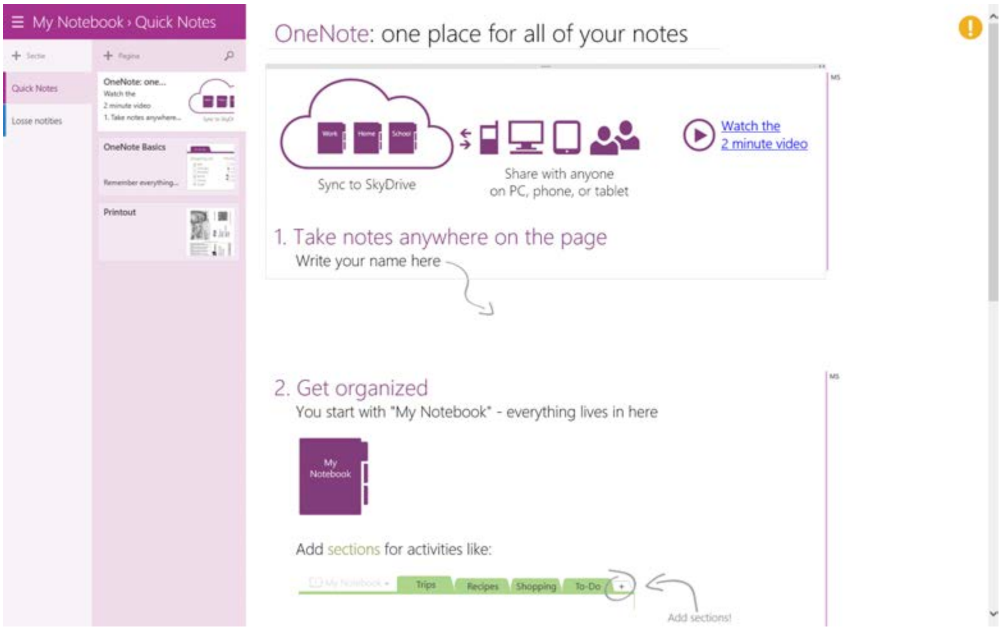


FIGURE 1-1 OneNote 2013 as a desktop app



OneNote: one place for all of your notes

1. Take notes anywhere on the page

Write your name here

2. Get organized

You start with "My Notebook" - everything lives in here

Add sections for activities like:

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.

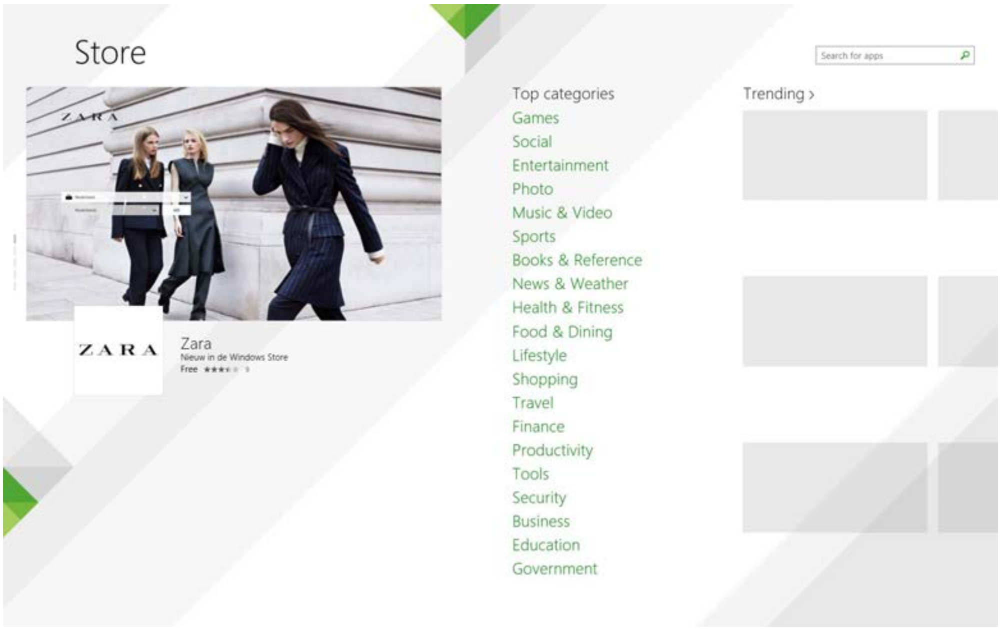


FIGURE 1-3 A Windows Store app showing the search box

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.

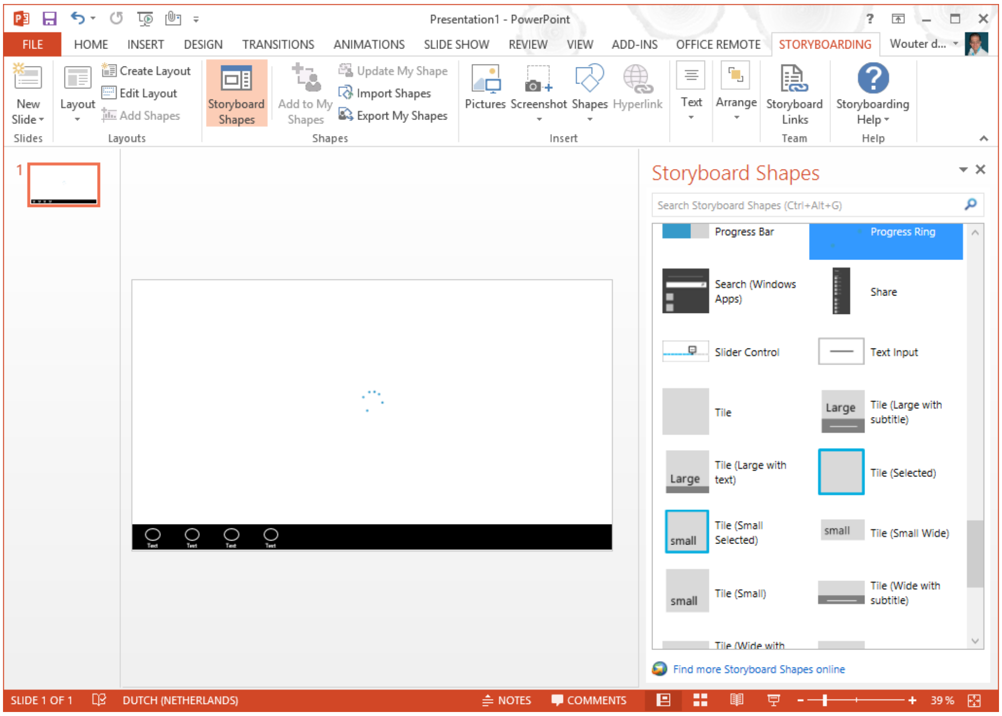


FIGURE 1-4 PowerPoint storyboarding for Windows Store apps

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.

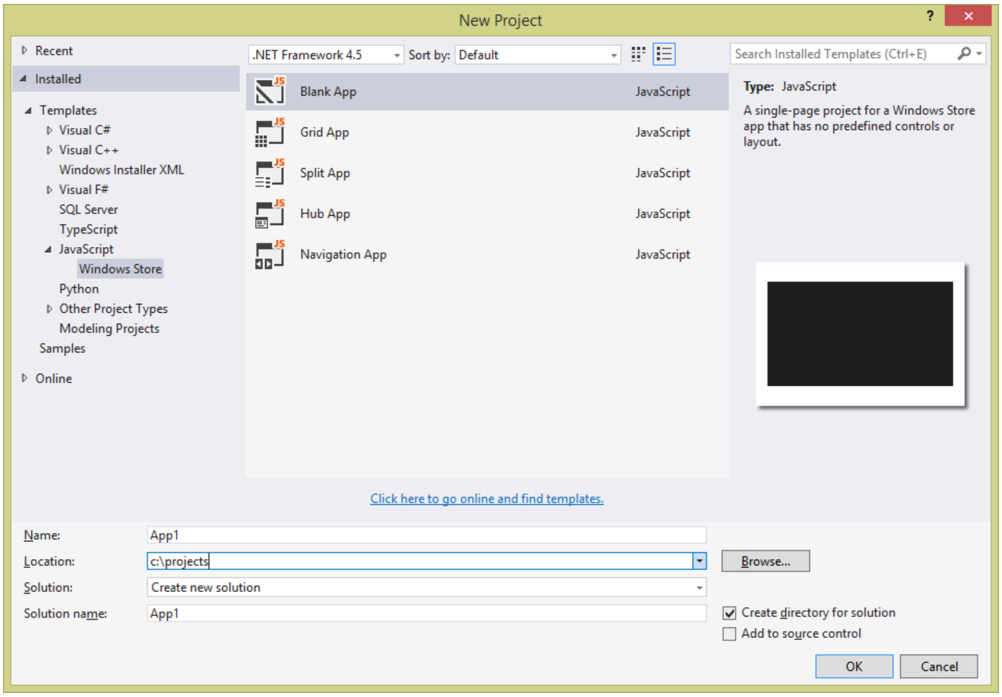


FIGURE 1-5 Creating a new JavaScript Windows Store app

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.

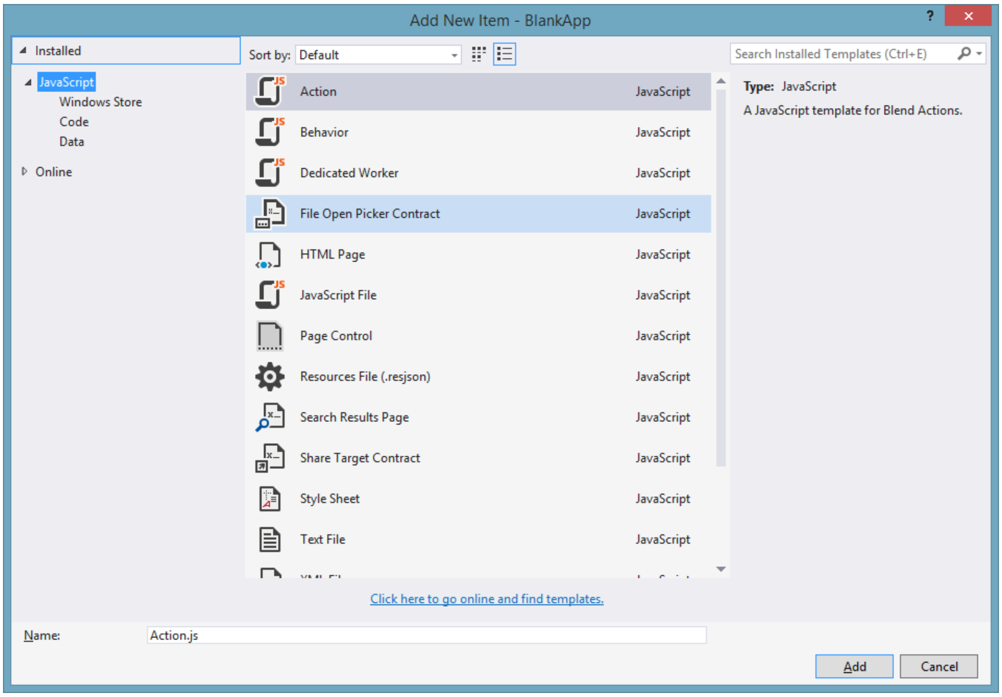
EXAM TIP

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.

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.

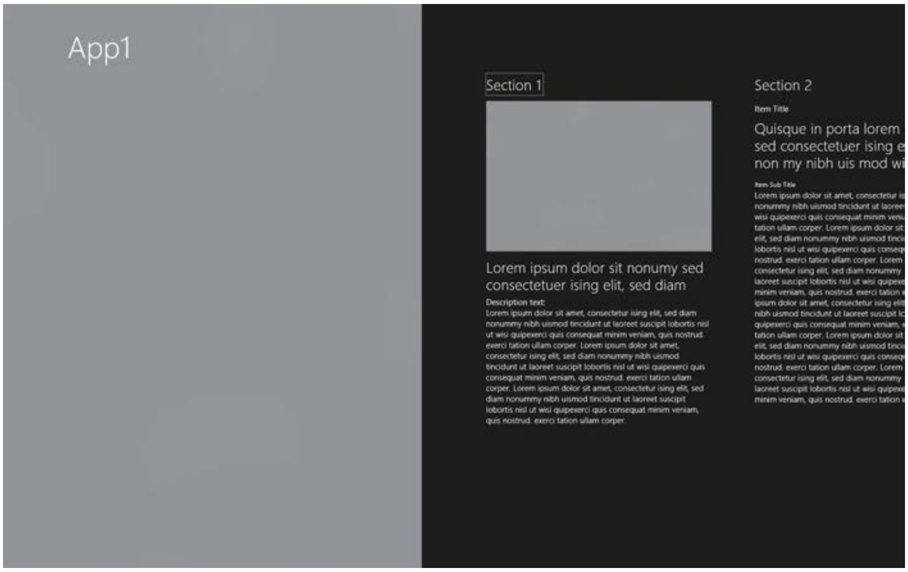
Using the Hub App template

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.

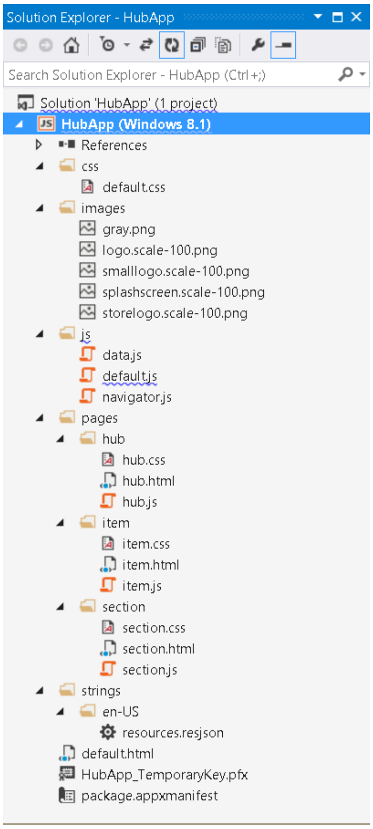
Figure 1-8 shows the initial files created by the template. The most interesting part is the Pages folder, which contains the Hub, Item, and Section pages. The data for Section and Item pages is loaded from the static data.js file in the js folder.



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.

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.

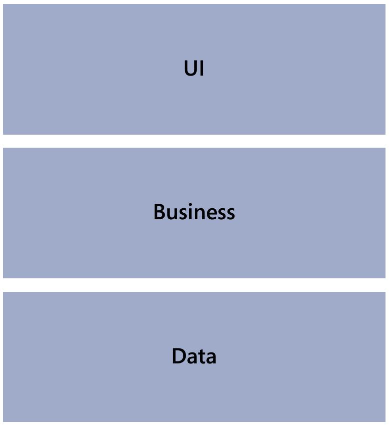


FIGURE 1-9 Diagram showing layering architecture

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.

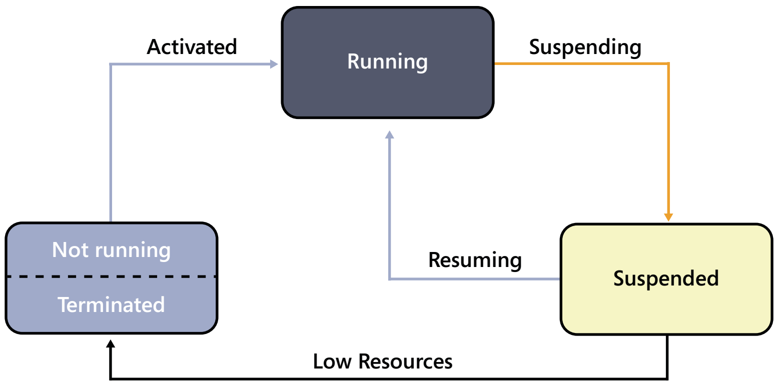


FIGURE 1-10 The life cycle of an app

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());
    }
};
```

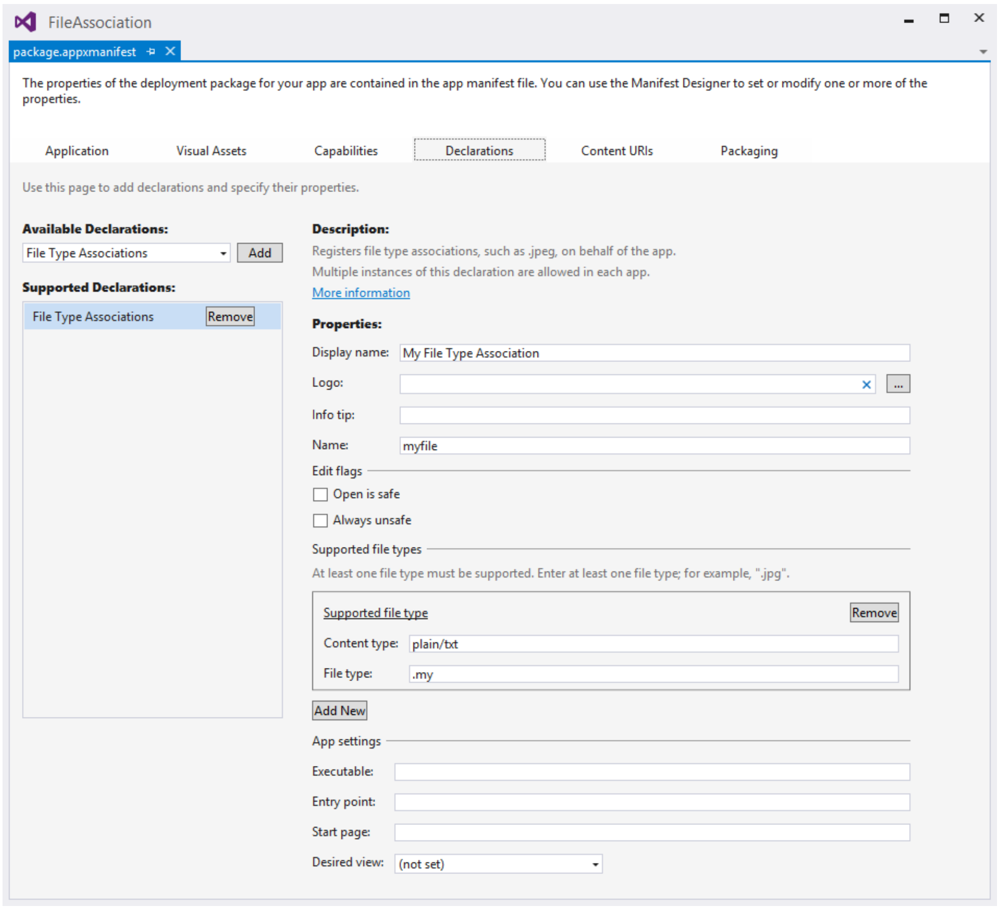
The code shows how to subscribe to the onactivated event of your WinJS application. Inside the event handler, you can see whether your app is newly launched or you are resuming from a suspended state.

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.


FileAssociation

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.

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.
More information

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:

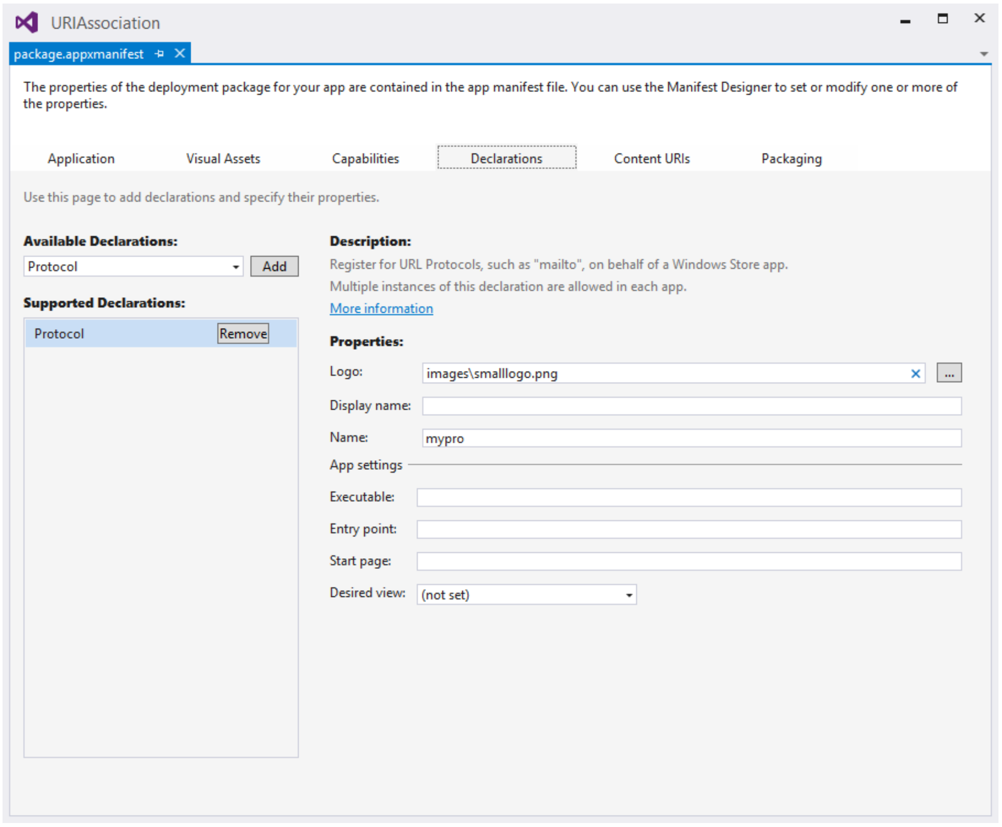
```
if (args.detail.kind === Windows.ApplicationModel.Activation.ActivationKind.file) {
    var file = args.detail.files[0];
    Windows.Storage.FileIO.readTextAsync(file).then(function (text) {
    });

    // 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.



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: Protocol

Supported Declarations: Protocol

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.

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

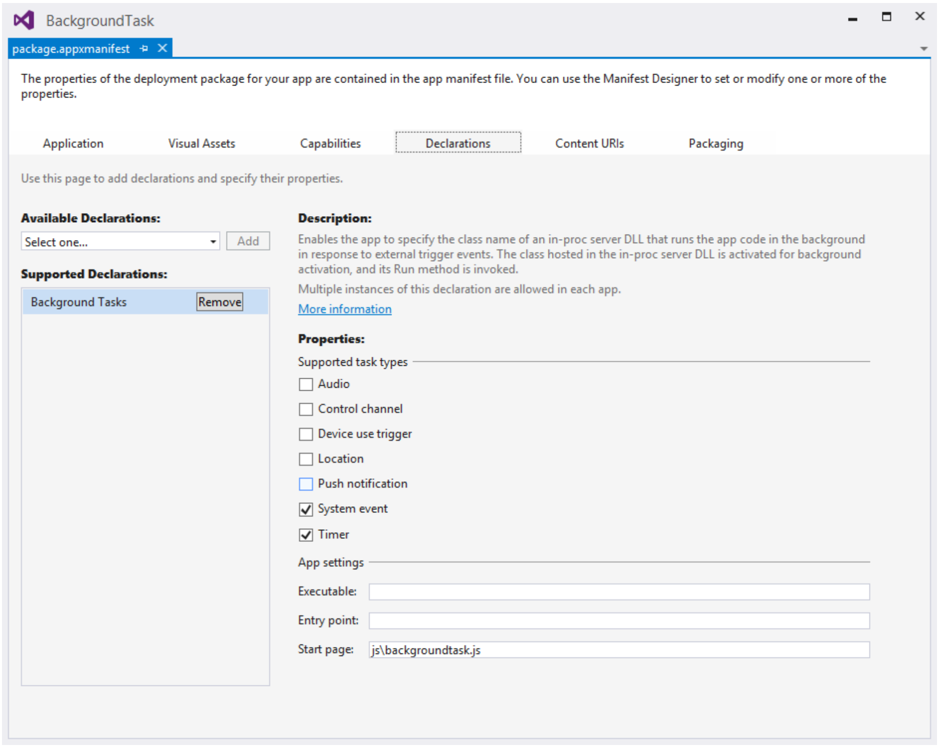
Remember that there is no separate event for termination. You should save all state in the checkpoint event and make sure that your app can completely recover from termination.

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:

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
- [✓] System event
- [✓] 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*.

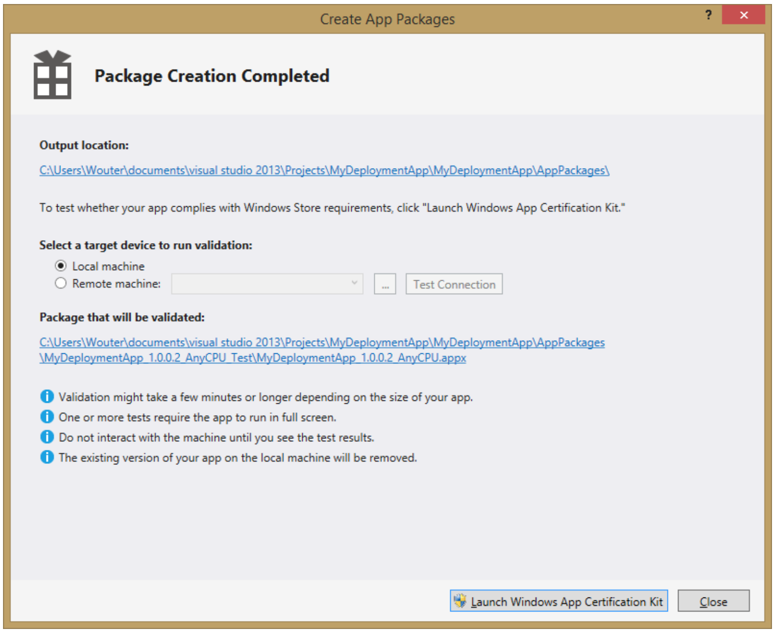
Some steps in this document are obvious. You shouldn't submit an app that doesn't work, doesn't add any value, or is not branded. You are not allowed to hack the system. Trying to communicate with other apps or loading remote scripts is forbidden. Be careful with privacy-related data by giving the user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


Create App Packages

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:

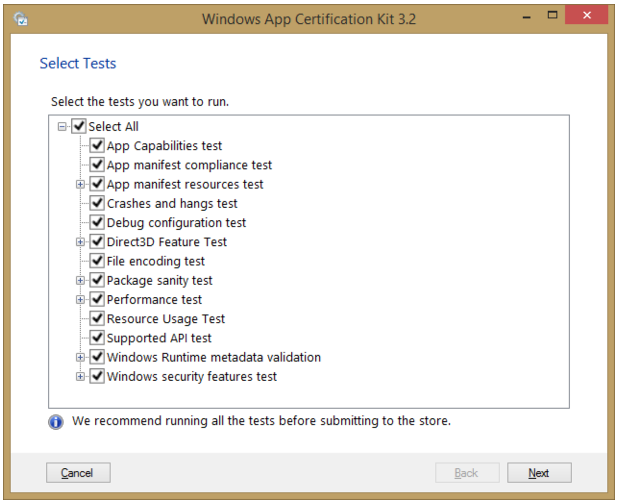
Package that will be validated:

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.


Windows App Certification Kit 3.2

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*.

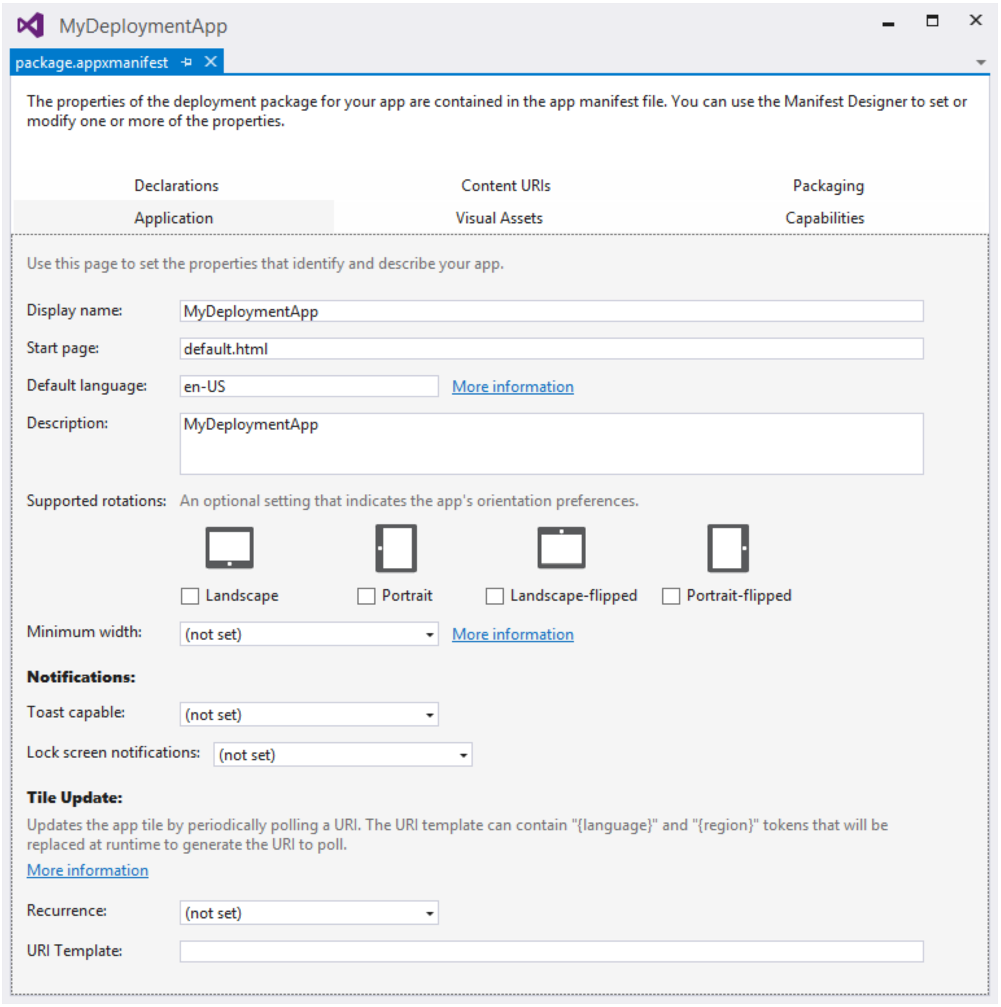
Preparing an app manifest (capabilities and declarations)

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 More information

Description: MyDeploymentApp

Supported rotations: An optional setting that indicates the app's orientation preferences.

☐ Landscape ☐ Portrait ☐ Landscape-flipped ☐ Portrait-flipped

Minimum width: (not set) More information

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**

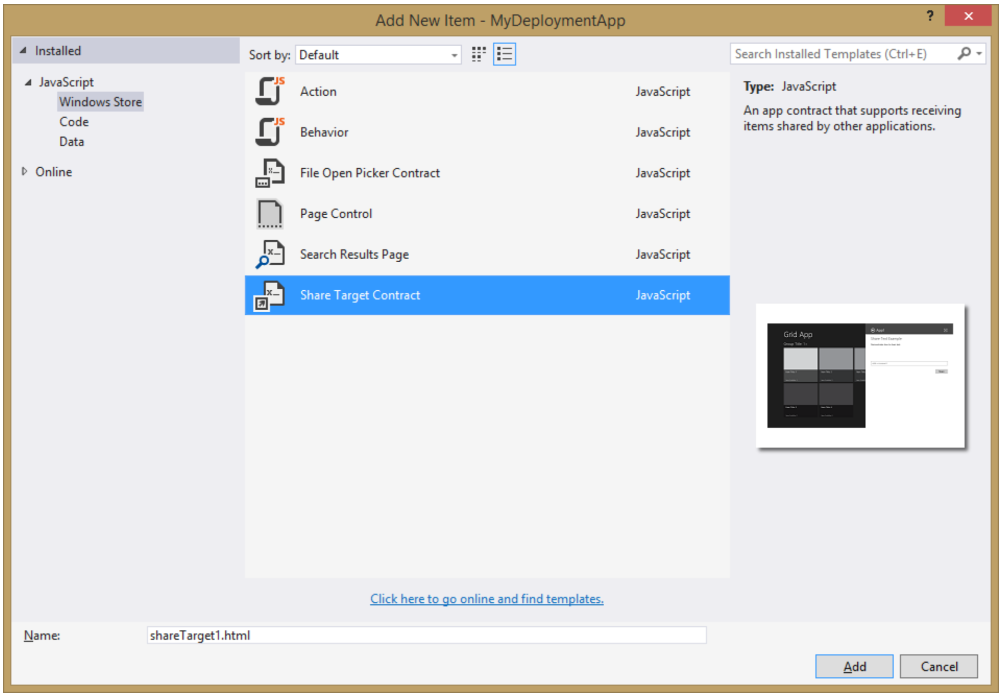
Contracts and extensions are discussed in more detail in Chapter 2.

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
Type: JavaScript
An app contract that supports receiving items shared by other applications.
Click here to go online and find templates.
Name: shareTarget1.html
Add
Cancel

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.

CHAPTER 2

Develop Windows Store apps

After you design your app, you want to start developing it. The Windows Store app environment is unique because of the integration options it offers. Your app can work together with the Windows operating system and with other apps through contracts and extensions, which is the topic of this chapter.

This chapter helps you integrate with contact information, search, share, and configuration settings. Implementing these features helps users find your app and make it easier to use. The result is more usage of your app and a more popular app overall! The chapter ends by discussing media features that your app can use.

This chapter covers the second objective in the Exam 70-481 objective domain and it consists of 19 percent of the exam's material. This chapter is a lot more code-heavy than the previous chapter and will help you to implement contract-related features in Windows Store apps.

Objectives in this chapter:

- Objective 2.1: Access and display contacts
- Objective 2.2: Design for charms and contracts
- Objective 2.3: Implement search
- Objective 2.4: Implement Share in an app
- Objective 2.5: Manage application settings and preferences
- Objective 2.6: Integrate media features

Objective 2.1: Access and display contacts

One way to integrate with other apps and Windows is by sharing contact information. You can support scenarios in which users can pick contacts from other apps (such as the People app) or your app can show contact information that other apps can use.

This objective shows you how to implement these scenarios.

This objective covers how to:

- Call the ContactPicker (Windows.Applicationmodel.Contacts) class
- Filter which contacts to display
- Display a set number of contacts
- Create and modify contact information
- Select specific contact data

Calling the ContactPicker class

You can find all functionality for dealing with contacts in the Windows.Applicationmodel. Contacts namespace. The ContactPicker class, which is defined in this namespace, enables users to select contacts that you can use inside your app.

The following function shows how to launch the ContactPicker class and create a string representation of a user's data:

```
function pickContact() {
    var picker = Windows.ApplicationModel.Contacts.ContactPicker();
    picker.commitButtonText = "Select";

    picker.pickContactAsync().done(function (contact) {
        if (contact != null) {
            document.getElementById('output').innerText = contact.displayName;
        }
    });
};
```

Start by creating a new instance of the ContactPicker class:

```
    var picker = Windows.ApplicationModel.Contacts.ContactPicker();
```

You can then configure the instance and show it to the user. In this case, the only thing you configure is the text the contact picker displays:

```
    picker.commitButtonText = "Select";
```

All work that might take a large amount of time is processed asynchronously, which helps your app stay responsive while external work is being done. You can recognize asynchronous methods by their names: they end with "async". This naming convention is also true for the contact picker. The pickContactsAsync method is processing asynchronously, which means the method doesn't return a result immediately. Instead, it returns a *promise*, which is an object that wraps the asynchronous action and helps you continue when the asynchronous action is finished or when something goes wrong.

In this case, use the done method to schedule some work that needs to process when the promise completes:

```
    picker.pickContactAsync().done(function (contact) { … });
```

The contact parameter is passed from the contact picker to your function. Perhaps the contact is null, which happens whenever the user cancels the contact picker and returns to your app without selecting a contact.

You can also select multiple contacts by using the pickContactsAsync function, which returns an array of contacts that you can use:

```
picker.pickContactsAsync().done(function (contacts) {
    if (contacts.length > 0) {
        var output = "";
        for (var i = 0; i < contacts.length; i++) {
            output += contacts[i].displayName;
            output += "\r\n";
        }
        document.getElementById('output').innerText = output;
    }
});
```

MORE INFO **CONTACT PICKER SAMPLES**

Microsoft created a lot of Windows Store samples that you can download. You can find a contact picker sample at *http://code.msdn.microsoft.com/windowsapps/Contact-Picker-App-sample-fc6677a1*.

Filtering which contacts to display

When using the contact picker to let users select contacts for your app, you normally get all contacts returned that are available for the user.

Sometimes your app needs to make sure that certain data is available for a contact. When you want to select only those contacts that have data in the specific fields you require, you can configure your contact picker to look for those fields.

By using the selectionMode property, you can configure the contact picker to select all contacts or only contacts with the fields that you require:

```
// Select specific fields
picker.selectionMode = Windows.ApplicationModel.Contacts.ContactSelectionMode.fields;
// Select all contacts
picker.selectionMode = Windows.ApplicationModel.Contacts.ContactSelectionMode.contacts;
```

When you use the fields option, you can use the desiredFieldsWithContactFieldType property to list the fields on which you want to filter:

```
picker.desiredFieldsWithContactFieldType.append(
     Windows.ApplicationModel.Contacts.ContactFieldType.email);
```

ContactFieldType can have one of the following values:

- Email
- PhoneNumber

- ConnectedServiceAccount
- ImportantDate
- Address
- SignificantOther
- Notes
- Website
- JobInfo

Other values, such as instantMessage, location and custom, are still available but shouldn't be used. They might not be available in the next Windows release.

Specifying the ContactFieldType gives you only those contacts that have a specific field set. After selecting contacts from the contact picker, you can apply additional filters to keep only the contacts you want to use in your app. Maybe you want to filter on the company name or some other criterion that's appropriate for your app.

EXAM TIP

Using ContactFieldType does not mean that you load only the specified field from the contact picker; you show only contacts that have a value for that field.

Because the data you get back from the contact picker is just an array, you can use array.filter to filter your contacts before displaying them:

```
contacts = contacts.filter(function (element)
{
    return element.displayName.indexOf("a") != -1
});
```

This snippet filters your contacts to only those that contain the letter "a" in their name. The function used to filter your data is called a *predicate*. You can make your predicate as complex as needed.

If you have a scenario in which data is changing more frequently (users select contacts multiple times, for example), you can use a WinJS.Binding.List object to store your data. A list can have a custom filter applied to it that is used whenever the data is updated.

The following code shows how to create a list that is filtered with the same predicate as the array in the previous example:

```
var itemList = new WinJS.Binding.List([]);
var filteredList = itemList.createFiltered(function (i) { return i.indexOf("a") != -1;
});
var publicMembers =
            {
                itemList: filteredList
            };
WinJS.Namespace.define("DataExample", publicMembers);
```

The list starts out empty; then a filtered version of the list is created. This list is then exposed through a custom namespace named DataExample.

You can bind to this list with the following HTML snippet:

```
<div id="basicListView" data-win-control="WinJS.UI.ListView"
     data-win-options="{itemDataSource : DataExample.itemList.dataSource}">
</div>
```

The data source for the list points to your custom created namespace and then the itemList property. Whenever a contact is added, you can use the following code in your pickContactAsync callback:

```
picker.pickContactAsync().done(function (contact) {
    if (contact != null) {
        itemList.push(contact.displayName);
    }
});
```

Whenever you push a new contact name to the list, the filter is applied, and the result is bound to the ListView in your markup. However, filtering after users select contacts might lead to unexpected behavior if they select a contact that then gets filtered when they return to the app. When dealing with these scenarios, inform users of your choices.

You can't directly use the contact object and push it to your item list; you should use a not-WinRT object to add to your list. A not-WinRT object can be a simple string such as the displayName or an object that you create.

Displaying a set number of contacts

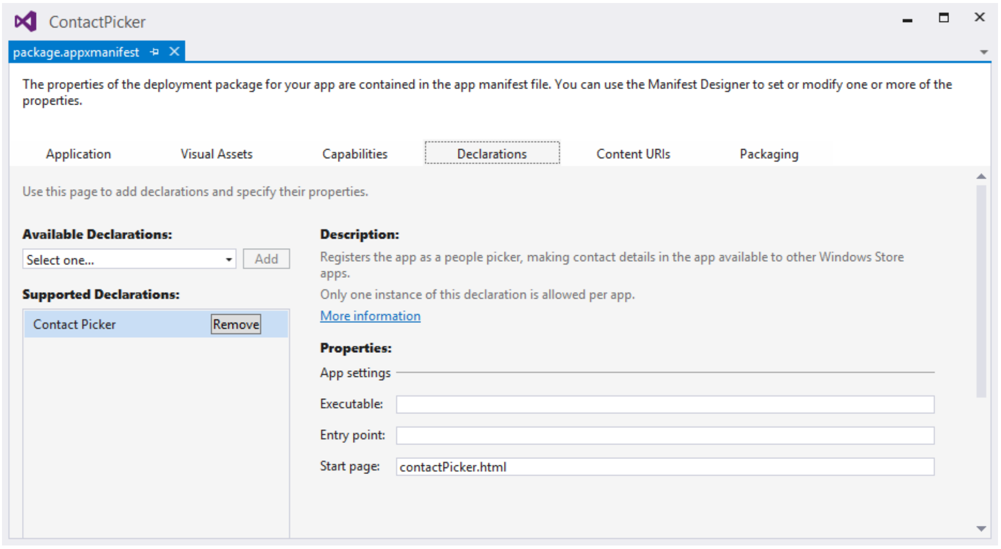
Until now, you used the ContactPicker class to select contacts and use the data in your app. Other apps offer you a set of contacts—and you use only the resulting data.

The People app is a good example of this type of app. If you launch it directly from the Start screen, it shows you a hub interface with information about you, your favorite contacts, and all other contacts you have.

But if you use the People app to select a contact from another app (through the ContactPicker class), you get a different interface because the app is then launched with ActivationKind. This different ActivationKind allows the app to show a different interface and helps users select contacts from the app.

By configuring the manifest file, you can add a declaration for your app to be a contact picker (see Figure 2-1).

The contact picker can be configured to use a Start page when your app is launched as a contact picker. Windows loads this page and shows it to users with a header that allows them to switch to other contact pickers and click a Select button or a Cancel button at the bottom (see Figure 2-2).



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... Add

Supported Declarations:

Contact Picker Remove

Description:

Registers the app as a people picker, making contact details in the app available to other Windows Store apps.

Only one instance of this declaration is allowed per app.

More information

Properties:

App settings

Executable:	
Entry point:	
Start page:	contactPicker.html

FIGURE 2-1 The App Manifest Designer shows the ContactPicker declaration

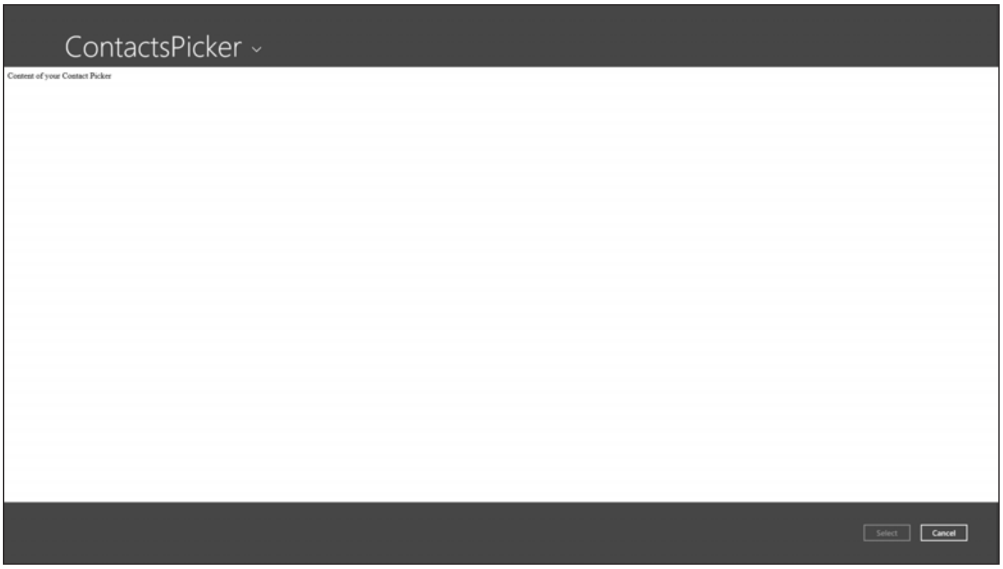


FIGURE 2-2 A custom contact picker showing a blank page

Suppose that you have the following content for your contactPicker.html page:

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact picker sample</title>

    <link rel="stylesheet" href="//Microsoft.WinJS.2.0/css/ui-light.css" />
    <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

    <script src="/js/contactPicker.js"></script>
</head>
<body>
    <div id="contactList"></div>
</body>
</html>
```

This is a basic HTML page that references the required cascading style sheets (CSS) and scripts for a Windows Library for JavaScript (WinJS) application and loads the contactPicker.js file.

The contactPicker.js file looks like this:

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

    function activated(eventObject) {
        if (eventObject.detail.kind === Windows.ApplicationModel.Activation.
                                        ActivationKind.contactPicker) {
            sampleContacts.forEach(createContactUI);
            contactPickerUI = eventObject.detail.contactPickerUI;
        }
    }

    function createContactUI(sampleContact, index) {
        var element = document.createElement("div");
        document.getElementById("contactList").appendChild(element);
        element.innerHTML = "<div class='contact'><label>" +
                            "<input id='" + sampleContact.id +
                              "' value='" + index + "' type='checkbox' />" +
                            sampleContact.displayName + "</label></div>";

        element.firstElementChild.addEventListener("change", function (ev) {
            if (ev.target.checked) {
                addContactToBasket(sampleContact);
            } else {
                removeContactFromBasket(sampleContact);
            }
        }, false);
    }
    function addContactToBasket(sampleContact) {
        var contact = new Windows.ApplicationModel.Contacts.Contact();
        contact.firstName = sampleContact.firstName;
        contact.lastName = sampleContact.lastName;
        contact.id = sampleContact.id;

        contactPickerUI.addContact(contact);
    };

    function removeContactFromBasket(sampleContact) {
        if (contactPickerUI.containsContact(sampleContact.id)) {
            contactPickerUI.removeContact(sampleContact.id);
        }
    }
    var sampleContacts = [
        {
            displayName: "Wouter de Kort",
            firstName: "Wouter",
            lastName: "de Kort",
            id: "60666FA6-7A68-4F78-B3CE-6CBBC436E8CE"
        },
         {
             displayName: "Satya Nadella",
             firstName: "Satya",
             lastName: "Nadella",
             id: "60666FA6-7A68-4F78-B3CE-6CBBC436E8CE"
         },
    ];

    WinJS.Application.addEventListener("activated", activated, false);
    WinJS.Application.start();
})();
```

This file has a couple of important parts:

- Sample data at the end of the file is some sample data defined. This data can come from an external web service, a file, or some other resource.

  The activated event is where your UI is created when the page loads. This event is activated by registering the event and calling WinJS.Application.start at the bottom of the file. The event handler loops through the sample data and generates HTML that shows each contact name and a check box that allows the user to select and deselect the contact.

  The add event handler is the place where a new Windows.ApplicationModel.Contacts.Contact is created and initialized with the values from the selected contact. This contact is then added to the selection by calling contactPickerUI.addContact(contact).
- The remove event handler sees whether the contact is present in the contactPickerUI and then removes it from the selection.

  Those parts form the basis of implementing your own contact picker. Of course, you can put a lot more data into a contact and send it to the requesting app. You should also add error handling in case something goes wrong.

EXAM TIP

The previously mentioned sample (*http://code.msdn.microsoft.com/windowsapps/Contact-Picker-App-sample-fc6677a1*) also shows how to implement a custom contact picker. You should use this sample to become more familiar with how to create your own contact picker. The code in this objective is based on this sample.

Creating and modifying contact information

The previous section described using the Contact class from the Windows.ApplicationModel. Contacts namespace to create a contact that can be sent through the contact picker.

You can add a lot more data to a contact. The Contact class has the following properties:

- **Addresses** Gets the contact addresses for a contact.
- **ConnectedServiceAccounts** Gets the connected service accounts for a contact.
- **DataSuppliers** Gets the data suppliers for a contact. The maximum string length for each data supplier is 50 characters. Data suppliers are external sources such as Twitter, LinkedIn, and Outlook. Data from those sources gets combined into one single contact on the user's device.
- **DisplayName** Gets the display name for a contact. You can access this property only from a UI thread.
- **Emails** Gets the email addresses for a contact.
- **FirstName** Gets and sets the first name for a contact. The maximum string length for the first name is 64 characters.
- **HonorificNamePrefix** Gets and sets the honorific prefix for the name for a contact. The maximum string length for the honorific prefix is 32 characters. Examples of honorific name prefixes are academic titles, but also titles such as Your Highness and Your Majesty.
- **HonorificNameSuffix** Gets and sets the honorific suffix for the name for a contact. The maximum string length for the honorific suffix is 32 characters.
- **Id** Gets and sets the identifier for a contact. The maximum string length for the identifier is 256 characters.
- **ImportantDates** Gets the important dates for a contact.
- **JobInfo** Gets the job info items for a contact.
- **LastName** Gets and sets the last name for a contact. The maximum string length for the last name is 64 characters.
- **MiddleName** Gets and sets the middle name for a contact. The maximum string length for the middle name is 64 characters.
- **Notes** Gets and sets notes for a contact. The maximum string length for notes is 4,096 characters.

- **Phones** Gets information about the phones for a contact.
- **ProviderProperties** Gets the property set object for the contact.
- **SignificantOthers** Gets the significant others for a contact.
- **Thumbnail** Gets or sets a thumbnail image that represents a contact.
- **Websites** Gets the website for a contact.
- **YomiDisplayName** Gets the Yomi (phonetic Japanese equivalent) display name for a contact.
- **YomiFamilyName** Gets the Yomi (phonetic Japanese equivalent) family name for a contact. The maximum string length for a Yomi family name is 120 characters.
- **YomiGivenName** Gets the Yomi (phonetic Japanese equivalent) given name for a contact. The maximum string length for the Yomi given name is 120 characters.

A contact is much more than just a name. Addresses, phone numbers, and other communication details can all be linked to a contact. Contact details can be further divided into different types, such as work and personal details. You even have specific fields for dealing with different cultures.

The following code fragment shows how to add additional data to the Contact class:

```
var contact = new Windows.ApplicationModel.Contacts.Contact();
contact.firstName = "Satya";
contact.lastName = "Nadella";
contact.id = "861cb6fb-0270-451e-8725-bb575eeb24d5";

var workEmail = new Windows.ApplicationModel.Contacts.ContactEmail();
workEmail.address = "ceo@xxx.com";
workEmail.kind = Windows.ApplicationModel.Contacts.ContactEmailKind.work;
contact.emails.append(workEmail);

var workPhone = new Windows.ApplicationModel.Contacts.ContactPhone();
workPhone.number = "1234567890"
workPhone.kind = Windows.ApplicationModel.Contacts.ContactPhoneKind.work;
contact.phones.append(workPhone);

var workAddress = new Windows.ApplicationModel.Contacts.ContactAddress();
workAddress.streetAddress = "1 157th Ave NE"
workAddress.locality = "Redmond";
workAddress.region = "Washington";
workAddress.postalCode = "98052";
workAddress.kind = Windows.ApplicationModel.Contacts.ContactAddressKind.work;
contact.addresses.append(workAddress);

contactPickerUI.addContact(contact);
```

Selecting specific contact data

When a contact is returned from the contact picker to an app, it contains all data available for that contact. So when a contact has multiple phone numbers, email addresses, and/or addresses, they are present in the returned data.

All these contact details specify their type of data through a kind property. By using the predefined enumerations that are part of the Windows application model, you can filter your data, as shown in the following example:

```
var result = "";

if (contact.phones.length > 0) {
    result += "Phone Numbers:";
    result += "\r\n";
    contact.phones.forEach(function (phone) {
        switch (phone.kind) {
            case Windows.ApplicationModel.Contacts.ContactPhoneKind.home:
                result += phone.number + " (home)";
                break;
            case Windows.ApplicationModel.Contacts.ContactPhoneKind.work:
                result += phone.number + " (work)";
                break;
            case Windows.ApplicationModel.Contacts.ContactPhoneKind.mobile:
                result += phone.number + " (mobile)";

                break;
            case Windows.ApplicationModel.Contacts.ContactPhoneKind.other:
                result += phone.number + " (other)";
                break;
        }
        result += "\r\n";
    });
}
```

In this example, you loop through all phone numbers on a contact. By using phone.kind you can easily handle different types of phone numbers. The same can be done for other data on a contact.

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 exploring the Windows Store ecosystem and are interested in brainstorming scenarios in which you can use contact picker integration.

List a type of app that can use the following:

1. **The contact picker to select contacts from other apps.**
2. **The contact picker to expose contacts to other apps.**

Objective summary

- The ContactPicker class can show a UI to users. They can then select one or more contacts to be used in your app.
- You can register your app as a contact picker so that apps (including your own) can view contacts and select them from your app.
- A user can have contacts that have some fields filled in and others left blank. You can filter contacts by using the selectionMode property and the desiredFieldsWithContactFieldType collection. Using such a filter shows only those contacts that have the fields you require.
- A contact has multiple phone numbers, addresses, and so on. You can filter those collections by using the kind property.

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 added code to your activated event to respond to the activation kind of contact picker, but your app doesn't show up as a possible source of contacts when launching the contact picker. What should you do to fix the problem?

   A. The user should have your app running in the background to serve as a contact picker.

   B. You have to call the activated event to launch the contact picker.

   C. You have to add a Contact Picker declaration to your app manifest.

   D. You have to use the pickContactAsync method to launch the contact picker.

2. You want to launch the contact picker from your app so a user can select one contact. Which method should you use?

   A. ContactPicker.addContact

   B. ContactPicker.pickContactAsync

   C. ContactPicker.pickContactsAsync

   D. ContactPicker.pickContact

3. You want to make sure that a user selects only those contacts who have email addresses. What can you do?

   A. Use array.filter to exclude those contacts who don't have an email address.

   B. Use a WinJS list with a filter that excludes contacts who don't have an email address.

   C. Use a ContactSelectionMode of fields with a desiredFieldsWithContactFieldType of email.

   D. Use a ContactSelectionMode of contact with a desiredFieldsWithContactFieldType of email.

Objective 2.2: Design for charms and contracts

A Windows Store app doesn't work in total isolation. The Windows operating system offers a complex ecosystem that your app can integrate with. This integration makes Windows Store apps unique and it uses the Microsoft design principle of "win as one": letting apps interact with each other in a familiar manner so they complement each other.

Integrating apps is done through charms and contracts. This objective shows you the available charms and contracts and describes how they can benefit your app. Understanding charms and contracts is an important part of the exam because it is one of the cornerstones of Windows 8.

This objective covers how to:

- Choose the appropriate charms based on app requirements
- Design an application to be charm- and contract-aware
- Configure the application manifest for correct permissions

Choosing the appropriate charms based on app requirements

Charms can be accessed by swiping from the left, moving your mouse to the top- or bottom-right corner, or pressing Windows+C. The charms bar that displays is shown in Figure 2-3.

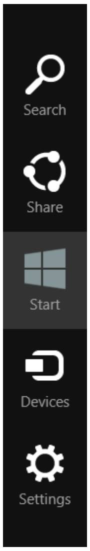


FIGURE 2-3 The charms bar shows the Search, Share, Start, Devices, and Settings charms

The five charms you see here are the following:

- Search
- Share
- Start
- Devices
- Settings

The Start charm, which is the least interesting, allows users to return to the Start screen. With this charm, users can always find a familiar location to return to the Start screen if they want to leave an app. The other charms offer more possibilities and are discussed in the following sections.

Search

Search is an important part of many apps. When you launch Search from the Start screen of Windows, the Search charm displays (see Figure 2-4).

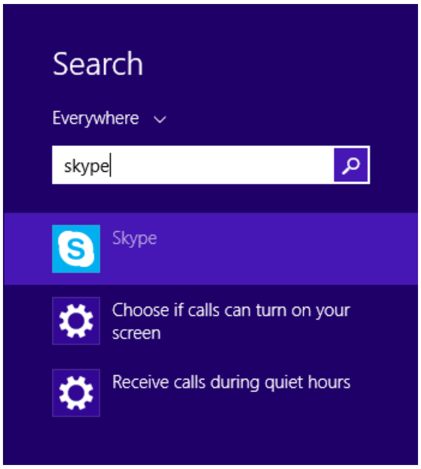

Search
Everywhere
skype
Skype
Choose if calls can turn on your screen
Receive calls during quiet hours

FIGURE 2-4 The Search charm

Search goes through all data on your PC, including installed applications and files, but also content found on the Internet.

This search is global. You can also implement an in-app search. For an in-app search, use the SearchBox control, which changed between Windows 8 and Windows 8.1. In Windows 8, the Search charm is used for in-app and global searches. In Windows 8.1 you can choose between using a search box, such as the one shown in Figure 2-5, or using the Search charm.

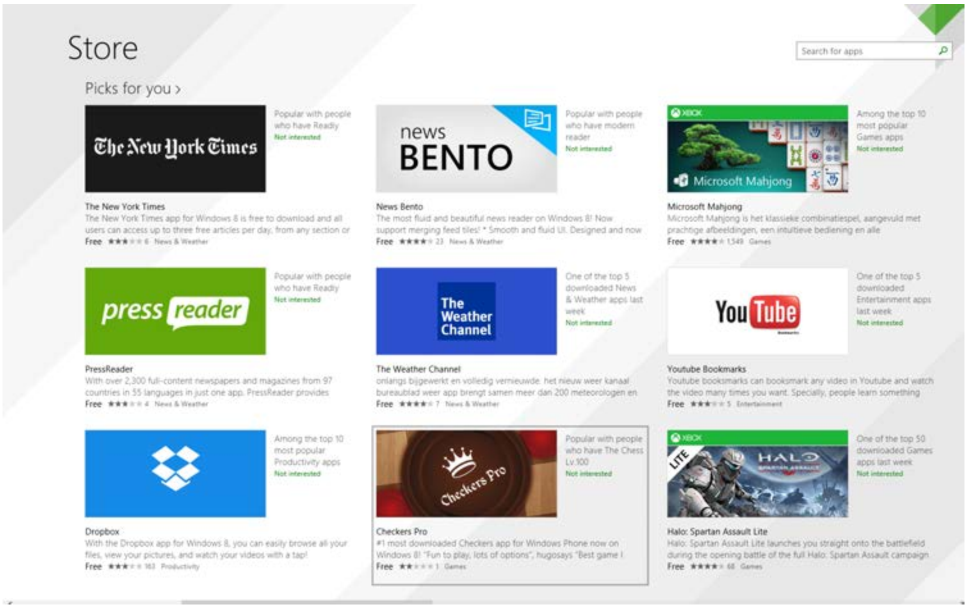


FIGURE 2-5 The Windows Store shows a search box in the top-right corner

Users can use this Windows Store search box to search for apps in the store. When they launch a search, a new page is shown that lists the search results (see Figure 2-6).

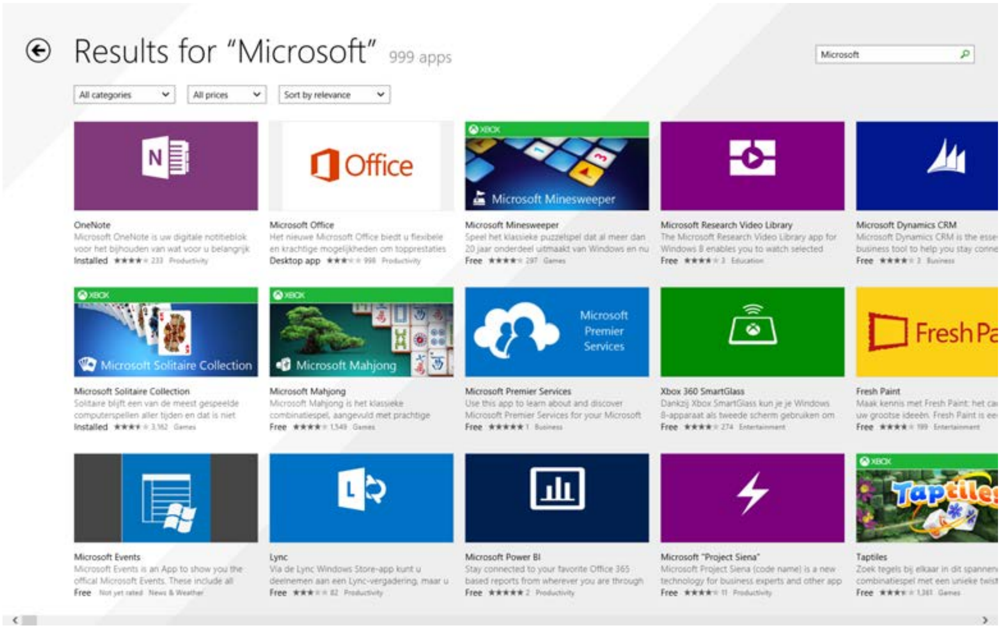


FIGURE 2-6 The Windows Store shows the results of searching on "Microsoft"

This page is completely specific to your app. You can define its layout, the way a user can interact with it, and the results it shows.

You can help users by offering them both query and result suggestions. The difference between them is shown in Figure 2-7. A result suggestion is a close match and immediately navigates to a result. A query suggestion is used more as an autocomplete in which you start a search for the specified query text.

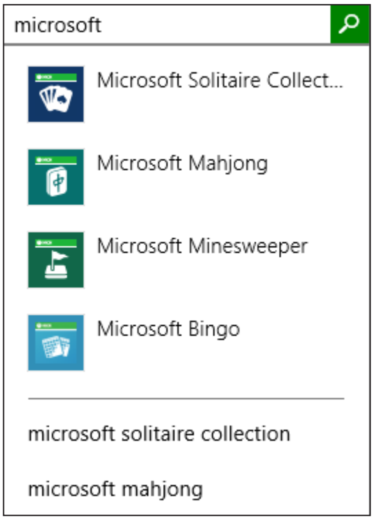


FIGURE 2-7 The search box shows result suggestions at the top and query suggestions at the bottom

Another feature that helps your users is *type to search*. A user can just start a new search simply by typing a search term; there is no need to type the term in the search box. Users can launch an app and immediately start typing to search for something.

By integrating with the Search charm, you can let users launch your app from the charm and directly start on the search results page.

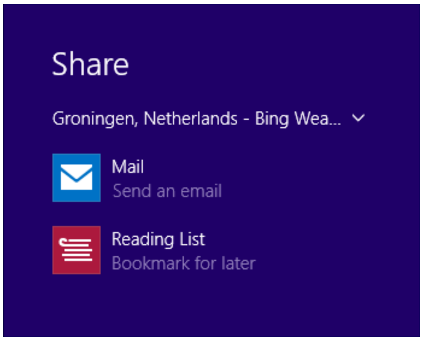
Share

The Share charm enables data sharing between apps. For example, you browse the web, see something you like, activate the Share charm, and send an email with a link to the page to a friend.

The Share contract supports these types of scenarios. Apps can be both a share source (the browser in this case) and a target (the email application). The source can provide the data that it wants to share, and the target can take this data and use it.

Typical examples of applications that act as a share target are Twitter, Facebook, and Mail; they take data and easily share it with other people.

Figure 2-8 shows an example of activating the Share charm from the Weather app. A list of possible share targets is shown that the user can choose from.



Share

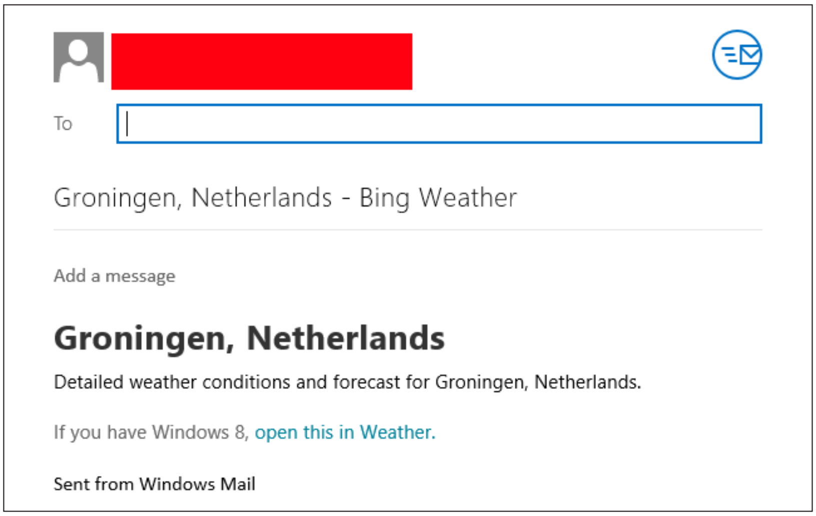
Groningen, Netherlands - Bing Wea...

Mail
Send an email

Reading List
Bookmark for later

FIGURE 2-8 Activating the Share charm from within the Weather app and showing possible share targets

After selecting the Mail option, you see the display shown in Figure 2-9.



[redacted]

To

Groningen, Netherlands - Bing Weather

Add a message

Groningen, Netherlands

Detailed weather conditions and forecast for Groningen, Netherlands.

If you have Windows 8, open this in Weather.

Sent from Windows Mail

FIGURE 2-9 Choosing the Mail share target to share data from within the Weather app

When considering the Share contract, you have to split the implementation into two parts. First, do you want to share any data from within your app? Suppose that you're building a Rich Site Summary (RSS) reader app. When users read something interesting, they can launch the Share charm to send the article to others. In a game, you can use the Share charm to post a recent achievement to Facebook. In a business app, you can share a report with other users.

The possibilities are endless, and you really need to consider it carefully. By integrating Share, you add a social component to your app that can really improve its popularity.

Second, you need to decide whether your app can be a share target. Can other apps share some data with your app? (This is true for social apps like Twitter and Facebook, or for email.) If your app can receive data (text, images, or files), your app can act as a share target.

Devices

Devices connected to your device can be accessed through the Devices charm. These devices might include a printer, a projector, or a TV for streaming media. By integrating with the Devices charm, you can support printing or streaming media to other devices.

Depending on your requirements, you can choose to integrate with the Devices charm. Maybe you have information that a user should be able to print, such as a summary page or a picture. If you have multimedia, such as audio or video, you can choose to stream them to other devices.

Settings

Microsoft created the Settings charm to create a uniform configuration experience across all Windows Store apps. Application–specific configuration options can be found in the Settings charm. Figure 2-10 shows an example of the Settings charm for the Weather app.

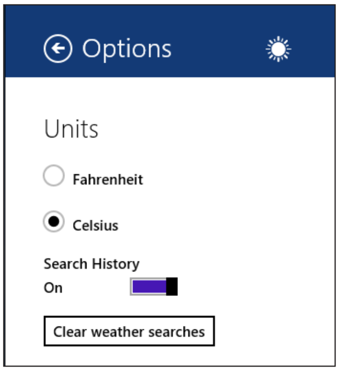


FIGURE 2-10 The Settings charm for the Weather app

You should use the Settings charm for all configuration settings for your app.

Designing an application to be charm- and contract-aware

When designing your application, be aware of the different charms that are available. You shouldn't put any elements in the UI of your app if they belong in one of the charms.

The Search charm has some clear design guidelines that tell you what to do. By using the search box in your app, preferably on all pages, you provide a clear location from which users can start their search. By combining query and result suggestions, you help users find what they are looking for.

The Share charm can greatly benefit your app. By thinking about the content of your app, you can look for possibilities to implement sharing to make sure that your app integrates well with other apps. This sharing can lead to higher app use when users start sending data to your app or importing data from it.

The Devices charm is useful whenever your app needs to connect to other devices. When you encounter such a requirement, you usually put this functionality in the Devices charm. You can add a specific print button to your UI only when the functionality is an essential or logical part of your application, such as printing an order confirmation at the end of an order process.

The Settings charm almost always should have a place in your app. All global settings should be placed in this charm. If you have settings that are specific to a page, you can place them in the app bar.

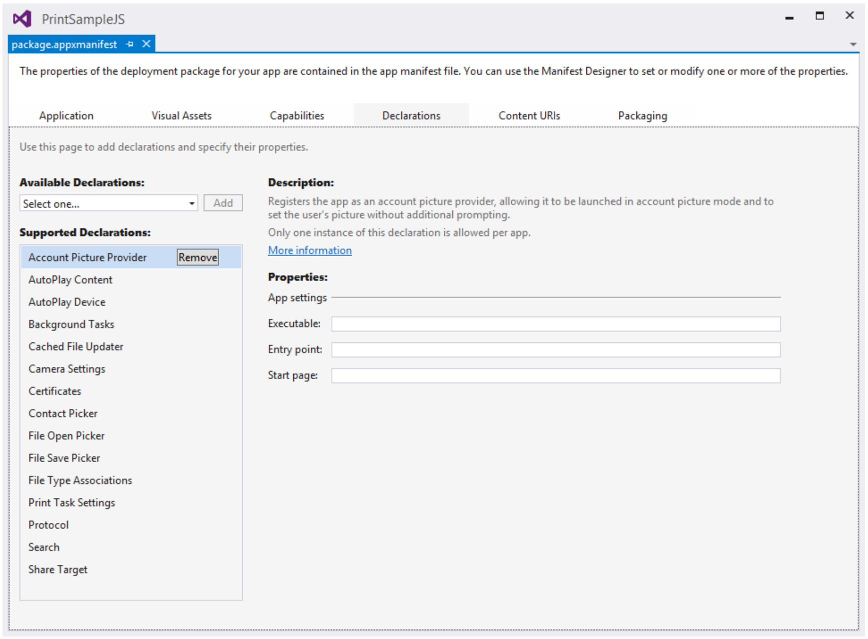
The remainder of this chapter focuses on implementing these charms so you get a better sense of their use and implementation. Remember for now that when you are thinking about a piece of functionality, you should always look at the charms and decide whether you can use them or whether you should implement your functionality in-app.

Configuring the application manifest for correct permissions

As mentioned in Chapter 1, "Design Windows Store apps," the app manifest contains settings that Windows can use to interoperate with your app. When your app wants to integrate with some external resource, such as the Internet or a charm, you need to specify it in the app manifest.

The requested permissions are shown to the user in the Windows Store, but are also used to validate your app when it is running. This validation step requires you to configure the app manifest whenever you start working with charms or contracts.

Figure 2-11 shows the App Manifest Designer in Visual Studio with a list of all possible declarations that you can add.



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:

Supported Declarations:

Account Picture Provider
AutoPlay Content
AutoPlay Device
Background Tasks
Cached File Updater
Camera Settings
Certificates
Contact Picker
File Open Picker
File Save Picker
File Type Associations
Print Task Settings
Protocol
Search
Share Target

Description:

Registers the app as an account picture provider, allowing it to be launched in account picture mode and to set the user's picture without additional prompting.

Only one instance of this declaration is allowed per app.

Properties:

App settings

Executable:

Entry point:

Start page:

FIGURE 2-11 The App Manifest Designer showing a list of declarations

Each declaration has its own custom settings that you need to set. Windows then knows how your app wants to integrate with other apps and can set the correct permissions for your app.

EXAM TIP

When you are asked to configure a charm or contract on the exam, don't forget to configure the application manifest.

Thought experiment

Using charms and contracts

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 exploring the Windows 8 ecosystem. You have some great app ideas, but you wonder whether using everything that Windows 8 has to offer is worth the trouble. With this in mind, think about the following questions:

1. **Why has Microsoft chosen to implement contracts and charms?**
2. **What are the advantages of using charms and contracts in your app?**
3. **Can you think of an existing application that would benefit from charm integration?**

Objective summary

- An important facet of Windows Store apps is "winning as one." Integration and cooperation with other apps is crucial for your app to succeed.
- By offering contracts and charms, Microsoft created a flexible yet familiar way to implement common tasks such as search, share, settings, and devices.
- By using the application manifest, you can configure your application to request permissions to use charms and contracts.

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 a ToDo app that keeps lists of tasks and you want to share these task lists with other people. Should you implement the Share charm?
   - **A.** No; it is easier to use a Share button directly on the pages that view task lists.
   - **B.** No; you implement the Devices charm to send data through email.
   - **C.** Yes; users can easily share task lists from within your application to other applications.
   - **D.** Yes; users can share tasks through email or can print them.
2. Your app contains a lot of data that is divided in categories and items and you want to be able to search. What should you do?
   - **A.** Implement the Search charm to support searching the data of your app.
   - **B.** Use a search box to allow for in-app searches.
   - **C.** Add a search button to the app bar that shows a custom menu for searching.
   - **D.** Use the Share charm to send your data to another application such as Excel, in which it can be easily filtered.
3. Your application implements a share target, but it doesn't show in the list of share targets when viewed from other apps. How can you fix it?
   - **A.** Configure the app manifest with a Share Target declaration.
   - **B.** Other apps can't share data to your app; this can be done only to the Mail app.
   - **C.** Add a configuration setting to the Settings charm to enable your app as a share target.
   - **D.** Reinstall the app now that the Share contract is implemented.

Objective 2.3: Implement search

By using the Search charm, you help users navigate your app and find what they want. When implementing search, you combine the built-in features of Windows with your app's specific requirements.

This objective shows you what built-in features Windows offers and how you can use them in your own app.

This objective covers how to:

- Provide search suggestions using the SearchPane and SearchBox control classes
- Search and launch other apps
- Provide and constrain search within an app, including inside and outside of the Search charm
- Provide search result previews
- Implement activation from within search
- Configure search contracts

Providing search suggestions using the SearchPane and SearchBox control classes

Search can be added in two different locations:

- In-app search through the SearchBox control
- In the Search charm with the SearchPane control

The following sections discuss these options, show you how to implement them, and highlight the differences so you can make an informed decision.

In-app search with a search box

To add in-app search to your Windows Store app, start with a SearchBox control found in the WinJS.UI namespace. Adding a SearchBox control is easy; all you need is the following HTML:

```
<div id="searchBoxId"
     data-win-control="WinJS.UI.SearchBox">
</div>
```

Figure 2-12 shows what the SearchBox control looks like on a black background.

FIGURE 2-12 The SearchBox control

A search box can be used when search is an essential part of your app. For example, in the Windows Store, most people will use a search box to access the content the Store offers. Showing the search box directly in your app signals to users where to start when they are new to your app. When space is a problem is in your app, consider showing only the search icon or moving the search box to your app bar.

By default, when the search box doesn't have the focus, the control is dimmed. When the control receives focus, it becomes active and changes color.

SearchBox exposes the querysubmitted event, which is important for responding to the user starting a new search query. You can subscribe to this event with the following code:

```
var elem = document.querySelector("#searchBoxId");
elem.addEventListener("querysubmitted", searchHandler);
```

The querysubmitted event gets an argument of type SearchBoxQuerySubmittedEventArgs. This class contains the following members:

- **KeyModifiers** Gets any modifier keys that are pressed when the user presses Enter to submit a query.
- **Language** Gets the Internet Engineering Task Force (IETF) language tag (BCP 47 standard) that identifies the language currently associated with the user's text input device.
- **LinguisticDetails** Gets information about query text that the user enters through an input method editor (IME).
- **QueryText** Gets the query text of the current search.

The most important property is QueryText, which contains the text that the user searched for. By combining it with the user's language and linguistic details, you can provide customized search results for different kinds of users. Your app can then start searching through its own content, but you can also search external content such as the Internet.

When it comes to search suggestions, SearchBox is quite intelligent. By default, it remembers the previous search actions of a user. You can also add your own code to supply custom search suggestions.

By subscribing to the suggestionsrequested event, you can supply suggestions while the user is typing a query text. You can provide two types of search suggestions:

- Query suggestions
- Result suggestions

MORE INFO **RESULT SUGGESTIONS**

Read the following section, "Providing search result previews," for more information on implementing result suggestions.

Query suggestions can be seen as an autocomplete for the user's search text. When a user selects a query suggestion, the search page is displayed with the selected query text.

Your app has total freedom in retrieving those query suggestions. For example, you can use a static list of data inside your app, get it from a web service, or use one of the standard formats such as OpenSearch or XML search.

The following code shows how to subscribe to the suggestionsrequested event:

```
var elem = document.querySelector("#searchBoxId");
elem.addEventListener("suggestionsrequested", suggestionsRequestedHandler);
```

Now you can handle the event with the following code:

```
function suggestionsRequestedHandler (args) {
    var queryText = args.detail.queryText,
    query = queryText.toLowerCase(),
    suggestionCollection = args.detail.searchSuggestionCollection;
    if (queryText.length > 0) {
        suggestionCollection.appendQuerySuggestion(query + " suggestion");
    }
}
```

The event argument you get has the same query details as the querysubmitted event. It also has the searchSuggestionCollection that you can use to append search suggestions. This example only appends the word "suggestion" to your query text. Of course, you can add multiple suggestions from multiple sources, but this example shows the essential structure.

Issuing an OpenSearch or XML web service call follows the same pattern. Within your suggestionsrequested event handler, you can fire a web service call and parse the result. This data can then be added to the searchSuggestionCollection property.

Implementing search through the Search charm

When working with the Search charm, you use the Windows.ApplicationModel.Search.SearchPane class to get access to the SearchPane for your app and expose events that you can use to subscribe to queries and suggestion requests. The following code shows how to register for query submissions to your app:

```
Windows.ApplicationModel.Search.SearchPane.getForCurrentView().onquerysubmitted =
    function (eventObject) {
    };
```

When you add this code to the global scope of your app, you register for queries that are submitted to your app. To integrate with the Search charm, you also need to edit the app manifest and add the Search declaration. If you forget this, you get an exception when executing the previous code.

You can use SearchPane the same way. In addition to the querysubmitted event, you also have a suggestionsrequested event and a showOnKeyboardInput property, with which you can make sure that the Search charm opens when a user starts typing and you can suggest search results.

You can't use both a search box and the Search contract in one app. You need to decide which type of search you want to implement and then choose one.

Searching and launching other apps

One of the design decisions that changed between Windows 8 and Windows 8.1 has to do with search. Because search is such an important scenario, Windows 8 offers a unique location to start searching that's the same for all apps: the Search charm.

Based on feedback and telemetry, this design wasn't the ideal situation the designers originally imagined. With Windows 8.1, Microsoft improved on the search experience by changing the following characteristics:

- Searching everywhere—files, apps, settings, and the web—is now the default when the Search charm is invoked, which saves the step of switching targets.
- For in-app search, the recommendation is to implement a search control directly on the app canvas. The WinJS.UI.SearchBox control is included in WinJS for this purpose.
- For apps that want to provide a search capability but aren't yet prepared to provide a full in-app experience, you can use a simple button that invokes the Search charm directly and work with the Search contract from there. In this case, the Search charm is scoped to the app by default instead of "everywhere."

A user can change the target of the Search charm to files, settings, or the Internet. The capability to search other apps has been removed as of Windows 8.1.

Understanding these changes in the exam is important. By default, your app should now use a search box for an in-app experience.

Providing and constraining search within an app

As a result of the design changes mentioned in the previous section, you should limit your search results and suggestions to your in-app data. In Windows 8, search would go over multiple apps and even the Internet.

With the newly designed search box, your in-app search should limit its results to in-app results. If it makes sense in the context of your app, you might choose to add web results to your in-app results.

This is true when using the recommended search box, but also when using the Search charm launched from within your app.

Providing search result previews

In the first section of this objective, you learned about adding query suggestions to both SearchBox and SearchPanel. You can also add result suggestions to help users. These suggestions, which are more specific to what the user is looking for, immediately link to a specific search result. Figure 2-13 shows an example of both query and result suggestions in the Windows Store. The result suggestions are shown at the top; after a separator, the query suggestions display.

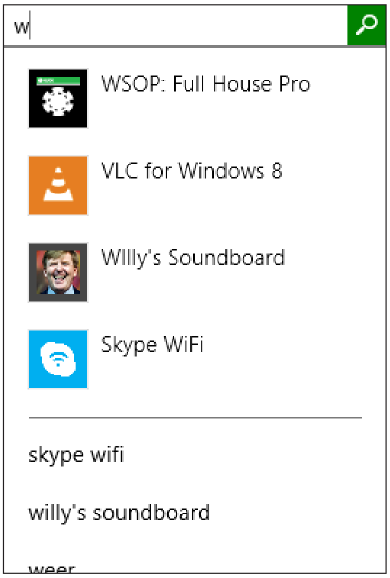


FIGURE 2-13 The query and result suggestions when searching in the Windows Store

When working with the SearchBox control, you can use the suggestionsrequested event to handle query suggestions. The searchSuggestionCollection object that you get past as an event argument can be used to add query suggestions, result suggestions, and separators.

The following code shows an example:

```
function suggestionsRequestedHandler(eventObject) {
    var queryText = eventObject.detail.queryText,
        suggestionCollection = eventObject.detail.searchSuggestionCollection;

    var imageUri = new Windows.Foundation.Uri("ms-appx:///images/image.png");
    var imageSource = Windows.Storage.Streams.RandomAccessStreamReference.
                       createFromUri(imageUri);

    suggestionCollection.appendResultSuggestion("text", "detailText", "tag",
        imageSource, "imageAlternateText");

    suggestionCollection.appendSearchSeparator("");
    suggestionCollection.appendQuerySuggestion(queryText + "suggestion");
}
```

By using the appendResultSuggestion method, you can add a result suggestion for users. Next to some string parameters, this method also expects a RandomAccessStreamReference that points to an image that you want to display next to the result suggestion. This image should be 40 × 40 pixels; otherwise, Windows scales it for you. This image can be created from another image that you have somewhere in your app package by using an URI that points to the location of your image.

After adding a separator, you can then add any query details you have. When a user selects a query result, Windows uses this text and raises the querysubmitted event. But when selecting a result suggestion, the user expects to be taken to the item directly instead of the search results page.

You can handle result selections by subscribing to the resultsuggestionchosen event:

```
searchBox.addEventListener("resultsuggestionchosen", resultSelectedHandler);
```

This event gets an instance of SearchBoxResultSuggestionChosenEventArgs that has a tag property set to the value of tag used when adding the result suggestion.

SearchPane has the same functionality. By subscribing to the onsuggestionsrequested event and onresultsuggestionchosen, you can implement the same functionality as for SearchBox.

EXAM TIP

Make sure that you understand that responding to a result suggestion should show the result page instead of the search page.

Implementing activation from within search

Chapter 1 discussed the life cycle of an app. When an app is launched, you can check the ActivationKind enumeration for the reason why it launched. You can see whether a user launched your app from the Search charm with the following code:

```
if (eventObject.detail.kind ===
    Windows.ApplicationModel.Activation.ActivationKind.search) {
    if (eventObject.detail.queryText === "") {
        // No query. Show the standard landing page
    } else {
        // Show search results
    }
}
```

When your app is activated through search, you first see whether the user entered any query text. If not, the user just wants to open your app. If a query text is available, you can immediately go to the search results page.

EXAM TIP

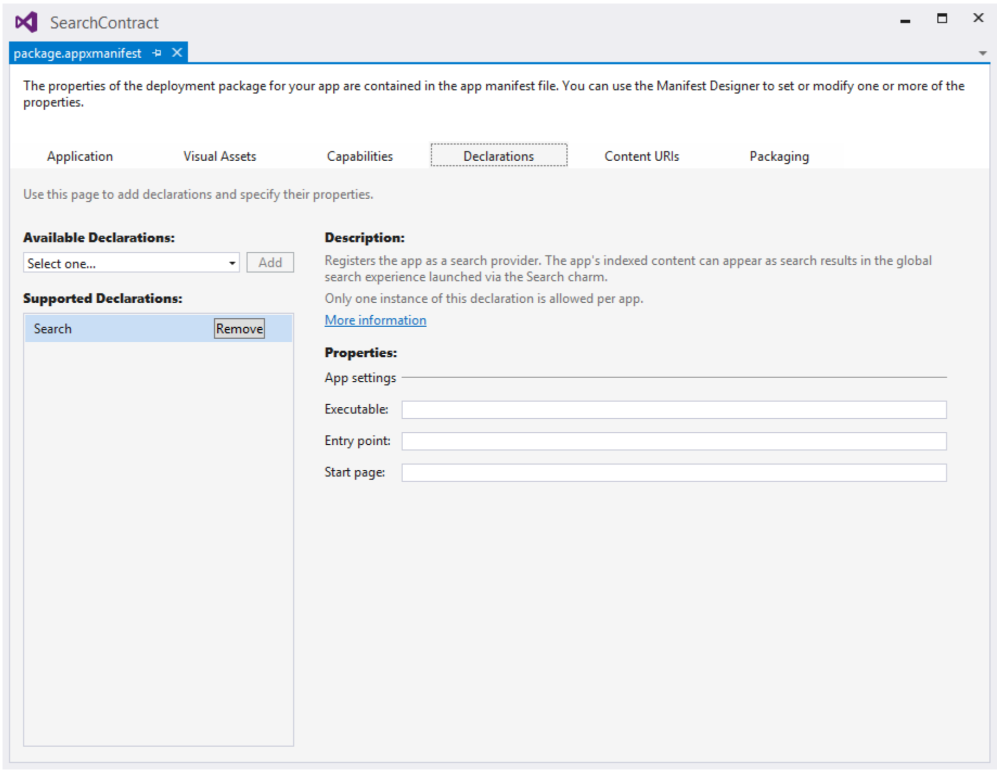
The activated event and the ActivationKind property are very important when it comes to working with all contracts and charms that Windows supports. Become familiar with this event and make sure that you understand the different values of ActivationKind.

Configuring search contracts

Windows uses the app manifest to check the permissions that your app requires. If your app doesn't request a permission for a feature, it can't use that feature.

To use in-app search with SearchBox, you don't need a contract. To integrate with the Search charm, however, you have to create an explicit declaration in your app manifest to request permission.

Figure 2-14 shows the Search declaration in the App Manifest Designer.



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... Add

Supported Declarations: Search Remove

Description:

Registers the app as a search provider. The app's indexed content can appear as search results in the global search experience launched via the Search charm.

Only one instance of this declaration is allowed per app.

More information

Properties:

App settings

Executable:

Entry point:

Start page:

FIGURE 2-14 The App Manifest Designer showing the Search declaration

Executable and Entry Point are specific to C++ apps. In your JavaScript app, you don't need to configure them. The Start Page is specific to JavaScript apps. However, when using it in combination with the Search declaration, it doesn't do anything.

Thought experiment

Choosing between SearchBox and SearchPane

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 designing a recipe app, and you are certain that your app needs search. You have to choose between using SearchBox or using SearchPane. With that in mind, answer the following questions:

1. What are the differences between SearchBox and SearchPane?
2. Describe a situation in which you would use SearchBox and describe a situation in which you would use SearchPane.

Objective summary

- SearchBox can be used for in-app search.
- SearchPane can be used with the Search charm. You can't use both SearchBox and SearchPane in one app. When using SearchPane, you have to add the Search declaration to the app manifest.
- You can provide query and result suggestions to help users.
- Your app can be activated from search.

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 trying to access SearchPane in your app. When you try to add a handler for the querysubmitted event, your app crashes with an Access Is Denied error. What should you do?
   - **A.** Run your app as administrator.
   - **B.** Use SearchBox instead of SearchPane.
   - **C.** Use the onsuggestionsrequested event.
   - **D.** Add a Search declaration to the app manifest.
2. Your app helps users find information about ingredients. A user searches for the text "garl", and you want to provide search suggestions. What should you do? (Choose all that apply.)
   - **A.** Add a separator.
   - **B.** Add a query suggestion for "garl".
   - **C.** Add a query suggestion for "garlic".
   - **D.** Add a result suggestion for "garlic".
3. You have a web service that returns XML suggestions to your app, and you use the SearchBox control. You want to use those suggestions and display them to the user. Which event should you use?
   - **A.** suggestionchosen
   - **B.** resultsuggestionsrequested
   - **C.** querysuggestionsrequested
   - **D.** suggestionsrequested

Objective 2.4: Implement Share in an app

How often do you share data? Maybe you are reading an interesting article and you copy the URL to an email. Or you see a funny picture and you want to share it on Twitter. Or you want to share your new high score with friends.

The possibilities are endless. Microsoft understands that sharing data is an important part of working with apps, and that is why Microsoft designed a complete infrastructure for sharing data. You can help users easily share content they find in your app, and you can design your app so other apps can share data with your app.

It all works together to make sure that apps become more popular and can "win as one." This objective shows you how to share data.

This objective covers how to:

- Use the DataTransferManager class to share data with other apps
- Accept sharing requests by implementing activation from within Share
- Limit the scope of sharing using the DataPackage object
- Implement in-app Share outside of the Share charm
- Use web links and application links

Using the DataTransferManager class to share data with other apps

The most prevalent Share scenario is sharing data from your app to other apps. In such a scenario, your app acts as a Share source (the next section discusses being a share target).

To initiate a Share action, you need a reference to the DataTransferManager class:

```
var dataTransferManager =
    Windows.ApplicationModel.DataTransfer.DataTransferManager.getForCurrentView();
```

You don't have to add a declaration to your app manifest to share data from your app. When you have an instance of the dataTransferManager, you can add an event handler for the datarequested event. This event is fired whenever the user opens the Share charm.

The following code shows a simple example:

```
function registerForShare() {
    var dataTransferManager = Windows.ApplicationModel.DataTransfer.
        DataTransferManager.getForCurrentView();
    dataTransferManager.addEventListener("datarequested", shareTextHandler);
}
```

```
function shareTextHandler(e) {
    var request = e.request;
    request.data.properties.title = "Title for share";
    request.data.properties.description = "Description of share";
    request.data.setText("Hello Share!");
}
```

When you call the registerForShare method from the activated event, your app is set up for Share. Figure 2-15 shows what the Share charm looks like when using this code.

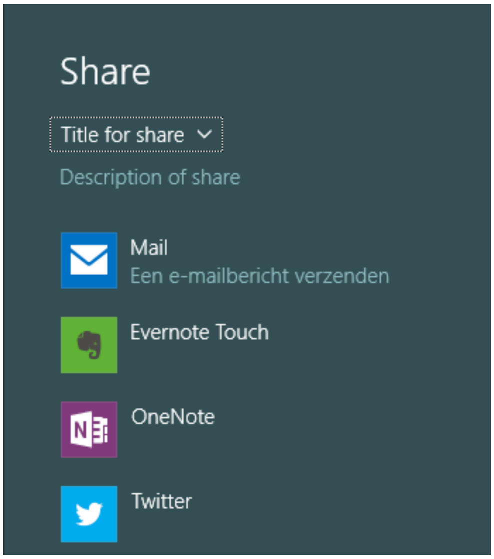


FIGURE 2-15 The Share charm with sample content

Your datarequested handler gets an event object of type DataRequestedEventArgs. The only property this object has is request, which is the object you use to supply the data you want to share.

You can share different data formats. In the previous example, you saw how to share some text. The formats you can share are these:

- Text
- Link
- HTML
- File
- Single
- Multiple files and images

MORE INFO **CUSTOM FORMATS**

You can create custom formats that are based on a published schema or on a schema that you create and publish. For more information, see *http://msdn.microsoft.com/en-us/library/windows/apps/hh750306.aspx*.

To configure what data to share, use the request.data property. This object has the following methods that you can use to set data:

- **SetApplicationLink** Sets the application link that a DataPackage object contains
- **SetBitmap** Sets the bitmap image contained in the DataPackage object
- **SetData** Sets the data contained in the DataPackage object in a RandomAccessStream format
- **SetDataProvider** Sets a delegate to handle requests from the target app
- **SetHtmlFormat** Adds HTML content to the DataPackage object
- **SetRtf** Sets the Rich Text Format (RTF) content contained in a DataPackage object
- **SetStorageItems(IIterable(IStorageItem))** Sets the files and folders contained in a DataPackage object
- **SetStorageItems(IIterable(IStorageItem), Boolean)** Adds files and folders to a DataPackage object
- **SetText** Sets the text contained in a DataPackage object
- **SetUri** Sets the URI contained in a DataPackage object
- **SetWebLink** Sets the web link contained in a DataPackage object

By using these methods, you can share all kinds of data with other apps.

Your app might have data to share that takes some time to process. If you put all the logic to create those items in your datarequested handler, activating the Share charm might take much too long.

You can use deferring to avoid this problem. Instead of creating the data in your datarequested handler, specify a method that the target app can request whenever the data is needed by using the setDataProvider method on the DataPackage object. This method expects the format that you want to share and a method that can be called whenever the target needs the data:

```
request.data.setDataProvider(Windows.ApplicationModel.DataTransfer.StandardDataFormats.
    bitmap, onDeferredImageRequested);
```

In this case, you create a deferral for an image request, which gives your method the time to process the image and send it to the share target when required.

Remember that this is different from executing a method asynchronously. To process an asynchronous method in your datarequested handler, use the request.getDeferral method.

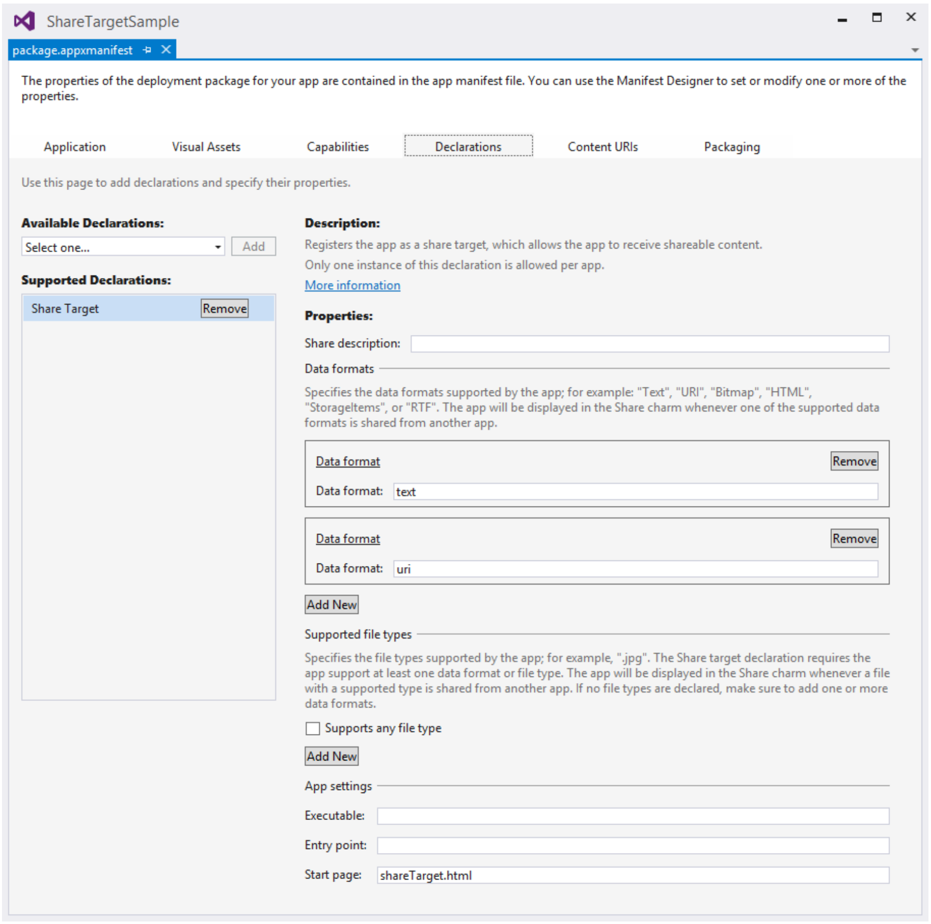
After completing your asynchronous operation, call deferral.complete to signal to Windows that your Share data is ready.

Accepting sharing requests by implementing activation from within Share

In addition to sharing content from your app with other apps, you can configure your app to be a share target. Other apps can then share content with your app, and your app suddenly shows up when users are working with other apps.

To become a share target, you have to configure the app manifest. When your app is installed, Windows then registers it as a share target, and your app shows up in the Share charm.

Figure 2-16 shows the declaration for the Share contract.



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: Share Target

Description:
Registers the app as a share target, which allows the app to receive shareable content.
Only one instance of this declaration is allowed per app.

Properties:
Share description:

Data formats

Specifies the data formats supported by the app; for example: "Text", "URI", "Bitmap", "HTML", "StorageItems", or "RTF". The app will be displayed in the Share charm whenever one of the supported data formats is shared from another app.

Data format: text

Data format: uri

Supported file types

Specifies the file types supported by the app; for example, ".jpg". The Share target declaration requires the app support at least one data format or file type. The app will be displayed in the Share charm whenever a file with a supported type is shared from another app. If no file types are declared, make sure to add one or more data formats.

☐ Supports any file type

App settings

Executable:
Entry point:
Start page: shareTarget.html

FIGURE 2-16 The Share declaration in the App Manifest Designer

To configure your app as a share target, define the data formats that your app supports. Figure 2-16 shows a format of text and URI. Whenever an app wants to share these types of data, your app will show up in the share targets list.

You also see a value for the Start page, which is the HTML page that Windows shows when your app is selected as a share target.

As you saw in previous sections, Windows supports different ways of activating your app. One way is as a share target. Whenever your app is registered as a share target, you should check the ActivationKind like this:

```
app.onactivated = function (args) {
    if (args.detail.kind === Windows.ApplicationModel.Activation.ActivationKind.
shareTarget) {
        // initialize view for share
    }
};
```

When your app is activated through Share, you get a shareOperation object passed as an argument to your activated method: args.detail.shareOperation. This object contains the data package that an app wants to share with you. It also lets you give progress reports to users when the share operation is taking too much time.

For example, when an app wants to share some text with your app, you can use the following code:

```
var shareOperation = eventObject.detail.shareOperation;
if (shareOperation.data.contains(
    Windows.ApplicationModel.DataTransfer.StandardDataFormats.text)) {
        shareOperation.data.getTextAsync().done(function (text) {
        // use text
        }, function (e) {
        // handle error
        }
    });
}
```

In your Share UI, you need a button to process your share logic. That code is completely custom for your application.

The easiest way to get started implementing the share target is to open the dialog box for adding a new item to your application and choosing Share Target Contract. It adds HTML, JavaScript, and CSS files to your app that implement a basic UI for sharing. It also adds a Share Target declaration to your manifest with a data format of text and URI.

MORE INFO **SHARING CONTENT TARGET APP SAMPLE**

The sharing content target app sample provided by Microsoft shows how your app can receive content from another app. You can find the sample at *http://code.msdn.microsoft.com/windowsapps/Sharing-Content-Target-App-e2689782*.

EXAM TIP

Make sure you understand the difference between being a share source and a share target. Know when to use each one and how to implement both scenarios.

Limiting the scope of sharing using the DataPackage object

Data can be shared in multiple different data formats. Windows offers the following formats that you can use:

- Bitmap is used for sharing images.
- HTML is used for sharing HTML content.
- Rich Text Format (RTF) is used for sharing RTF content.
- Storage Items is used for sharing files.
- Text is used for sharing plain text.
- URI is used for sharing URIs.

When looking at this list and thinking about an app you are planning to develop, you probably see some potential candidates that you can use in your app. However, not all apps find all data formats useful. Sharing HTML content from your music player? Or sharing files from a recipe app? Maybe those scenarios are useful for your app, but you should think carefully about the items you want to share.

You share data from your app by using the DataPackage class, which has several methods that you can use to initialize your DataPackage:

- **SetApplicationLink** Sets the application link
- **SetBitmap** Sets the bitmap image contained in the DataPackage object
- **SetData** Sets the data contained in the DataPackage object in a RandomAccessStream format
- **SetDataProvider** Sets a delegate to handle requests from the target app
- **SetHtmlFormat** Adds HTML content to the DataPackage object
- **SetRtf** Sets the RTF content contained in a DataPackage object
- **SetStorageItems** Sets the files and folders contained in a DataPackage object
- **SetText** Sets the text contained in a DataPackage object
- **SetUri** Sets the URI contained in a DataPackage object
- **SetWebLink** Sets the web link contained in a DataPackage object

By choosing the correct method, you can specify the type of data you want to share, which determines which apps show up in the Share charm.

Implementing in-app Share outside of the Share charm

The Share charm, which is always available for users to activate, is useful when they decide to share some data that your app has to offer.

In certain scenarios you might want to stimulate users to share something. Maybe they have just achieved a new high score in your game, and you want them to share this with their friends (hoping to attract new users to your app!).

To help them, you can activate the Share charm directly from your app with the following line of code:

```
Windows.ApplicationModel.DataTransfer.DataTransferManager.showShareUI();
```

Figure 2-17 shows a very simple UI. When you click the Share text, the Share charm opens, and you see the title and description that you configure in your app. The user can then select the app that should act as a share target.

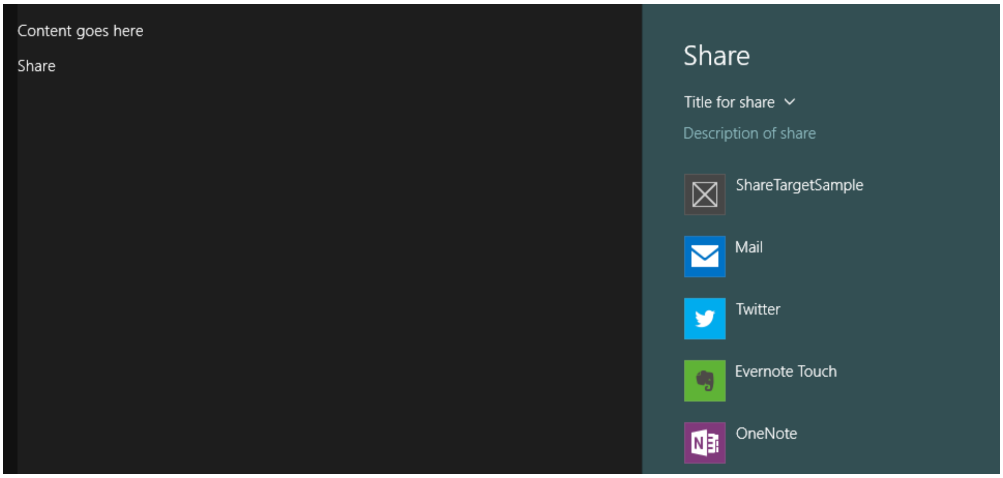
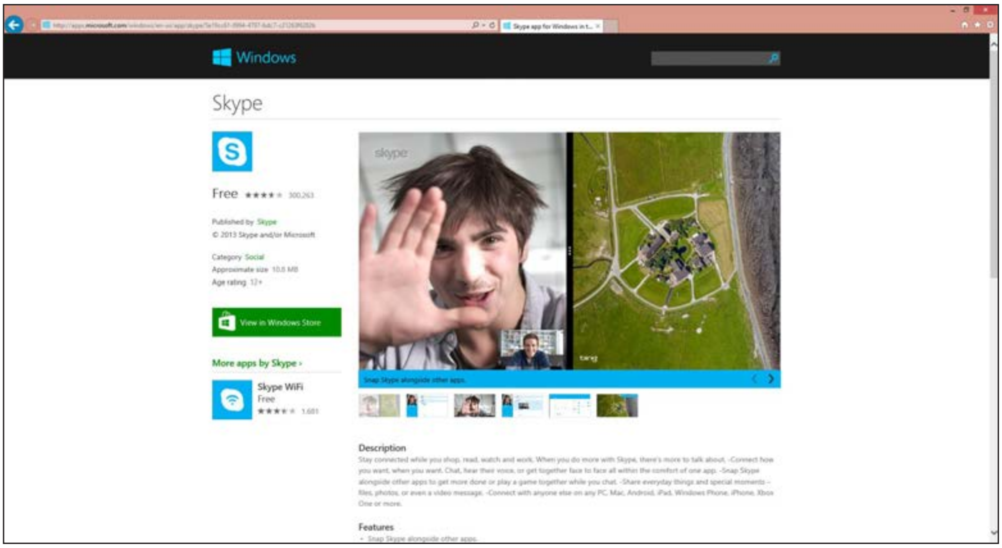


FIGURE 2-17 Opening the Share charm directly from within an app

Using web links and application links

To increase the visibility of your app, Microsoft not only adds your app to the Windows Store but also generates a webpage that describes your app. Figure 2-18 shows an example: the webpage for the Skype app.



Stay connected while you shop, read, watch and work. When you do more with Skype, there's more to talk about. -Connect how you want, when you want. Chat, hear their voice, or get together face to face all within the comfort of one app. -Snap Skype alongside other apps to get more done or play a game together while you chat. -Share everyday things and special moments – files, photos, or even a video message. -Connect with anyone else on any PC, Mac, Android, iPad, Windows Phone, iPhone, Xbox One or more.

- Snap Skype alongside other apps.

FIGURE 2-18 Internet Explorer shows the webpage for the Windows Store Skype app

The good thing about this webpage is that whenever a user opens the page on a Windows 8 device, the Windows Store is automatically launched and shows the app page directly in the store. These web links allow you to share a normal URL with users to showcase your app. When users with Windows 8 click your link, they are taken to the store. Users can just view the webpage and share the URL with others.

Application links are a little different because they expect the user to be on a Windows 8 device and directly link to the store. Application links can be useful when you link to another app from your own app. Because users are already running your app, you know that they have a Windows 8 device, and you can skip the part where you first launch their browser with a web link and then take them to the Windows Store.

An application link that shows an app listing page has the following format:

```
ms-windows-store:PDP?PFN=
```

The Package Family Name (PFN) is unique for your app. You can find it in Visual Studio or in the app listing page such as the one shown in Figure 2-18. When you open the HTML source of the Skype app listing page, you find the following line:

```
var packageFamilyName = 'Microsoft.SkypeApp_kzf8qxf38zg5c';
```

It is the PFN that you can use to link directly to Skype. Of course, you can use the same technique to link to whatever package is in the Windows Store.

In addition to displaying an app listing page, you can process the following with the Windows Store:

- **Publisher** Opens the page displaying all apps from a publisher
- **Updates** Opens the Windows Store updates page
- **Search** Runs a search query and displays the results
- **PDP** Opens an app's listing page
- **Review** Opens the "Write a review" page of an app's listing

Thought experiment

Sharing 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 building a recipe app, and you wonder whether share is something your app should support.

1. **Should you make the app a share source?**
2. **Should you make the app a share target?**

Objective summary

- The DataTransferManager class is crucial for a share source or share target.
- By using a DataPackage object, you can specify the data that you want to share with other apps.
- By adding a Share declaration, you can respond to activation through share and become a share target.
- Think about the data formats you want to support in your app and make sure that they make sense to the user.
- You can directly activate share from within your app by calling showShareUI on the DataTransferManager.
- Through web and application links, you can share URIs with users so they can find your app.

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. Your app should act like a share source. You are sharing some text data with other apps. What should you do? (Choose all that apply.)
   - **A.** Add a Share declaration to your manifest.
   - **B.** Listen for the datarequested event.
   - **C.** Initialize a data package with the data you want to share.
   - **D.** Add a deferral to load the data for your share whenever required.
2. You want to add a button to your UI that shows the Share charm whenever a user clicks it. What do you do?
   - **A.** You can't add a button for share in your UI. Users should use the Search charm.
   - **B.** You can create a DataPackage object in the onclick of the button and configure it with the data you want to share.
   - **C.** You can call showShareUI on the DataTransferManager to show the Share charm.
   - **D.** You can use an application link in your app that links to the Share charm.
3. Your app acts as a share target. What should you do?
   - **A.** Listen to the sharetargetedactivated event.
   - **B.** Listen to the activated event and check the ActivationKind.
   - **C.** Listen to the datarequested event.
   - **D.** Listen to the deferallsharerequest event.

Objective 2.5: Manage application settings and preferences

The Settings charm offers a central location in which users can configure your app. Windows adds some default content to the Settings charm such as a rate and review option. Your app can add other options to the Settings charm and save the changes so they are synced across user devices.

This objective shows you how to use the Settings charm in your apps.

This objective covers how to:

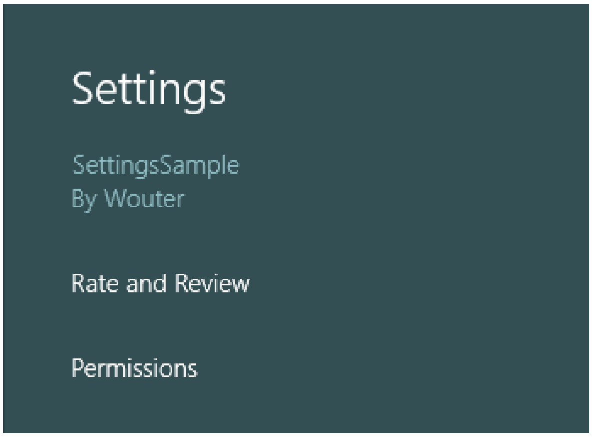
- Choose which application features are accessed in AppSettings
- Add entry points for AppSettings in the Settings window
- Create settings flyouts using the SettingsFlyout control
- Add settings options to the SettingsFlyout control
- Store and retrieve settings from the roaming app data store

Choosing which application features are accessed in AppSettings

When you install your app through the Windows Store, Windows automatically adds two sections to your Settings charm:

- Rate And Review
- Permissions

You can see an example of those two sections in Figure 2-19. When your app is not installed through the Windows Store, there is no Rate And Review option in the Settings charm.


Settings

SettingsSample
By Wouter

Rate and Review

Permissions

FIGURE 2-19 The Settings charm shows the Rate And Review and Permissions settings

The additional settings are entirely up to you. You should generally think about settings as those that change frequently and those that don't.

For example, users of a paint app frequently change the size and types of brushes, but they probably configure the metric system only once. If a setting is part of the normal workflow, it should be in the app bar. If it is a setting that changes infrequently, you should put it in the Settings charm.

Items that definitely belong in the Settings charm are informational items such as the app's privacy policy, help, version, and copyright information.

Adding entry points for AppSettings in the Settings window

The Settings charm doesn't contain settings; instead, it shows the groups of settings in your app and allows you to navigate to them. These settings groups are called *entry points*.

You can add entry points with the WinJS.Application.onsettings event, which is raised whenever the user launches the Settings charm. In your event handler, you can add additional application commands and use them to populate the Settings charm:

```
app.onsettings = function (e) {
    e.detail.applicationcommands = {
        "defaultsDiv": { href: "html/DefaultSettings.html", title: "Defaults" },
        "helpDiv": { href: "html/HelpUI.html", title: "Help" }
    };
    WinJS.UI.SettingsFlyout.populateSettings(e);
}
```

This code adds two entry points to the Settings charm: Defaults and Help. Each points to a different HTML page that contains the actual settings.

It is a best practice to add no more than four different sections to a Settings charm. You should group settings as much as possible and give them a descriptive, one-word label. If you don't have a name for your group, name it Default.

From within your application, you can link directly to specific sections in the Settings charm. Suppose that you want to add a Help button to your app bar that should show the Help section in the Settings charm when clicked. You can do it with the following code:

```
document.getElementById("showHelp").addEventListener('click', function () {
    WinJS.UI.SettingsFlyout.showSettings("helpDiv", "html/HelpUI.html");
});
```

Make sure that the path and ID match that of the application command in the onsettings event. To make your code more maintainable, move those IDs and paths to separate variables and use them when referring to sections in the Settings charm.

If a specific section group is not useful in a scenario, don't hide the label; instead, disable it so the user can't select it.

EXAM TIP

Never add a setting directly to the main settings page. Always use entry points that open SettingsFlyout controls with the actual settings.

Creating settings flyouts using the SettingsFlyout control

The setting flyouts that you add as HTML pages should follow a few strict rules that describe their appearance and behavior:

- Always launch a settings flyout from entry points in the Settings pane.
- Use a light-dismiss surface that appears on top of the main app content and disappears when the user clicks outside the flyout or resizes the app. Closing the flyout automatically lets people change a setting quickly and get back to the app.
- Make sure that the settings flyout appears on the same side of the screen as the charms and Settings pane. Use the SettingsEdgeLocation property to determine which side of the screen the Settings charm appears on.
- Slide the flyout in from the same side of the screen as your Settings pane instead of from the top or bottom of the screen.
- A flyout must be full screen height, regardless of orientation, and should be narrow (346 pixels) or wide (646 pixels). Choose the width that's appropriate for the content; don't create custom sizes.
- The flyout header should include a back button, the name of the entry point that opened the flyout, and the app's icon.
- The header background color should be the same as the background color of your tile.

- The border color should be the same color as the header, but 20 percent darker.
- Display settings content on a white background.

Fortunately, Microsoft added a WinJS.UI.SettingsFlyout control to help implement all these rules. A basic template for your settings flyout can look like this:

```
<div id="defaultsDiv" data-win-control="WinJS.UI.SettingsFlyout"
    aria-label="Defaults Settings flyout"
    data-win-options="{settingsCommandId:'default',width:'narrow'}">
    <div class="win-header" style="background-color:#464646">
        <button type="button" onclick="WinJS.UI.SettingsFlyout.show()"
                class="win-backbutton"></button>
        <div class="win-label">Defaults</div>
        <img src="ms-appx:///images/smalllogo.png"
             style="position: absolute; right: 40px;" />
    </div>
    <div class="win-content">
        {App defaults content goes here}
    </div>
</div>
```

The div defines that it is a control of type WinJS.UI.SettingsFlyout. It also adds a header with a back button, which lets the user return to the Settings charm overview.

You can put this template inside a HTML page that defines the HTML, head, and body tags. You can also link to CSS and JavaScript files that are specific to your settings flyout.

MSDN helps you by defining some guidelines that are important when working with settings. You don’t have to memorize them all, but make sure that you understand the general rules and know how to apply them for your exam.

Some general principles include these:

- Create entry points for all app settings in the Settings pane.
- Keep your settings simple. Define smart defaults and keep the number of settings to a minimum.
- If necessary, link from elements of your app’s UI to the Settings pane or deep-link to a specific settings flyout. For example, you can link to your help settings flyout from a Help button in the bottom app bar and from a Help entry point in the Settings pane.
- When a user changes a setting, the app should reflect the change immediately. Apply settings changes instantly or as soon as a user is done interacting with the flyout.
- Use the WinJS.UI.SettingsFlyout control. This control implements the UI design guidelines by default.
- Don’t include commands that are part of common app workflow in app settings, such as changing the brush color in an art app. These commands belong on an app bar or on the canvas.

- Don’t use an entry point in the Settings pane to perform an action directly. Entry points should open settings flyouts.
- Don’t use the Settings pane for navigation. When it closes, users should be in the same place where they were when they clicked the Settings charm. The top app bar is a more appropriate place for navigation.
- Don’t use the SettingsFlyout classes as all-purpose controls. They are intended only for settings flyouts launched from entry points in the Settings pane.

When creating your entry points, keep the following in mind:

- Group similar or related options together under one entry point. Avoid adding more than four entry points to your Settings pane.
- Display the same entry points regardless of the app context. If some settings are not relevant in a certain context, disable them in the settings flyout.
- Use descriptive, one-word labels for your entry points whenever possible. For example, for account-related settings, name the entry “Accounts” rather than “Account settings.” If you want only one entry point for your settings, and the settings don’t lend themselves to a descriptive label, use “Options” or “Defaults.”
- If an entry point is linking directly to the web instead of a flyout, let the user know with a visual clue—for example, “Help (online)” or “Web forums” styled as a hyperlink. Consider grouping multiple links to the web into a flyout with a single entry point. For example, an “About” entry point could open a flyout with links to your terms of use, privacy statement, and app support.
- Combine less-used settings into a single entry point so that more common settings each has its own entry point. Put content or links that only contain information in an “About” entry point.
- Don’t duplicate the functionality in the “Permissions” pane. Windows provides this pane by default, and you can’t modify it.

Adding settings options to the SettingsFlyout control

Now that you have added options to the Settings charm and created settings flyouts that are shown whenever a user selects a label, you need to add some options to the flyouts to help users configure your app.

When creating your settings flyout, think about good design. You shouldn’t create a settings flyout that overwhelms users with many options and makes it hard for them to find the particular option that they are looking for.

Instead, make sure that your settings flyout isn’t extremely long. Limit the scrolling to a maximum of twice the screen height. Group the settings, add descriptive names, and don’t create nested hierarchies of settings.

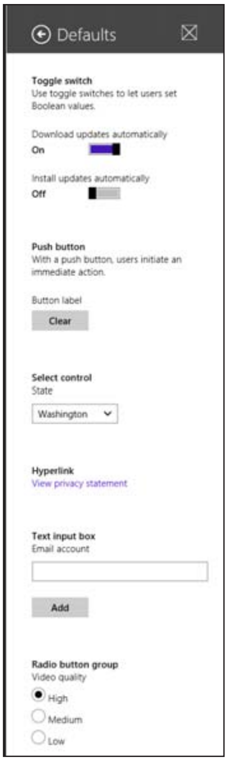
Microsoft also created controls that you can use in your settings flyout:

- **Toggle Switch** Users can set values on or off.
- **Radio Button** Users can choose one item from a set of up to five mutually exclusive, related options.
- **Select Control** Users can choose one item from a set of six or more text-only items.
- **Text Input Box** Users can enter text. Use the type of text input box that corresponds to the type of text you get from the user, such as an email or password.
- **Hyperlink** Users are taken to another page within the app or to an external website. When a user clicks a hyperlink, the settings flyout is dismissed.
- **Button** Users can initiate an immediate action without dismissing the current settings flyout.

The following code shows an example of using these controls:

```
<div class="win-content">
    <div class="win-settings-section">
        <h3>Toggle switch</h3>
        <p>Use toggle switches to let users set Boolean values.</p>
        <div id="Toggle1" data-win-control="WinJS.UI.ToggleSwitch"
             data-win-options="{title:'Download updates automatically',checked:true}">
        </div>
        <div id="Toggle2" data-win-control="WinJS.UI.ToggleSwitch"
             data-win-options="{title:'Install updates automatically'}">
        </div>
    </div>
    <div class="win-settings-section">
        <h3>Push button</h3>
        <p>With a push button, users initiate an immediate action.</p>
        <label>Button label</label>
        <button type="button" onclick="WinJS.log &&
                WinJS.log('Clear history button pressed', 'samples', 'status')">
            Clear
        </button>
    </div>
    <div class="win-settings-section">
        <h3>Select control</h3>
        <p>Use the select control to allow users to select
           one item from a set of text-only items.</p>
        <label>State</label>
        <select aria-label="State select control">
            <option value="AK">Alaska</option>
            <option value="CA">California</option>
            <option value="CO">Colorado</option>
            <option value="HI">Hawaii</option>
            <option value="ID">Idaho</option>
            <option value="KS">Kansas</option>
            <option value="MT">Montana</option>
            <option value="NE">Nebraska</option>
            <option value="NV">Nevada</option>
            <option value="NM">New Mexico</option>
            <option value="ND">North Dakota</option>
            <option value="OR">Oregon</option>
            <option value="SD">South Dakota</option>
            <option value="TX">Texas</option>
            <option value="UT">Utah</option>
            <option value="WA" selected>Washington</option>
            <option value="WY">Wyoming</option>
        </select>
    </div>
    <div class="win-settings-section">
        <h3>Hyperlink</h3>
        <p>Use a hyperlink when the associated action will
           take the user out of this flyout.</p>
        <a href=http://go.microsoft.com/fwlink/?LinkID=190175
           target="fix_link_too">View privacy statement</a>
    </div>
    <div class="win-settings-section">
        <h3>Text input box</h3>
        <p>Use a text input box to allow users to enter text.
          Set the type of the text input box according to the
          type of text you're capturing from the user (e.g. email or password).</p>
        <label>Email account</label>
        <input type="text" aria-label="Enter email account" />
        <button type="button" onclick="WinJS.log &&
                                        WinJS.log('Add email account button pressed',
                                        'samples', 'status')">Add</button>
    </div>
    <div class="win-settings-section">
        <h3>Radio button group</h3>
        <p>Lets users choose one item from a small set of mutually exclusive,
           related options.</p>
        <label>Video quality</label>
        <label><input type="radio" name="video" value="High" checked />High</label>
        <label><input type="radio" name="video" value="Medium" />Medium</label>
        <label><input type="radio" name="video" value="Low" />Low</label>
    </div>
</div>
```

Figure 2-20 shows what those controls look like when used in a settings flyout.


Defaults

Toggle switch
Use toggle switches to let users set Boolean values.

Download updates automatically
On

Install updates automatically
Off

Push button
With a push button, users initiate an immediate action.

Button label
Clear

Select control
State
Washington

Hyperlink
View privacy statement

Text input box
Email account

Add

Radio button group
Video quality
● High
○ Medium
○ Low

FIGURE 2-20 The settings flyout shows basic controls

Your app should immediately respond to changes in the settings; a user shouldn't have to push a button to apply the changes.

You can use the JavaScript file of your flyout to attach events to all your setting controls. By responding to those events, you can then call into your app and apply the new settings. A simple example consists of a toggle that you attach to:

```
(function () {
    "use strict";
    var page = WinJS.UI.Pages.define("/html/DefaultSettings.html", {

        ready: function (element, options) {
            document.getElementById('Toggle1').
                            addEventListener('change', toggleChanged);
        },
    });
    function toggleChanged(arg) {
        var toggleControl = arg.target.winControl;
        WinJS.Application.updateSettings(toggleControl.checked);
    }
})();
```

This code defines the JavaScript that runs for DefaultSettings.html. It attaches a handler to the change event of a toggle. When this event is raised, it calls a custom defined method on the application:

```
app.updateSettings = function (arg) {
    WinJS.log && WinJS.log(arg);
}
```

This method outputs only the new settings value to the log, but you see how to easily write code to update the UI.

Storing and retrieving settings from the roaming app data store

Of course, you have to save the changes users make to app settings somewhere. In a Windows Store app, you have three locations in which you can save data: local, roaming, and temporary.

Temporary data can be deleted by the operating system at any time. Local data is persistent between updates and stays on the same device.

When you save settings, consider the roaming app data store. Roaming data is automatically synced between users' devices. So if you store your app settings in the roaming app data store, and a user installs your app on another device, the user's favorite settings are automatically loaded. Syncing settings between devices creates the uniform experience that Windows Store apps should offer.

Not all data is suitable to be saved in a roaming app data store. Data that is specific to the device, such as local paths, shouldn't be saved to a roaming location.

You can access different data stores through the Windows.Storage.ApplicationData property. You can use the RoamingSettings property to store and retrieve settings for your app.

You can easily change the toggle control of the previous example to use RoamingSettings:

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

    var roamingSettings = Windows.Storage.ApplicationData.current.roamingSettings;
    var page = WinJS.UI.Pages.define("/html/DefaultSettings.html", {
        ready: function (element, options) {
            document.getElementById('Toggle1')
                    .addEventListener('change', toggleChanged);
            var value = roamingSettings.values["toggleState"];
            document.getElementById("Toggle1").winControl.checked = value;
        }
    });

    function toggleChanged(arg) {
        var toggleControl = arg.target.winControl;
        roamingSettings.values["toggleState"] = toggleControl.checked;
    }
})();
```

By using values[propertyname], you can read and write properties to and from the roaming settings. You can also remove a value from your roaming settings by using roamingSettings.values.remove(settingName).

Roaming settings sync across devices, so settings sometimes change because they are updated on other devices. Windows.Store.ApplicationData exposes a datachanged event that you can attach to listen to changes in your roaming settings:

```
Windows.Storage.ApplicationData.current.addEventListener("datachanged",
roamingDataChangedHandler);
```

Inside this handler, you can read the new values that were written to the roaming settings object and then update your app accordingly.

Remember that there is a maximum amount of data you can store in the roaming settings (You can check it by using the ApplicationData.RoamingStorageQuota property.) If you add more data than is allowed, synchronization across devices is paused until you have removed data.

Other ideas to keep in mind when working with roaming settings include these:

- Assuming that there is network connectivity, an app's roaming state is roamed within 30 minutes on an active machine. It is also roamed immediately when the user logs on or locks the machine.
- Locking the machine is always the best way to force a sync to the cloud. Note that if the cloud service is aware of only a single device for a user (that is, for any given a Microsoft account), synchronization with the cloud service happens only once per day. When the service is aware that the user has multiple machines, it begins synchronizing within the 30-minute period.
- If the app is uninstalled on all machines except one, synchronization reverts to the longer period.
- When saving roaming state, you can write values whenever you like, such as when those settings change. Don't worry about grouping your changes; Windows has a built-in debounce period. This means that if subsequent updates occur over a small period of time, Windows will combine those changes into one update of the roaming state to reduce overall network traffic.
- If you have a group of settings that must be roamed together, manage it as a composite setting in your roamingSettings container.
- Files you create within the roamingFolder container are not roamed as long as you have the file open for writing (that is, as long as you have an open stream). For this reason, it's a good idea to make sure that all streams are closed when the app is suspended.
- Windows allows each app to have a "high priority" setting that is roamed within one minute, thereby enabling apps on multiple devices to stay much more closely in sync. This one setting (which can be a composite setting) must exist in the root of your roamingSettings container with the name HighPriority: roamingSettings.

values["HighPriority"]. The setting must also be 8 KB or smaller to maintain the priority. If you exceed 8 KB, the setting roams with normal priority. (Note that the setting must be a single or composite setting; a settings container with the same name roams with normal priority.)

- On a trusted PC, general user settings such as the Start page configuration are automatically roamed independently of the apps. They include encrypted credentials saved by apps in the Credential Locker (if enabled in PC Settings). Apps should never attempt to roam passwords themselves; they should use the Credential Locker for this
- Apps that create secondary tiles can indicate whether such tiles should be copied to a new device when the app is installed.
- When multiple state versions are in use by different versions of an app, Windows manages each version of the state separately; newer state versions aren't roamed to devices with apps that use older state versions. So it is a good idea to be less aggressive in versioning your state because it breaks the roaming connection between apps.
- The cloud service retains multiple versions of roaming state as long as multiple versions are in use by the same Microsoft account. Older versions of the roaming state are eligible for deletion only when all instances of the app have been updated or uninstalled.
- When an updated app encounters an older version of roaming state, it should load it according to the old version but then call setVersionAsync to migrate to the new version.
- Avoid using secondary versioning schemes within roaming state that introduce structural differences without changing the state version through setVersionAsync. Because the cloud service manages the roaming state by this version number, and because the last writer always wins, a version of an app that expects to see some extra bit of data (and saved it there) might find that it has been removed because a slightly older version of the app didn't write it.
- Even if all apps are uninstalled from a user's devices, the cloud service retains roaming state for "a reasonable time" (maybe 30 days). So if users reinstall the app within that time period, their settings are still intact. Use the clearAsync method to avoid retention and explicitly clear roaming state from the cloud.
- To debug roaming state, check out the Roaming Monitor Tool available in the Visual Studio Gallery. It provides status information on the current sync state, a Sync Now button to help with testing, a browser for roaming state, and a file editor. (At the time of this writing, this tool is available only for Visual Studio 2012 for Windows 8 and has not been updated for Windows 8.1; it might appear directly in Visual Studio and not as an extension.)

EXAM TIP

Make sure that you understand how roaming settings work and in which scenarios you would use them.

Thought experiment

Configuring 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. The restaurant's owners want users to be able to place orders through the app, and the app should keep track of user preferences and make special offers based on their purchase history.

1. **Users require an account to log on to the app. How should they manage their account?**
2. **Should you store the user preferences in the roaming settings?**

Objective summary

- The Settings charm should be used for all settings that are not part of the user's workflow.
- Add entry points for all groups of settings in your app. You do so by subscribing to the onsettings event and populating the application commands with the commands you want to add to the Settings pane.
- For each group of settings, use a SettingsFlyout control that contains the actual controls that configure your settings.
- You can use roaming settings to automatically synchronize settings across user devices.

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 want to add an option to configure the color theme of your app. What should you do?
   - **A.** Add a button to the app bar that switches color themes. Immediately apply the change.
   - **B.** Add a switch button directly to the Settings charm, which frees a user from having to open a flyout when you have only a couple of options.
   - **C.** Add a settings flyout in which a user can select a color theme from a list. Add an apply button so the user can save changes.
   - **D.** Add a settings flyout in which a user can select a color theme from a list. Immediately apply the change.

2. Your app integrates with the file picker, and you want to save the most-used paths for the user to offer a quick way to get there. Where can you save this data?
   A. RoamingFolder
   B. RoamingSettings
   C. LocalSettings
   D. LocalFolder
3. Your settings flyout is 646 pixels wide and is too small for the content you want to show. What should you do?
   A. Use CSS to set the width of the flyout to auto size.
   B. Set the flyout options to a width of wide.
   C. Use CSS to set the width of the flyout to 1000px.
   D. Rearrange your layout to make the content fit the 646 pixels.

Objective 2.6: Integrate media features

Now that apps are more mature, you see them moving from relatively simple data processing apps to fully featured media apps. Microsoft recognizes this maturity and has added extra support for all types of media, from images to video and audio.

This section shows you how to use different media features to give your app an extra spike. You also learn about text-to-speech (TTS), which makes your app more accessible to all kinds of users.

This objective covers how to:

- Support DDS images
- Implement video playback
- Implement XVP and DXVA
- Implement TTS
- Implement audio and video playback using HTML5 DRM

Supporting DDS images

DirectDraw Surface (DDS) is a file format created by Microsoft for storing image data so that it can be easily rendered by a graphics processing unit (GPU).

Using DDS files improves the performance of an image-heavy app because it doesn't involve the CPU. DDS files can store different types of data; support layering; and support mipmaps (precalculated, optimized collections of images that accompany a main texture, intended to increase rendering speed and reduce aliasing artifacts), cube maps (using the six faces of a cube as a map shape), volume maps (used for two-dimensional [2-D] projections of three-dimensional [3-D] data sets), and texture arrays. Most of these features are used in games that rely on DirectX for rendering.

With the release of Windows 8.1, Windows Store apps use DDS files. With WebGL (a JavaScript application programming interface [API] that renders 3-D and 2-D graphics within a browser), you can use the DDS file format for large textures with good performance.

The Windows Store samples contain a sample called a block compressed images sample, which consists of a C++ project and a JavaScript Windows Store app.

Why C++? It is used for converting regular JPEG files to DDS files by adding the ImageContentTask build customization to the project. When you start working with DDS files for your own projects, you can copy the C++ project and use it to generate DDS files that automatically follow the Windows Store instructions (correct block compression format and alpha settings).

After you have generated DDS files, using them in your Windows Store apps is easy. By using an HTML5 canvas object, you can first load the images and then draw them on the canvas.

The following code is taken from the block compressed images sample:

```
var guitarPath = "BlockCompressedAssets\\guitar-transparent.dds";
var packageLocation = Windows.ApplicationModel.Package.current.installedLocation;
var guitar;

packageLocation.getFileAsync(guitarPath).then(function (file) {

    guitar = new Image();
    guitar.src = window.URL.createObjectURL(file, { oneTimeOnly: true });
    return packageLocation.getFileAsync(guitarPath);

})).then(function (file) {
    var context = document.getElementById("canvas").getContext("2d");
    context.drawImage(guitar, 0, 0);

});
```

This code first loads the image asynchronously and then gets a reference to the HTML canvas and draws the image on it. Because you are drawing on a canvas, you can apply all kinds of animations and other logic required for your app.

MORE INFO **BLOCK COMPRESSED IMAGES SAMPLE**

You can find the complete example at *http://code.msdn.microsoft.com/windowsapps/BlockCompressedImages-19830098/*. You have to download the C++, JavaScript version of the sample.

Implementing video playback

Playing video in your Windows Store app is easy. Because Windows Store apps with HTML, JavaScript, and CSS run in Internet Explorer 11, you can use the advanced techniques that HTML5 offers you.

The HTML5 tag you want to use is the <video> tag:

```
<video id="myVideo" src="http://www.mydomain.com/myclip.mp4" controls/>
```

After specifying the source of your video and the controls attribute, you see the display shown in Figure 2-21.

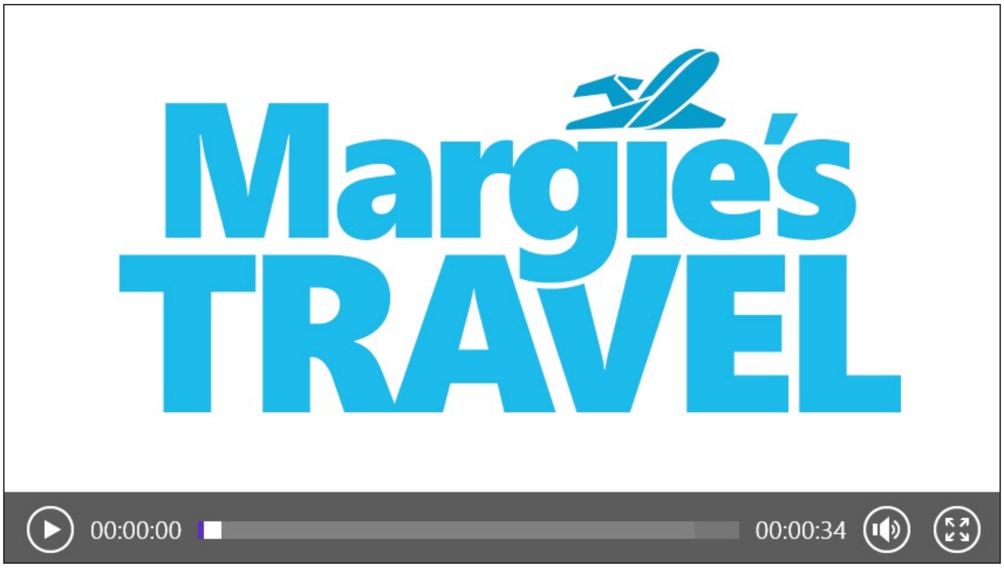


FIGURE 2-21 The video tag shows a video of Margie's Travel with controls

You should include the width and height of your video as attributes on your video tag so that Internet Explorer knows the size of your video before it is loaded and can correctly calculate the layout of other items.

Your video object has methods, such as play and pause, for basic control of video. But you can also use more advanced features such as adding subtitles, using multiple audio tracks (in different languages, for example), and scaling video to the user's screen.

MORE INFO **HTML MEDIA PLAYBACK SAMPLE**

A sample that shows the different things you can do with video in your Windows Store app can be found at *http://code.msdn.microsoft.com/windowsapps/Media-Playback-Sample-3bdbe7c9*.

Implementing XVP and DXVA

Transcoding video means that you convert one format to another format. The Media Foundation Transcode Video Processor (XVP) defines standards that you can use to transcode your videos.

Some of these formats, such as MP3 and MP4, are probably familiar to you. Transcoding elements can be supported by DirectX Video Acceleration (DXVA), which means that your GPU is used to help transcode your video.

Using the GPU makes your performance dependent on the device's hardware. The XVP added the MrfCrf444 mode, which always runs in software and does not use DXVA hardware acceleration. What does this mean? The transcoding of your video to MrfCrf444 does not depend on hardware acceleration—you get the same results across platforms, independent of the underlying hardware.

MrfCrf444 runs entirely in software, so transcoding might take longer and use more power. The benefit of using MrfCrf444 is that you don't need any special hardware because the converting is entirely done in software.

You can transcode videos by using the Windows.Media.Transcoding namespace. The following code transforms a video from Mp4 to the new MrfCrf444 format:

```
// sourceFile and destFile are IStorageFile objects defined elsewhere.
var profile = Windows.Media.MediaProperties.MediaEncodingProfile.createMp4
    (Windows.Media.MediaProperties.VideoEncodingQuality.hd720p);
var transcoder = new Windows.Media.Transcoding.MediaTranscoder();
transcoder.videoProcessingAlgorithm =
    Windows.Media.Transcoding.MediaVideoProcessingAlgorithm.mrfCrf444;
transcoder.prepareFileTranscodeAsync(sourceFile, destFile, profile);
```

This code starts by creating a new profile specifying that you are dealing with an Mp4 file. It then creates a new instance of MediaTranscoder and configures it to use the MrfCrf444 profile. The code then starts the transcoding, passing it an existing source file, the name and location of the destination file you want to create, and the profile you want to use.

MORE INFO **TRANSCODING SAMPLE**

A sample that shows how to implement transcoding between all types of formats can be found at *http://code.msdn.microsoft.com/windowsapps/media-transcode-sample-f7ba5269*.

Implementing TTS

Text To Speech (TTS) was possible before the release of Windows 8.1 only by using an external Internet service such as Text-To-Speech with Microsoft Translator Service.

Microsoft added the Windows.Media.SpeechSynthesis namespace in Windows 8.1. The classes in this namespace help you take a string of text and convert it to an audio stream, so the computer can then read any on-screen text to the user.

TTS can read plain text or a special XML format called Speech Synthesis Markup Language (SSML). SSML uses special attributes to give instructions to the speech synthesizer, such as how to read dates or numbers, where to add emphasis, and where to pause.

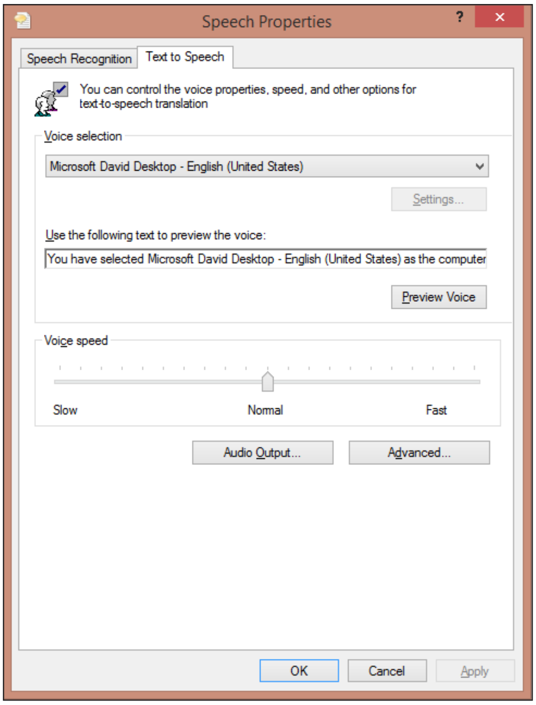
The following code shows an example of how to read some plain text to the user:

```
function readText() {
    var audio = new Audio();
    var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();

    synth.synthesizeTextToStreamAsync("Hello Exam Ref").then(function (markersStream) {
        var blob = MSApp.createBlobFromRandomAccessStream(
                            markersStream.ContentType,
                            markersStream);
        audio.src = URL.createObjectURL(blob, { oneTimeOnly: true });
        audio.play();
    });
}
```

This example creates a new SpeechSynthesizer class and then creates a temporary object that contains the audio stream and plays it to the user.

Depending on your language settings, you have a couple of different voices installed (you can view the voices on your device by searching for **text to speech**). The Text To Speech dialog box shown in Figure 2-22 shows the installed languages and the capability to preview them.



You can control the voice properties, speed, and other options for text-to-speech translation

Voice selection

Microsoft David Desktop - English (United States)

Use the following text to preview the voice:

You have selected Microsoft David Desktop - English (United States) as the computer

Voice speed

FIGURE 2-22 The Speech Properties dialog box shows TTS options

You can choose different voices for your application by using the Windows.Media.SpeechSynthesis.SpeechSynthesizer.allVoices property, which returns an array of voices available on your device. By setting the SpeechSynthesizer.voice property, you can configure which voice to use for TTS.

Reading SSML follows the same pattern. Suppose that you have the following piece of SSML (taken from the Windows Store sample on TTS):

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
       http://www.w3.org/TR/speech-synthesis/synthesis.xsd" xml:lang="en-US">
        This sentence has marks <mark name="mark1"/>Here and another mark here<mark
          name="mark2"/>
        This is an example of how to speak the word <phoneme alphabet="x-microsoft-ups"
           ph="S1 W AA T . CH AX . M AX . S2 K AA L . IH T">whatchamacallit</phoneme>.
        This is an example of how to use the say-as tag to say a date
           <say-as interpret-as="date:mdy">04/30/2013</say-as>
        This <say-as interpret-as="ordinal"> 4 </say-as> example is how to use the
          ordinal data type
</speak>
```

Instead of using synthesizeTextToStreamAsync, you use synthesizeSsmlToStreamAsync.

Implementing audio and video playback using HTML5 DRM

Digital Rights Management (DRM) is important when it comes to intellectually protected content. To use DRM-protected content in your app, use the MediaProtectionManager object, which is responsible for handling access to protected media content. You can attach it to a HTML video or audio tag or by using its API.

Start by creating a new MediaProtectionManager object:

```
var mediaProtectionManager = new Windows.Media.Protection.MediaProtectionManager();
```

Now you have to configure the media protection system so it can access the protected content. You can configure it by using a special certificate or GUID that represents the protection system ID. You can also pass extra data to the protection system by using the properties property:

```
mediaProtectionManager.properties["Windows.Media.Protection.MediaProtectionSystemId"] =
        '{F4637010-03C3-42CD-B932-B48ADF3A6A54}';
```

Attach the MediaProtectionManager instance to your video or audio element like this:

```
video.msSetMediaProtectionManager(mediaProtectionManager);
```

The MediaProtectionManager object exposes three events that you can handle when something goes wrong:

- **ComponentLoadFailed** Fired when the load of binary data fails
- **RebootNeeded** Fired when a reboot is needed after the component is renewed
- **ServiceRequested** Fired when the content protection system encounters an issue and needs the app's help

You should subscribe to these events to handle errors in accessing the DRM content. After taking these steps, you can now point your video or audio element to the source of a DRM-protected file. MediaProtectionManager gives you access to the content, and the file can be played on the user's device.

Thought experiment

Designing an app that uses video and audio

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 use video and audio, and you are considering all the possibilities Windows Store apps offer you.

3. **Is it difficult to implement audio and video in a Windows Store app? How would you start implementing it?**
4. **What are the advantages of DRM-protected content?**

Objective summary

- DDS images can be used in Windows 8.1 apps.
- Video and audio playback is easy to implement by using video and audio HTML5 tags.
- Transcoding media can be done by using the MediaTranscoder class, which supports different formats that can benefit from hardware acceleration when transcoding.
- TTS can be implemented in Windows Store apps by using the SpeechSynthesizer class, which synthesizes plain text and SSML.
- DRM is important when working with video and audio content. You can access DRM-protected content by using MediaProtectionManager.

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 have lots of JPEG files that you want to use in a game you are developing. You want to use those files as DDS files. What should you do? (Choose all that apply.)
   - **A.** Use C++ to draw the DDS images to an HTML5 canvas.
   - **B.** Convert the JPEG files to DDS files by using a JavaScript project.
   - **C.** Use JavaScript to draw the DDS images to an HTML5 canvas.
   - **D.** Convert the JPEG files to DDS files by using a C++ project.

2. You have added a HTML5 video tag to your app. Your app now shows the video, but doesn't show play, pause, or other buttons. What should you do?
   - **A.** Add the buttons you want to your UI and use JavaScript to wire the buttons to actions on the video element.
   - **B.** Showing buttons is not possible. The video element is completely self-contained and can't be manipulated to avoid piracy issues.
   - **C.** Add the buttons attribute to the video tag.
   - **D.** Add the controls attribute to the video tag.
3. You have some specialized XML that contains hints to the speech synthesizer on how to read. Which method should you use to read this XML?
   - **A.** synthesizeTextToStreamAsync
   - **B.** synthesizeTextToStream
   - **C.** synthesizeSsmlToStreamAsync
   - **D.** synthesizeSsmlToStream

Answers

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

Objective 2.1: Thought experiment

1. Some examples include a meeting app in which you can invite people, a task planner app in which you can collaborate on tasks, and a game in which you want to invite people to play.
2. Although a less-common scenario, you shouldn't dismiss it from the start. If your app has contact with some external data source, it can be used to expose contacts. Maybe you are building an intranet app to list all colleagues or you are connecting to some social network that you can use to show people to the user.

Objective 2.1: Review

1. **Correct answer:** C
   - **A. Incorrect:** Your app doesn't have to be running. Windows launches your app whenever the user selects it as a contact picker.
   - **B. Incorrect:** You have to subscribe to the activated event, but your app is launched by Windows.
   - **C. Correct:** To show up as a possible source of contacts, you have to declare this in your manifest. Windows then registers your app upon installation.
   - **D. Incorrect:** You launch the contact picker only when your app needs contacts and you want them to be selected from within another app.
2. **Correct answer:** B
   - **A. Incorrect:** This method is used when your app acts as a contact picker source and you want to add a selected contact to the basket that gets returned to the app that is looking for contacts.
   - **B. Correct:** This shows the contact picker allowing the user to select a single contact.
   - **C. Incorrect:** This would allow the user to select multiple contacts.
   - **D. Incorrect:** The pickContactAsync method is correct. The async postfix shows that this method returns a promise and processes asynchronously.

3. **Correct answer:** C
    - A. **Incorrect:** This would filter the contacts after the user has left the contact picker. You want to show only those contacts that have this field already filled in.
    - B. **Incorrect:** This would filter the contacts after the user has left the contact picker. You want to show only those contacts that have this field already filled in.
    - C. **Correct:** By using a selection mode of field, you can specify the required field. In this case, it is the email field.
    - D. **Incorrect:** A selection mode of contact shows all contacts. It doesn't filter on specific fields.

Objective 2.2: Thought experiment

1. By using contracts and charms, apps can integrate without knowing explicit details about each other. This integration helps to fulfill the Microsoft design principle of "win as one."
2. Your app gains visibility when a user is using another app and sees that it integrates with your app. It also makes using your app easier because users can work with it in a familiar way.
3. A desktop application such as Microsoft Outlook could integrate with the contact picker to select people to whom to send an email or invite for a meeting. Other apps could benefit from the Share contract. Easily sharing documents, email, and images is more natural with charm integration.

Objective 2.2: Review

1. **Correct answer:** C
    - A. **Incorrect:** If sharing is not an essential part of the user's workflow, you should not add a share button to the UI.
    - B. **Incorrect:** The Devices charm is for sending data to another device such as a printer or TV; it is not for sending email.
    - C. **Correct:** The Share contract allows you to share data with any other app that can receive your type of data (such as text).
    - D. **Incorrect:** The Share contract does not allow printing; the Devices contract enables printing.

2. **Correct answer:** B
    A. **Incorrect:** The search box is the recommended way to implement search in Windows 8.1.
    B. **Correct:** The SearchBox control allows for a consistent in-app search implementation.
    C. **Incorrect:** You should never create your own custom interface for search. Instead use a SearchBox control
    D. **Incorrect:** If searching is required for your app, you shouldn't handle it outside of your app.
3. **Correct answer:** A
    A. **Correct:** A manifest declaration is required for your app to show up as a share target.
    B. **Incorrect:** All apps that implement the Share contract can act as a share target. The Mail app is just one of the apps that implement the Share contract.
    C. **Incorrect:** This is not required. When you configure the manifest correctly, your app is registered as a share target on installation.
    D. **Incorrect:** When the manifest is not configured, reinstalling the app doesn't make a difference.

Objective 2.3: Thought experiment

1. SearchBox is for in-app search. SearchPane is used in combination with the Search charm.
2. Since the release of Windows 8.1, SearchBox is the preferred method.

Objective 2.3: Review

1. **Correct answer:** D
    A. **Incorrect:** Apps run under the current user account; they can't be run with extra privileges as desktop applications can.
    B. **Incorrect:** It is true that SearchBox doesn't require any extra configuration, but that would require a redesign of the app. Changing the configuration is much easier.
    C. **Incorrect:** This event is required to respond to search queries. It doesn't fix the access denied error, however, which occurs because of a wrongly configured manifest.
    D. **Correct:** This fixes the access denied error.

2. **Correct answers:** A, C, D
   - **A.** **Correct:** The separator should be added between the query and result suggestions.
   - **B.** **Incorrect:** Because the user has already typed “garl,” it is not useful to add a query suggestion for this.
   - **C.** **Correct:** The user can select this option to process a query for the word “garlic.”
   - **D.** **Correct:** This is a highly probable search result that the user can select to go directly to the garlic page.
3. **Correct answer:** D
   - **A.** **Incorrect:** Suggestionchosen does not exist. The resultsuggestionchosen event exists, but is used whenever the user selects a result suggestion from the list.
   - **B.** **Incorrect:** The resultsuggestionsrequested event does not exist. Instead, you should use the suggestionsrequested event to add both query and result suggestions.
   - **C.** **Incorrect:** The querysuggestionsrequested event does not exist. Instead, you should use the suggestionsrequested event to add both query and result suggestions.
   - **D.** **Correct:** This event can be used to parse the XML and add both query and result suggestions.

Objective 2.4: Thought experiment

1. Definitely. Whenever users see a recipe they like, they often want to share it with others. Implementing Share helps the user quickly send an email with the recipe information (or post it on Twitter or Facebook).
2. This might be useful. Imagine that users see a recipe they like on the Internet. By sharing the webpage to your app, they can quickly copy the recipe to your app—maybe as plain HTML, or you can implement some kind of parsing.

Objective 2.4: Review

1. **Correct answers:** A, B, C
   - **A.** **Correct:** The Share declaration is required to be able to participate in the Share contract.
   - **B.** **Correct:** The datarequested event is raised whenever Windows asks your app for data to share.
   - **C.** **Correct:** A DataPackage object contains the data that your app wants to share with other apps.
   - **D.** **Incorrect:** Because you are sharing only text data, you can create the DataPackage object directly in the datarequested event.

2. **Correct answer:** C
   - **A. Incorrect:** Adding a button to show the Share UI from your app is allowed.
   - **B. Incorrect:** The DataPackage object needs to be created in the datarequested event.
   - **C. Correct:** Calling showShareUI shows the Share UI.
   - **D. Incorrect:** Applications links are not used for opening charms.
3. **Correct answer:** B
   - **A. Incorrect:** This event does not exist. Instead, you should listen to the Activated event and check ActivationKind.
   - **B. Correct:** The Activated event is raised with an ActivationKind of ShareTarget.
   - **C. Incorrect:** The datarequested event is fired whenever the user wants your app to act as a share source.
   - **D. Incorrect:** Deferrals are used whenever your app needs to prepare lots of data as a share source and you want to postpone it to the last moment.

Objective 2.5: Thought experiment

1. Account management is perfectly suited for the Settings charm. Users often log on only once and then want the app to remember their account. If they want to sign out or change account settings, they can open the Settings charm.
2. It depends on the amount of data. The roaming settings have a maximum size that can prevent synchronizing across devices. Because your app already has a back end, you could probably store the user preferences in the back end.
3. Data such as the last page that a user viewed or other app-specific data can be stored in the roaming settings.

Objective 2.5: Review

1. **Correct answer:** D
   - **A. Incorrect:** Selecting a color theme is not done on the app bar. Instead it should be nicely hidden in the Settings charm, where users can probably configure it only once when they launch your app.
   - **B. Incorrect:** The Settings charm can contain entry points only to flyouts that contain settings. It can't contain settings directly.
   - **C. Incorrect:** You should not use an apply button. Changes in settings should be reflected in the app immediately.
   - **D. Correct:** The recommended way to implement these scenarios is with a flyout in the Settings charm that immediately applies the user's changes.

2. **Correct answer:** C
   - **A. Incorrect:** RoamingFolder should be used for files, not for settings. Local settings shouldn't be stored in a roaming app data store.
   - **B. Incorrect:** Local settings shouldn't be stored in a roaming app data store. The path is specific to the current device and might not work on other devices.
   - **C. Correct:** LocalSettings can be used to store the most-used paths.
   - **D. Incorrect:** LocalFolder is there for storing files, not for storing settings.
3. **Correct answer:** D
   - **A. Incorrect:** The Settings charm has a strict design rule that it can be only 346 or 646 pixels. Auto size is definitely not allowed.
   - **B. Incorrect:** The width is already 646 pixels.
   - **C. Incorrect:** A size of 1000 pixels is not allowed. The Settings charm can be only 646 pixels maximum.
   - **D. Correct:** Because the maximum allowed width is 646 pixels, you should arrange your layout and make sure it fits.

Objective 2.6: Thought experiment

1. No. Because Windows Store apps can use the HTML5 elements such as video and audio tags, you can easily start using them in your app.
2. DRM-protected content ensures that you can make video and audio available over the Internet and that only authorized users can access it, thus protecting your content from piracy.

Objective 2.6: Review

1. **Correct answers:** C, D
   - **A. Incorrect:** Drawing to the HTML5 canvas should be done from JavaScript.
   - **B. Incorrect:** Converting images to DDS can be done by a C++ project with the ImageContentTask build customization.
   - **C. Correct:** After the images have been converted, you can draw them on a canvas with JavaScript.
   - **D. Correct:** Converting images to DDS can be done by a C++ project with the ImageContentTask build customization.

2. **Correct answer:** D
   - **A. Incorrect:** You don't have to create these buttons manually. If you add the controls attribute to your video tag, Internet Explorer adds those controls.
   - **B. Incorrect:** DRM can be used to protect content and still allows you to control the video playback at the client.
   - **C. Incorrect:** This attribute does not exist.
   - **D. Correct:** The controls attribute adds all video playback controls to the video element.
3. **Correct answer:** C
   - **A. Incorrect:** The specialized XML is called SSML. The synthesizeTextToStreamAsync method is used to synthesize text, not SSML.
   - **B. Incorrect:** The synthesizeTextToStream method does not exist. Only the async variant exists, but it does not synthesize SSML.
   - **C. Correct:** This method synthesizes SSML asynchronously.
   - **D. Incorrect:** There is only an async variant of this method.

CHAPTER 3

Create the user interface

A huge part of the development of your app is the UI. Making sure that your app is pixel perfect and a real work of craftsmanship is not an easy undertaking. Fortunately, Microsoft helps you build a great UI by not only giving guidance about what makes a great design but also offering you controls that already follow those guidelines.

This chapter focuses on the UI features that Windows Store apps can use. You learn about controls for in-app and for the app bar. You also learn how to create apps that run well, both full screen and in a snap mode, in which your app shares screen space with other apps.

This chapter focuses topics that comprise 20 percent of your exam. Make sure that you get to know all the different controls and styling options that Windows Store apps can use. If you experiment with the samples and try out the controls on your own, you will be fully prepared for the exam.

Objectives in this chapter:

- Objective 3.1: Implement WinJS controls
- Objective 3.2: Implement HTML layout controls
- Objective 3.3: Create layout-aware apps to handle windowing modes
- Objective 3.4: Design and implement the app bar
- Objective 3.5: Apply CSS styling

Objective 3.1: Implement WinJS controls

When building your apps in JavaScript, CSS, and HTML, you build them on top of Windows Runtime (WinRT), which lets you interact with the operating system and can be accessed from JavaScript, C++, and C# apps.

Besides WinRT, the Windows Library for JavaScript (WinJS) offers you additional features that are specific to JavaScript Windows Store apps. This objective describes the controls that WinJS offers you. At the end of this objective, you will know the possibilities, be able to use the controls, and make informed choices about when to use what control.

This objective covers how to:

- Use a FlipView control
- Use a flyout
- Use a Grid layout and a List layout
- Use a menu object
- Use a WebView control
- Use an item container
- Use the Repeater control

Using a FlipView control

FlipView is a control that shows a series of items that a user can easily flip through. These items can be any type of data, including images, static JavaScript Object Notation (JSON) data, or data that you load from a web service. You can use the FlipView control if you have items for users to flip through.

Adding a FlipView control to your app is easy. First, make sure that you load the ui.js file from WinJS:

```
<script src="//Microsoft.WinJS.2.0/js/ui.js"></script>
```

This link is included in all templates by default. The ui.js file contains the JavaScript required for all WinJS controls. Because that file is plain JavaScript, you can look at the definitions of all controls to see exactly how they work.

You can define a FlipView control in your HTML with the following markup:

```
<div id="basicFlipView"
    data-win-control="WinJS.UI.FlipView">
</div>
```

A data attribute is ignored by your browser, so you can use the data attribute to add metadata to HTML elements that you can then use from JavaScript. The same is true for the data-win-control attribute. This attribute specifies that you want to add a WinJS control on top of the div that you defined. A call to WinJS.UI.processAll goes through your HTML and creates controls for you. This method call is added to the activated event handler of all Visual Studio project templates.

If you don't add any Cascading Style Sheets (CSS) to style your FlipView control, it spans the whole width of your app by default. Of course, a FlipView control without any data is not that interesting. The easiest way to add some data is to create static JSON test data to load into your FlipView control.

As you read in Chapter 1, "Design Windows Store apps," separation of concerns (SoC) is an important part of building your app. Making sure that you have distinct objects, each having its own responsibility, is a good way to create a maintainable app.

SoC is also required when getting the data for your app. It is a good practice to create a separate JavaScript file that contains your data access logic. In this example, the app uses only some static data, but that data is still defined in a separate file:

```
(function () {
    "use strict";
    var dataArray = [
    { type: "item", title: "Item1"},
    { type: "item", title: "Item2"},
    { type: "item", title: "Item3"},
    { type: "item", title: "Item4"},
    { type: "item", title: "Item5"}
    ];

    var dataList = new WinJS.Binding.List(dataArray);

    var publicMembers =
      {
          itemList: dataList
      };
    WinJS.Namespace.define("Data", publicMembers);

})();
```

This code defines a new namespace called Data and exposes a WinJS.Binding.List that points to some static JSON data. A List object contains items that can be accessed by index or by a specific string key. You can search, sort, and filter the data and then bind the result to your UI.

The List object exposes a dataSource property that can be used by the FlipView control to bind to the data:

```
<div id="basicFlipView"
     data-win-control="WinJS.UI.FlipView"
     data-win-options="{itemDataSource : Data.itemList.dataSource}">
</div>
```

By using the data-win-options attribute, you specify additional options to configure your WinJS control. In this case, you bind the itemDataSource property of your FlipView control to the dataSource property of the binding list.

Now when you run your app, you see the screen shown in Figure 3-1.

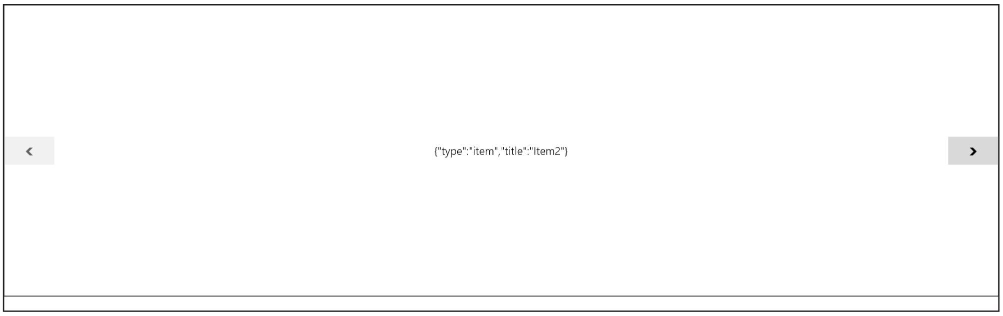

{"type":"item","title":"Item2"}

FIGURE 3-1 The FlipView control shows the second item of the sample data with navigation buttons

The navigation buttons show up only when the user touches or moves the mouse over the FlipView control. What's more interesting is the content in the middle: a string representation of the JSON for the current item. Of course, that isn't what you want, but the FlipView control does not know how to render your data.

To help the FlipView control, you can create an item template, which is a piece of markup that is used by WinJS to render the individual items in the FlipView control. Inside the item template you can use HTML and style it with CSS. The HTML elements can bind to the properties that you have on each data item.

A very simple item template for the sample data that you just saw can be added to the HTML page like this:

```
<div id="ItemTemplate" data-win-control="WinJS.Binding.Template">
    <div>
        <h2 data-win-bind="innerText: title"></h2>
    </div>
</div>
```

Again, the data-win-control attribute is used to signal to WinJS that this is not an ordinary div. Your template must have a single root element, which is a div in this case. An h2 element inside the div is defined that uses the data-win-bind attribute to bind its content to the title property on your data item.

After creating the item template, you can link it to your FlipView control:

```
<div id="basicFlipView"
     data-win-control="WinJS.UI.FlipView"
     data-win-options="{ itemDataSource : Data.itemList.dataSource,
                         itemTemplate : ItemTemplate }">
</div>
```

Now the FlipView control looks like Figure 3-2. The JSON string has disappeared; instead, the title of the item is rendered as an h2 element.

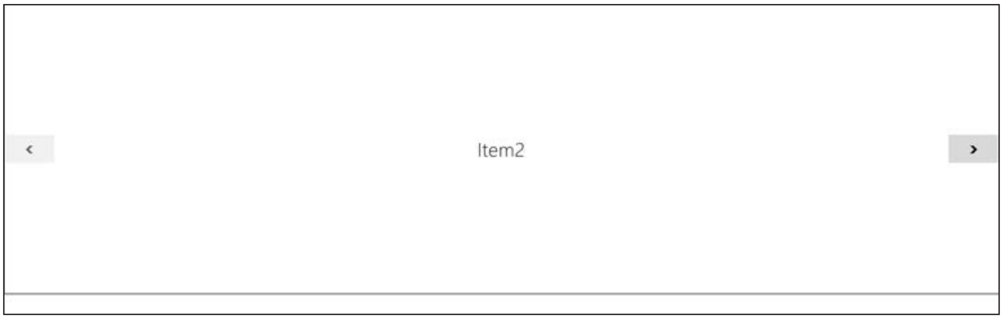


FIGURE 3-2 The FlipView control shows the second item of the sample data with navigation buttons

One simple change you can make with FlipView is to change the orientation:

```
<div id="basicFlipView"
     data-win-control="WinJS.UI.FlipView"
     data-win-options="{ itemDataSource : Data.itemList.dataSource,
                         itemTemplate : ItemTemplate,
                         orientation: 'vertical' }">
</div>
```

After the orientation: 'vertical' option is added, the FlipView control doesn't flip from left to right; it flips from top to bottom.

The FlipView control raises events whenever a user navigates through it:

- onpagecomplete is raised when the FlipView control flips to a page and its renderer function completes.
- onpageselected is raised when the FlipView control flips to a page.
- onpagevisibilitychanged occurs when an item becomes invisible or visible.

By subscribing to those events, you can create content outside of the FlipView control that keeps in sync with your FlipView control, such as a set of radio buttons that shows you which page is selected.

Although the itemTemplate property allows you to render your items with a nice HTML layout, you can't create interactive items. If you want to create them, don't use an item template; use a templating function, which is a method that is called for each item in the FlipView control. It allows you to return completely custom HTML that the FlipView control will render.

You can set a templating function by using the itemTemplate property of a FlipView control:

```
var flipView = document.getElementById("basicFlipView").winControl;
flipView.itemTemplate = mytemplate;
```

A template function takes an itemPromise object. This promise represents the item that will be loaded and displayed in your FlipView control. When the data is loaded synchronously, this promise is already finished; with asynchronous data, the promise might not be done yet.

Your template function has to return one of the following two options:

- A Document Object Model (DOM) element.
- An object containing two properties, element and renderComplete, which is a promise that finishes when the item is completely rendered. WinJS then updates the element property with the final DOM element.

The following code replaces the item template that you saw in the previous example with a templating function:

```
function mytemplate(itemPromise) {
    return itemPromise.then(function (currentItem) {
        var element = document.createElement("h2");
        element.innerText = currentItem.data.title;
        return element;
    });
}
```

This code ensures that the itemPromise finishes and then returns an h2 element with the title of the element. Because you have total control of your items, you can add additional classes and other attributes to identify the items. By binding to the click event of your FlipView control, you can then use these classes to determine which item was clicked. Although it is not a simple scenario to implement, it works well.

MORE INFO **FLIPVIEW SAMPLE**

Microsoft published a sample that demonstrates all capabilities of the HTML FlipView control at *http://code.msdn.microsoft.com/windowsapps/FlipView-control-sample-18e434b4*.

Using a flyout

Another control that WinJS offers is the flyout, which is a lightweight pop-up that you use to show a temporary UI to the user. The flyout can be a menu, a confirmation box, or just a pop-up with more information about an item.

A flyout should be activated by the user. A user can close the flyout by clicking or tapping outside of it, or by pressing the Escape button.

An example of a flyout is found on the Start screen. When you click the user tile, you see a flyout (see Figure 3-3).

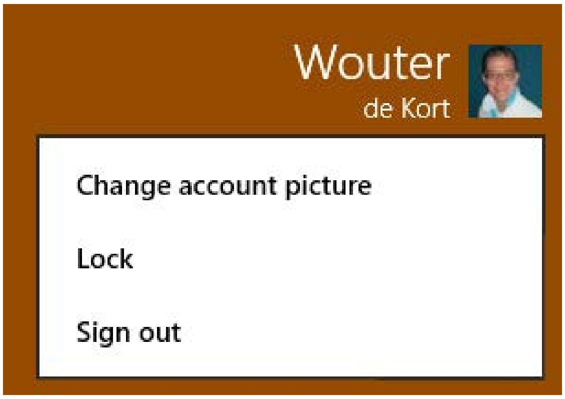


FIGURE 3-3 The user tile on the Start screen shows a flyout

A flyout has three parts:

- **Title** The title is shown at the top of the flyout. Use a title only when needed.
- **Main content** You can show only information to the user, collect input, or let the user change settings. Try to include as few controls as possible.
- **Controls** Controls for submitting changes should not be used. A button should be used only when the user starts an action (such as logon or save).

A flyout is always anchored to an existing control, so you need at least two controls to create one. If you create a new blank app, you can add the following markup for a button and a flyout:

```
<button class="action" id="showFlyoutButton">Show flyout</button>
<div id="confirmFlyout" data-win-control="WinJS.UI.Flyout"
     aria-label="{Confirm action flyout}">
    <div>An action will happen.</div>
    <button id="confirmButton">Do it!</button>
</div>
```

In JavaScript, you have to show the flyout when showFlyoutButton is clicked. You also want to dismiss the flyout when confirmButton is clicked:

```
document.getElementById("showFlyoutButton")
    .addEventListener("click", showFlyout, false);
document.getElementById("confirmButton")
    .addEventListener("click", confirmOrder, false);

function showFlyout(flyout, anchor, placement) {
    var button = document.getElementById('confirmButton');
    var flyout = document.getElementById('confirmFlyout');

    flyout.winControl.show(button, "bottom");
}
function confirmOrder() {
    document.getElementById("confirmFlyout").winControl.hide();
}
```

Dismissing the flyout when a user clicks it or taps outside of it is handled automatically. You have to think about the alignment of your flyout and the anchor control. For a user, it should be immediately clear to which control the flyout belongs.

You can style your flyout to match your app's design. The default styling is done by the light or dark theme that your app uses. In addition to those styles, you can change the CSS styles shown in Table 3-1.

TABLE 3-1 CSS styles for the Flyout control

Property	Example
Font-family	font-family:'Segoe UI';
Font-size	font-size:9pt;
Color	color:rgb(0, 0, 0);
Background-color	background-color:rgb(255, 255, 255);
Border	border:2px solid rgb(128, 128, 128);
Max width	max-width:400px;

You can use a flyout in all parts of your app. For example, you can use a flyout to attach it to a button on your app bar but also to a piece of text to show some extra information or settings.

Using a Grid layout and a List layout

Data can come from anywhere: static JSON, the file system, or a web service. But you also need some way to display this data. One of the controls that you can use in a Windows Store app is ListView control, which is similar in some ways to FlipView. It uses a binding source and templates to render its items.

The way those items are rendered is controlled by a Grid layout or a List layout. Figure 3-4 shows an example of both a Grid layout and a List layout showing the same items.

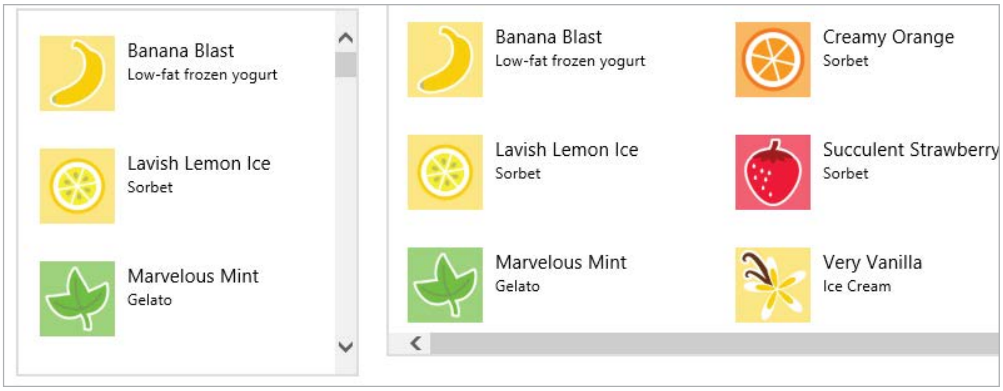


FIGURE 3-4 The List layout on the left and the Grid layout on the right show the same items

Using a List layout or a Grid layout starts with creating a ListView control. Suppose that you have the following set of data:

```
var myData = new WinJS.Binding.List([
        { title: "1", text: "One" },
        { title: "2", text: "Two" },
        { title: "3", text: "Three" },
        { title: "4", text: "Four" },
        { title: "5", text: "Five" },
        { title: "6", text: "Six" },
        { title: "7", text: "Seven" },
        { title: "8", text: "Eight" },
        { title: "9", text: "Nine" },
]);
```

A template and a ListView control that uses the template to show the data could look like this:

```
<div id="myTemplate" data-win-control="WinJS.Binding.Template" style="display: none">
    <div class="item">
        <h2 data-win-bind="innerText: title"></h2>
        <h3 data-win-bind="innerText: text"></h3>
    </div>
</div>

<div id="listView"
     class="win-selectionstylefilled"
     data-win-control="WinJS.UI.ListView"
     data-win-options="{ itemDataSource: myData.dataSource,
            itemTemplate: myTemplate,
            layout: { type: WinJS.UI.GridLayout }
        }"></div>
```

The ListView control takes the data and uses the template to display the data in a Grid layout. Changing to a List layout is simple: Just change the layout type to WinJS.UI.ListLayout.

Automatically changing between a Grid layout and a List layout is sometimes done when the available screen space of an app changes because it is snapped or unsnapped. You can find more details in Objective 3.3, later in this chapter.

The ListView control calculates the size of the first item and uses that size for all additional items. This calculation method might be a problem if you have items of different sizes. For this scenario, you have the WinJS.UI.CellSpanningLayout. You define the size of your base item and then let larger items span multiple cells, both horizontally and vertically. You still have a Grid layout, but now the items can have different sizes.

EXAM TIP

Although CellSpanningLayout is not mentioned directly in the exam objectives, it pays to be familiar with it. You can find instructions on how to use it at *http://msdn.microsoft.com/en-us/library/windows/apps/jj657974.aspx*.

A Grid layout also supports grouping items. You can add group headers and lay out the items according to their group.

If you take the previous example with ungrouped data, you can change it to the following code to create groups:

```
var myUngroupedData = new WinJS.Binding.List([
        { title: "1", text: "One" },
        { title: "2", text: "Two" },
        { title: "3", text: "Three" },
        { title: "4", text: "Four" },
        { title: "5", text: "Five" },
        { title: "6", text: "Six" },
        { title: "7", text: "Seven" },
        { title: "8", text: "Eight" },
        { title: "9", text: "Nine" },
]);

function compareGroups(leftKey, rightKey) {
    return leftKey.charCodeAt(0) - rightKey.charCodeAt(0);
}

function getGroupKey(dataItem) {
    return dataItem.text.toUpperCase().charAt(0);
}

function getGroupData(dataItem) {
    return {
        title: dataItem.text.toUpperCase().charAt(0)
    };
}

var myData = myUngroupedData.createGrouped(getGroupKey, getGroupData, compareGroups);
```

This code uses the createGrouped method on a list to group the data by its first character and sort the data alphabetically. The getGroupKey method returns the first character of the text property of an item. The getGroupData method returns an object containing a title property, and the compareGroups method controls the item order.

Now you can add a second template for the headers in your markup:

```
<div id="headerTemplate" data-win-control="WinJS.Binding.Template"
     style="display: none">
    <div class="simpleHeaderItem">
        <h1 data-win-bind="innerText: title"></h1>
    </div>
</div>
```

This template can then be used by ListView to render the headers for each group. The changes are shown in boldface:

```
<div id="listView"
     class="win-selectionstylefilled"
     data-win-control="WinJS.UI.ListView"
     data-win-options="{ itemDataSource: myData.dataSource,
            groupDataSource: myData.groups.dataSource,
            itemTemplate: myTemplate,
            groupHeaderTemplate: select('#headerTemplate'),
            layout: { type: WinJS.UI.GridLayout }
        }"></div>
```

This ListView control still uses GridLayout, but it now has a groupDataSource and a groupHeaderTemplate defined. You can view the result in Figure 3-5.

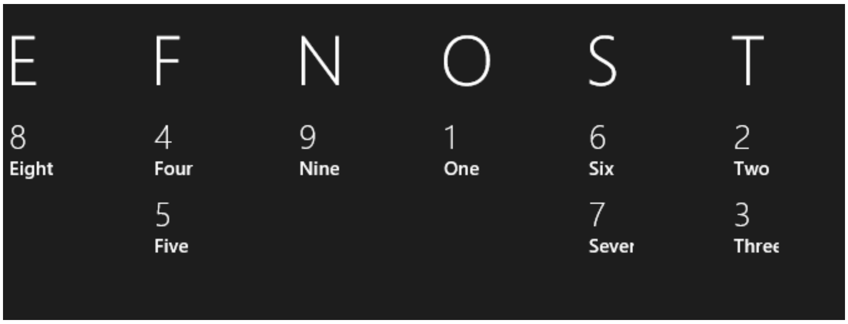


FIGURE 3-5 The grouped items in a Grid layout with a group header

Using a menu object

A flyout can be used as a pop-up to show some contextual information to the user. The menu is a special type of flyout that inherits from flyout and acts like a typical flyout. The most important difference is that the menu flyout can contain only objects of the WinJS.UI.MenuCommand type.

Your menu should be a direct child of the body element in your HTML. A typical menu can look like this:

```
<div id="myMenu" data-win-control="WinJS.UI.Menu">
    <button data-win-control="WinJS.UI.MenuCommand"
            data-win-options="{id:'toggleItem',label:'Check me',
                              type:'toggle', selected:'true'}"></button>
    <hr data-win-control="WinJS.UI.MenuCommand"
        data-win-options="{id:'separator',type:'separator'}" />
    <button data-win-control="WinJS.UI.MenuCommand"
            data-win-options="{id:'firstItem',label:'First'}"></button>
    <button data-win-control="WinJS.UI.MenuCommand"
            data-win-options="{id:'secondItem',label:'Second'}"></button>
    <button data-win-control="WinJS.UI.MenuCommand"
            data-win-options="{id:'thirdItem',label:'Third'}"></button>
</div>
```

This menu starts with a command of type toggle, which means a user can click the item that automatically adds or removes a check mark on the item. The second item is a separator, and the other items are regular menu options.

As with a normal flyout, you need some JavaScript to show the menu and attach event handlers to the different options:

```
document.getElementById("openMenuButton").addEventListener("click", showMenu, false);
document.getElementById("toggleItem").addEventListener("click", toggle, false);
```

This code uses an openMenuButton HTML element and attaches a showMenu method. It also attaches a handler to the toggleItem menu option. The handlers can look like this:

```
function showMenu() {
    var flyout = document.getElementById('myMenu');
    var button =  document.getElementById("openMenuButton");

    flyout.winControl.show(button, "bottom");
}

function toggle() {
    var state = document.getElementById("toggleItem").winControl.selected;
}
```

Of course, you can also add handlers for all the other menu items. You can process an action to respond to the user's choice inside those handlers.

The flyout results in the sample menu are shown in Figure 3-6.

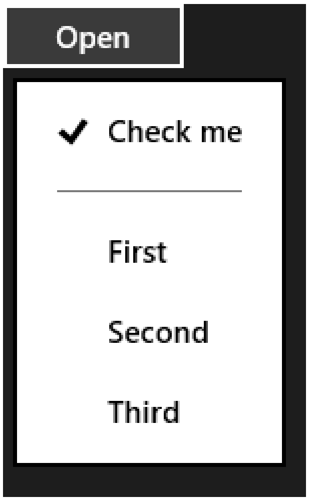


FIGURE 3-6 The sample menu with several menu items

You can show menus in many places in your UI. The previous code could be used to attach a menu to a button in your app bar or some other content in your app.

A menu can be useful if it is attached to the header of your app, which is mostly a cosmetic change in which you add a chevron (⌄) after the header and attach a menu to the click event. The chevron appears so that users know to click the header.

You can add a header with a chevron with the following HTML:

```
<header aria-label="Header content" role="banner">
    <button class="win-backbutton" aria-label="Back"></button>
    <div class="titlearea win-type-ellipsis" id="title">
        <button class="titlecontainer">
            <h1>
                <span class="pagetitle">Header</span>
                <span class="chevron win-type-x-large">&#xe099;</span>
            </h1>
        </button>
    </div>
</header>
```

Here you use the code &#xe099; to display a chevron. In your JavaScript, you can now add an event handler for when the user clicks the title area:

```
document.querySelector(".titlearea").addEventListener("click", showMenuAtHeader, false);
```

The following method shows the menu and aligns it correctly with the header:

```
function showMenuAtHeader() {
    var flyout = document.getElementById('myMenu').winControl
    var anchor = document.querySelector("header .titlearea");

    flyout.show(anchor, "bottom", "left");
}
```

This method gives you the result shown in Figure 3-7.

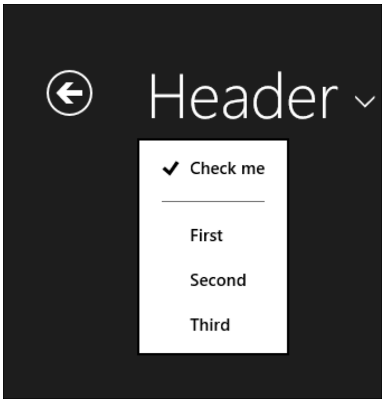


FIGURE 3-7 The sample menu attached to the header

You might consider changing the header when users navigate to another page using this menu. They then have a visual clue of where they are in your app.

Using a WebView control

Windows Store apps enable you to load a webpage and use it as part of your app. You could previously use an i-frame, but it didn't feature much content isolation or navigation functionality.

Microsoft has improved the situation with its new WebView control, which can render HTML5 content and use HTML5 functionality (except IndexedDB, the app cache, Geolocation, and the Clipboard). You can also integrate with the navigation history of the WebView control to navigate forward and backward.

Another useful feature is the ability to call scripts that are processed in the WebView control and to return results from the WebView control to your app. The webpage you are showing can then integrate with your app and complement it.

A WebView control can load content in a couple of different ways. The easiest way is to load an external URL and show it in your app like this:

```
<x-ms-webview id="webview" src="http://go.microsoft.com/fwlink/?LinkId=294155"
    style="width: 400px; height: 400px;">
</x-ms-webview>
```

This code shows the Windows Dev Center page in the WebView control in your app. You also see that an easy way to pass data to your webpage is by using the query string. JavaScript code in your webpage can parse the query string and use the parameters you sent.

In addition to loading an external webpage, you can load a piece of HTML directly from a string. The following code takes a WebView control with an ID of webview and calls the navigateToString method to show the string as HTML:

```
var htmlString = "<!DOCTYPE html>" +
        "<html>" +
        "<head><title>Simple HTML page</title></head>" +
        "<body>" +
            "<h1>Hi!</h1>" +
            "<p>This is a simple HTML page.</p>" +
        "</body>" +
        "</html>";
document.getElementById("webview").navigateToString(
    htmlString);
```

Another way to load an HTML page is to load an HTML file from your application data folder. Suppose that you have included an HTML file in your app package that you want to display in a WebView control. The following code takes the HTML file from your package and copies it to a local folder. It then points the WebView control to the local file:

```
Windows.Storage.ApplicationData.current.localFolder
    .createFolderAsync("Html", Windows.Storage.CreationCollisionOption.openIfExists)
    .then(function (htmlFolder) {
        Windows.ApplicationModel.Package.current.installedLocation
            .getFileAsync("html\\example.html")
            .then(function (htmlFile) {
                return htmlFile.copyAsync(htmlFolder, "example.html",
                    Windows.Storage.CreationCollisionOption.failIfExists);
            });
    }).done(function (e) {
        document.getElementById("webview").navigate(
            "ms-appdata:///local/Html/example.html");
    });
```

Because all file input/output (I/O) is done asynchronously, this code depends on the use of promises to make sure that the next piece of code runs when the asynchronous action finishes.

The WebView control uses the ms-appdata protocol to point it to a local file. This protocol is a special format for the URI that tells Windows to look at its own file system instead of a remote location.

There is one final way to load data. When you don't have direct access to an HTML file or an external URL, you can stream the data to the WebView control. Streaming can be useful when the external URL is encrypted, and you have some code that should first decrypt the content before showing it in your app.

You can't stream content from JavaScript. Instead, you should create a WinRT object in C# or C++. That class inherits from the IUriToStreamResolver interface. The URI that gets passed to this class has this form:

```
ms-local-stream://<package name>_<encoded contentIdentifier>/<relativePath>
```

By parsing this URI, your WinRT object can identify what data the WebView control wants to load. It can then retrieve this data and perform any required actions on it before streaming it to the user.

Now that you know how to load data for the WebView control, the next important step is to communicate with the WebView control and integrate it in your app.

Integrating with the navigation features of a WebView control is easy. The WebView control exposes two important properties:

- canGoBack
- canGoForward

These two properties return a Boolean value signaling whether the WebView control can go a page back (because the user navigated from one page to another) or forward (because the user navigated back from a page). If any of these properties returns true, you can call one of the corresponding goBack or goForward methods.

Another very important integration feature is the ability to invoke scripts that are loaded in a WebView control. You do so by calling the invokeScryptAsync method. This method takes the name of the script that you want to run and any parameters that you want to pass and returns an object of type MSWebViewAsyncOperation. By calling start on the async operation, you invoke this script:

```
document.getElementById("webview").invokeScriptAsync("myFunction", 42).start();
```

In this case, you call a method named myFunction and pass it one parameter with a value of 42.

You can also retrieve data from the WebView control in your app by calling window.external.notify from within the webpage loaded in your WebView control. You can pass any argument you want to the notify function. Inside your app, you subscribe to the MSWebViewScriptNotify event, which gets called whenever the notify method is called from within the WebView control.

MORE INFO **HTML WEBVIEW CONTROL SAMPLE**

An example of how to use the WebView control can be found at *http://code.msdn.microsoft.com/windowsapps/HTML-WebView-control-sample-56e773fa*.

Using an item container

When displaying groups of items, you have several choices. The FlipView control is there when you want to enable users to easily flip through a series of items. The ListView control uses the Grid layout and the List layout to show items to the user. A ListView control also offers a lot of functionality that you can use.

If you want a simpler solution, use the item container. Items placed inside an item container automatically support swipe, drag-and-drop, and hover functionality. The following markup creates two item containers that are selectable:

```
<div id="item1"
     data-win-control="WinJS.UI.ItemContainer"
     data-win-options="{tapBehavior: 'toggleSelect'}"
     style="width: 300px;">
    <div style="margin: 10px; padding: 10px; background-color: lightgray">
        <div class="win-type-x-large"
             style="margin-bottom: 5px;">
            Title 1
        </div>
        <div>Description 1</div>
    </div>
</div>
<div id="item2"
     data-win-control="WinJS.UI.ItemContainer"
     data-win-options="{tapBehavior: 'toggleSelect'}"
     style="width: 300px;">
    <div style="margin: 10px; padding: 10px; background-color: lightgray">
        <div class="win-type-x-large"
             style="margin-bottom: 5px;">
            Title 2
        </div>
        <div>Description 2</div>
    </div>
</div>
```

This code gives you the result shown in Figure 3-8.

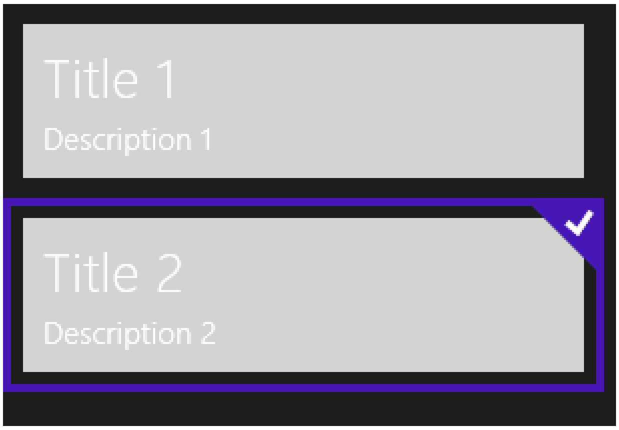


FIGURE 3-8 The two item containers with the bottom one selected

Of course, it's not always ideal to have to create all items by hand. (In the next section, you will look at the Repeater control that can help you automate.)

As shown in Figure 3-8, an item container can be selected, which is defined by the selectionDisabled property. By default, this property is false, allowing items to be selected. You can change it to true to disable selection:

```
document.getElementById('item1').winControl.selectionDisabled = true;
```

You can also implement dragging with an item container. There are several events involved with dragging:

- dragstart
- dragover
- dragenter
- dragleave
- drop

The following code shows how to implement dragging. The markup is as follows:

```
<div id="item1"
     data-win-control="WinJS.UI.ItemContainer"
     data-win-options="{draggable: true, selectionDisabled: true}">
    Item
</div>
<div id="dropTarget" style="width:100px;height:100px">
    Drop here!
</div>
```

Here is the JavaScript:

```
document.getElementById('item1').addEventListener("dragstart", function (e) {
    e.dataTransfer.effectAllowed = 'copy';
    e.dataTransfer.setData('Text', "Hello World!");
});
```

```
var dropTarget = document.getElementById("dropTarget");
dropTarget.addEventListener("dragover", function (eventObject) {
    eventObject.preventDefault();
});

dropTarget.addEventListener("dragenter", function (eventObject) {
    //Add Border
    WinJS.Utilities.addClass(dropTarget, "drop-ready");
});

dropTarget.addEventListener("dragleave", function (eventObject) {
    WinJS.Utilities.removeClass(dropTarget, "drop-ready");
});

dropTarget.addEventListener("drop", function (eventObject) {
    WinJS.Utilities.removeClass(dropTarget, "drop-ready");

    var itemText = eventObject.dataTransfer.getData('Text');

    document.getElementById("dropTarget").innerText = itemText;
});
```

This code allows you to drag item1 and drop it on the target div, which changes the content of the target div to Hello World.

MORE INFO **HTML ITEMCONTAINER CONTROL SAMPLE**

An example of how to use the ItemContainer control can be found at *http://code.msdn.microsoft.com/windowsapps/HTML-ItemContainer-Sample-932b2817/*.

Using the Repeater control

The Repeater control is somewhat similar to a ListView control. It is more flexible than a ListView control, but it doesn't offer advanced features such as control over how data items are loaded.

What it does offer is an easy way to repeat some markup (being HTML and WinJS controls) for a list of data. Repeater controls can even be nested to create advanced data binding scenarios.

A Repeater control works in the same way as the other data binding controls. You first define the data you want to bind to:

```
var data = new WinJS.Binding.List(
    [
        { title: "Item 1" },
        { title: "Item 2" },
        { title: "Item 3" },
        { title: "Item 4" }
    ]);
```

Then you need a template for each item:

```
<div id="listTemplate" data-win-control="WinJS.Binding.Template">
    <li data-win-bind="textContent: title"></li>
</div>
```

Finally, you need the Repeater control:

```
<ul data-win-control="WinJS.UI.Repeater"
    data-win-options="{data: data, template: select('#listTemplate')}">
</ul>
```

Now the Repeater control renders a list of HTML elements showing items 1 to 4.

Similar to the FlipView control, you can also use a templating function to generate your HTML in JavaScript.

Because you are binding to a List, you can dynamically add and remove items from it, which causes the Repeater control to update. By using CSS and WinJS animations, you can create good update effects when items are added and removed.

If you take the previous example, you can add an add and remove button with the following HTML:

```
<div>
        <button id="addCmd">Add item</button>
        <button id="removeCmd">Remove item</button>
</div>
```

To implement adding and removing items, follow these two steps:

1. Add event listeners for the buttons.
2. Add event listeners to the repeater for the animations.

Because of the data binding functionality implemented in WinJS, it's easy to add and remove items to the list, which are then added to the repeater. The following code from the activated event adds event handlers to the two buttons:

```
var add = document.getElementById("addCm\d");

add.addEventListener("click", function (ev) {
    if (!animation) {
        animation = true;
        numberOfItems++;
        data.push({ title: "Item " + numberOfItems });
    }
});
```

```
var remove = document.getElementById("removeCmd");
remove.addEventListener("click", function (ev) {
    if (!animation && numberOfItems > 0) {
        animation = true;
        numberOfItems--;


        data.pop();
    }
});
```

These two event handlers set a local variable animation to true, update the number of items, and then update the binding list that contains the actual data.

Without doing anything else, you can now add and remove items to the repeater. However, Windows Store apps should use animations to provide the fast and fluent interface.

The following code registers itself for the iteminserted event on the repeater to make sure that the correct animation is used:

```
var repeaterElement = document.getElementById("repeater");
var repeater = repeaterElement.winControl;
repeater.addEventListener("iteminserted", function (ev) {
    var a = WinJS.UI.Animation.createAddToListAnimation(ev.affectedElement);
    a.execute().then(function () {
        animation = false;
    });;
});
```

The animation is processed and returns a promise that finishes when the animation is done. By setting animation to false, new items can be added or removed from the repeater.

Removing an item follows the same pattern:

```
repeater.addEventListener("itemremoved", function (ev) {
    var affectedElement = ev.affectedElement;
    repeaterElement.appendChild(affectedElement);
    var a = WinJS.UI.Animation.createDeleteFromListAnimation(affectedElement);

    this.affectedElement = affectedElement;
    var that = this;

    // Return a promise to the Repeater so that I can dispose the item after we have 
animated it
    // and removed it from the DOM
    ev.setPromise(a.execute().then(function () {
        repeaterElement.removeChild(that.affectedElement);
        animation = false;
    }));
});
```

Here you create the correct animation and then store a reference to the affected item to make sure that you can remove the item from the repeater after the animation finishes. If you don't do this, the item is animated, doesn't show in the UI, but isn't removed from memory.

Adding animations is very easy and helps you create a better app.

MORE INFO **HTML REPEATER CONTROL SAMPLE**

An example of how to use the Repeater control can be found at *http://code.msdn.microsoft.com/windowsapps/HTML-Repeater-control-da22d278*.

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.

Look at the Weather app that is installed by Windows and make a list of the controls that you recognize in the app.

Objective summary

- The FlipView control can be used to show items to a user one at a time. The user can easily flip through the different items.
- Flyouts can be used to show a pop-up to the user that offers extra information or asks the user for input or confirmation.
- When using a ListView control, you can use both Grid and List layouts to show items to the user.
- The menu object is a special type of flyout that uses menu commands.
- A WebView control can be used to load HTML content in your app. This data can be external or come from a string, a local file, or a stream. You can invoke scripts on the webpage and listen for notifications from the webpage.
- Using the ItemContainer control is an easy way to create items that support selection and dragging.
- The Repeater control can be used to render HTML content for a list of items.

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 have added a FlipView control to your app and you want the user to be able to interact with the items you are showing in your FlipView control. What should you do?
   - **A.** Use an itemTemplate property and bind to the events of the HTML elements that get rendered.
   - **B.** Use a WinJS binding List as a data source.
   - **C.** Bind to the oniteminvoked event of the ListView control.
   - **D.** Use a templating function and add event handlers through JavaScript.
2. You want to group data in a List. Which methods do you have to create? (Choose all that apply.)
   - **A.** A method that goes through all data to create groups.
   - **B.** A method that returns the group key.
   - **C.** A method that compares individual items.
   - **D.** A method that returns descriptive data for each group.
3. You want to call a JavaScript method on a webpage that is loaded in a WebView control. Which method should you use?
   - **A.** IUriToStreamResolver
   - **B.** window.external.notify
   - **C.** MSWebViewScriptNotify
   - **D.** invokeScriptAsync

Objective 3.2: Implement HTML layout controls

Your app will be used on a multitude of devices, including large monitors with touch screens, regular PCs with a keyboard and mouse, and different types of tablets. Making sure that your app runs well on all those different devices is the subject of this objective. By using a combination of CSS3 and WinJS controls, you can create great apps that work well on all devices.

This objective covers how to:

- Implement layout controls to structure your layout
- Implement templates and bindings
- Support scrolling and zooming with CSS3
- Manage text flow and presentation, including overflow

Implementing layout controls to structure your layout

After you have an idea for your app, start with laying out the basic structure of your app. As discussed in Chapter 1, using the default Visual Studio project templates can be a great way to start defining your layout.

However, you're not finished after you pick a project template. Your app has specific needs: For example, a section of a page might need a specific layout or you are developing a custom control.

When developing your app, you have four layout types to choose from: Flexible Box, Grid, Multi-column, and Regions.

Flexible Box (FlexBox) layout

A Flexible Box (FlexBox) layout is intended for laying out complex scenarios in which the relative size and position of elements stays the same on different screen sizes.

FlexBox layouts are ideal for apps that use digital print media, such as newspapers or magazines.

Start with the following HTML:

```
<div class="myFlexBox myFirstFlexBox">
    <div>1</div>
    <div>2</div>
    <div>3</div>
</div>

<div class="myFlexBox mySecondFlexBox">
    <div>1</div>
    <div>2</div>
    <div>3</div>
</div>
```

You can add a FlexBox layout to the two containing divs with this CSS:

```
.myFirstFlexBox {
    flex-direction: row;
}

.mySecondFlexBox {
    flex-direction: column;
}

.myFlexBox {
    width: 200px;
    height: 100px;
    border: 2px solid white;
    display: -ms-flexbox;
}
    .myFlexBox div {
        background-color: red;
        width: 50px;
        height: 25px;
        text-align: center;
        color: white;
    }
        .myFlexBox div:first-child {
            background-color: green;
        }

        .myFlexBox div:last-child {
            background-color: blue;
        }
```

You specify that you want to use a FlexBox layout by using the display: -ms-flexbox property. The first div uses a flex-direction: row; the second one uses a flex-direction: column. You can view the results in Figure 3-9.

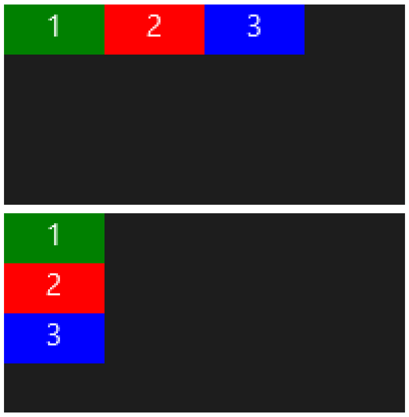


FIGURE 3-9 The two divs with a FlexBox layout; the top one shows a row direction, and the bottom one shows a column direction

You can configure your FlexBox layout with additional properties such as -ms-flex-align. This property has one of the following values: start, end, center, stretch, or baseline. These values specify the alignment of your child elements perpendicular to the direction of your FlexBox layout. You can center items in your FlexBox or place them at the top or bottom of the containing element.

Another property is -ms-flex-pack, which can be set to start, end, or justify. Because the elements you place in your FlexBox layout can have remaining white space, you have to decide what do with it. For example, if you choose justify, the white space is evenly distributed between all elements.

Because a FlexBox layout is meant to distribute its child items, -ms-flex-wrap is an important option. By default, the FlexBox layout puts all items in one row or column, depending on the direction. By using the -ms-flex-wrap property, you can specify that the FlexBox layout should wrap (or wrap-reverse) the items it contains.

You can also control the size of the items inside the FlexBox layout. Instead of specifying a hard-coded height or width, you can configure how items should resize themselves inside the FlexBox layout. Use the following syntax: -ms-flex: *<positive-flex>* *<negative-flex>* *<preferred-size>*.

For example, using -ms-flex: 2; on one of the three items in the previous example makes that item twice as large as the other two items. Your FlexBox layout automatically uses any additional space while keeping the relative sizes of its child elements.

Grid layout

A CSS3–based Grid layout allows you to define rows and columns and assign items to the cells that you create. As with the FlexBox layout, you start by configuring the display property and setting it to -ms-grid:

```
#myGrid {
  display: -ms-grid;
  background: orange;
}
```

You can add columns and rows to your grid. All cells can have a fixed size, but it’s more interesting to define some columns and rows that use the available screen size to allow for more content.

When defining column and row sizes, use the following units:

- **standard length unit** A percentage of the object’s width (for columns) or height (for rows)
- **auto keyword** Indicates that the width of the column or height of the row is sized based on the items inside of it
- **min-content keyword** Indicates that the minimum width or height of any child elements is used as the width or height

- **max-content keyword** Indicates the maximum width or height of any child elements used as the width or height
- **minmax(a, b) keyword** Indicates the width or height between a and b, as available space allows
- **fraction unit (fr)** Indicates that the available space should be divided among the columns or rows according to their fraction values

Look at the following example:

```
#myGrid {
  display: -ms-grid;
  background: orange;
  -ms-grid-columns: auto 100px 1fr 2fr;
  -ms-grid-rows: 50px 25px auto;
}
```

This example defines four columns and three rows. The first column autosizes; the second one is always 100 pixels (px). The last two columns occupy the remaining space with a factor of 1:2, which means that column 3 is 1/3, and column 4 is 2/3. The rows are 50px, 20px, and autosize.

After defining the rows and columns of the grid, you can start adding items by specifying to which row and column an item belongs:

```
#item1 {
  border: yellow solid 1px;
  -ms-grid-row: 1;
  -ms-grid-column: 1;
}
```

This example puts an item in row 1, column 1. You can also align items in their cells. You can use both -ms-grid-column-align and -ms-grid-row-align with a value of center, end, start, or stretch.

If you assign multiple items to the same cell, these items are stacked with the last item added on top. To configure it, use the z-index property. Where x and y position an item horizontally and vertically, z defines how far away an item is. Items with a greater z-index are positioned on top of items with a smaller z-index.

If you are dealing with larger items, they can span multiple cells with the -ms-grid-column-span and -ms-grid-row-span properties. For example, a -ms-grid-column-span value of 2 enables an item to occupy two adjacent columns. The same is true for -ms-grid-row-span; a value higher than 1 lets the item span multiple rows.

The following markup and CSS demonstrate the properties you can set on a Grid layout:

```
<div id="myGrid">
    <div id="item1">Item 1</div>
    <div id="item2">Item 2</div>
    <div id="item3">Item 3</div>
    <div id="item4">Item 4</div>
    <div id="item5">Item 5</div>
</div>
#myGrid {
    display: -ms-grid;
    background: gray;
    border: blue;
    -ms-grid-columns: auto 1fr 2fr;
    -ms-grid-rows: 200px auto;
}
    #myGrid div {
        background: lightgray;
        border: red solid 1px;
    }

#item1 {
    background: maroon;
    -ms-grid-row: 1;
    -ms-grid-column: 1;
}

#item2 {
    -ms-grid-row: 1;
    -ms-grid-column: 2;
}

#item3 {
    -ms-grid-row: 1;
    -ms-grid-column: 3;
}

#item4 {
    -ms-grid-row: 2;
    -ms-grid-column: 1;
}

#item5 {
    -ms-grid-row: 2;
    -ms-grid-column: 2;
    -ms-grid-column-span: 2;
}
```

The result is shown in Figure 3-10. All items are assigned to a Grid cell, and Item 5 spans two columns. The first column is autosized, and the second and third column fill the additional space with a ratio of 1:2. The first row has a static height; the second row is autosized.

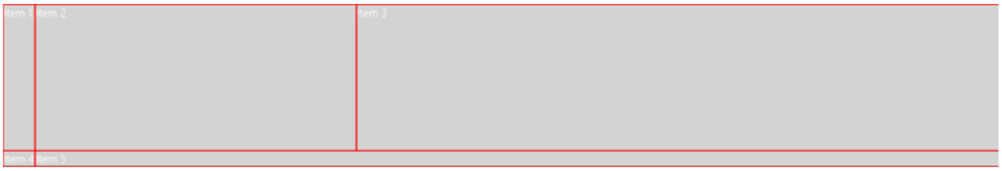


FIGURE 3-10 The Grid with three columns and two rows, showing five items

Multi-column layout

A Multi-column layout doesn't use a fixed amount of columns; it allows columns to be added based on screen size. Items can flow to other columns, and these columns can have a gap and a rule between them.

Suppose that you have a large amount of text and you want to show this text in columns so that the text automatically overflows to the next column. The following CSS creates a Multi-column layout with a gap between the columns and a white ruler:

```
#multicolumn {
    columns: auto 12em;
    column-gap: 1em;
    column-rule-width: 1em;
    column-rule-style: solid;
    column-rule-color: white;
}
```

Using this layout on some lorem ipsum text gives you the result shown in Figure 3-11.

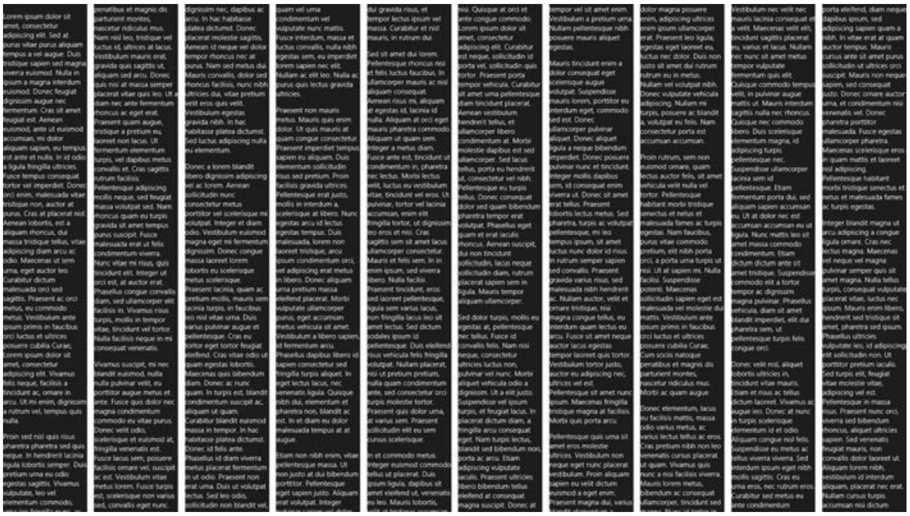


FIGURE 3-11 The example of a Multi-column layout

This example uses the columns: auto 12em; property to create a variable number of columns with a size of 12em.

It also uses the column gap and ruler options to configure the size and color between the columns. The previous example creates white gaps with the following CSS:

```
 column-gap: 1em;
 column-rule-width: 1em;
 column-rule-style: solid;
 column-rule-color: white;
```

You can influence when a break is inserted by using the break-before, break-inside, and break-after properties with a value of always, auto, avoid, empty string (same as auto), inherit, left, right, page, column, avoid-page, and avoid-column.

You can also let an item span multiple columns. Maybe you have a title for an article or an image that should span more than one column. You do so by specifying column-span: all on an item (you can't specify an exact number of columns to span).

When you let Internet Explorer (IE) automatically create columns and add breaks, one column might contain more text then another column. To evenly spread your text over all the columns, you can use the column-fill: balance property.

MORE INFO **SAMPLE MULTI-COLUMN LAYOUT**

Microsoft released a sample showing how to use CSS to create a Multi-column layout. Although not specific to Windows Store apps, you can use the sample to get a better understanding of how to use these concepts in your app. You can find the sample at *http://code.msdn.microsoft.com/ie/Multi-column-layout-4f467ffd*.

Regions layout

The final layout option is the Regions layout, which lets you specify elements as containers and then take an existing HTML file and spread its content over your containers. The Regions layout allows you to create a very specific layout in which you position your containers and then stream an HTML document through them.

The source HTML document is loaded through a hidden iframe, which you can add to your markup like this:

```
<iframe id="flow-data-source" src="content-source.html"></iframe>
```

The content-source.html file can include an HTML page as complex as you want. For this sample, a simple HTML file is used with the following content:

```
<div style="color:white">Content goes here 1.</div>
<div style="color:white">Content goes here 2.</div>
<div style="color:white">Content goes here 3.</div>
<div style="color:white">Content goes here 4.</div>
<div style="color:white">Content goes here 5.</div>
<div style="color:white">Content goes here 6.</div>
```

Now that you have loaded the data into an iframe, use CSS to create the flow data source:

```
#flow-data-source {
  -ms-flow-into: flow1;
}
```

You can add containers to your markup to flow the content of the source HTML file into. div elements are usually used as a container:

```
<div class="flow1-container"></div>
<div class="flow1-container"></div>
<div class="flow1-container"></div>
```

You can use CSS to flow the content of the iframe through the three div elements:

```
#flow-data-source {
  -ms-flow-into: flow1;
}
.flow1-container {
  -ms-flow-from: flow1;
  border: 1px solid red;
  width:100px;
  height:100px;
}
```

That is all you need to know about the Regions layout. You specify a data source with a flow into and containers with a flow from property. Of course, you can make the Regions layout as complex as you want. By using the Regions layout inside a Grid layout, you can assign containers to different cells and have the content of your HTML source document flow through them.

Implementing templates and bindings

Data binding is an essential feature for easily showing data on-screen and responding to user actions that change your data.

One property that is important for data binding is WinJS.Binding.optimizeBindingReferences. If you search for this property in the WinJS library, you see that it is always initialized on application instantiation.

You can bind data to any JSON object. Suppose that you have the following span element:

```
<span data-win-bind="innerText: message" id="messageSpan"></span>
```

You can use some JavaScript to bind the innerText property of this span to a value:

```
var message = { message: "Hello World!" };
var messageSpan = document.getElementById("messageSpan");
WinJS.Binding.processAll(messageSpan, message);
```

The attribute that specifies the binding configuration is the data-win-bind property. Here you can specify a property (innerText, in this case) and the name of a property that comes from the object that you bind to your element.

WinJS.Binding.processAll searches through your HTML for the data-win-bind attribute and processes the binding configuration for all elements it finds.

By default, data binding is one-way only; changes don't automatically update the UI. To facilitate two-way data binding, WinJS added the WinJS.Binding.as method. This method takes your object and returns an observable object that you can then use to automatically update the UI:

```
var message = { message: "Hello World!" };
var messageSpan = document.getElementById("messageSpan");
WinJS.Binding.processAll(messageSpan, message);
var bindingSource = WinJS.Binding.as(message);

document.getElementById("updateDataButton").addEventListener("click", function () {
    bindingSource.message = "Bye World!";

});
```

EXAM TIP

Make sure that you understand that data binding is one-way by default and you need to add two–way data binding by using WinJS.Binding.as.

If you add a button with an ID of updateDataButton to your UI, you can click it to update the message property of your bindingSource. Doing so automatically updates the value of the span element.

You can imagine scenarios in which you have a central object with some data that is bound to multiple objects on your screen. If the user changes something, or an external notification comes in, your app updates the object, and your UI automatically updates.

Many frameworks can help you with data binding (KnockoutJS, Angular, and Durandal, for example). However, for basic data-binding scenarios, WinJS has all the functionality you need.

Previous sections discussed templates, and you used them several times. You used templates in combination with binding data to a ListView control or FlipView control. A template describes the markup that you want to use for each item that gets generated through data binding.

Templates are added as regular divs to your markup. When WinJS initializes, it processes all templates and removes them from the DOM. You mark a div as being a template by using the data-win-control attribute with a value of WinJS.Binding.Template:

```
<div id="templateDiv" data-win-control="WinJS.Binding.Template"></div>
```

To change the previous example of rendering a simple message to use templating, you can use the following HTML:

```
<div id="templateDiv" data-win-control="WinJS.Binding.Template">
    <div>
        <span data-win-bind="innerText: message"></span>
    </div>
</div>
<div id="templateTargetDiv"></div>
```

Now you can use JavaScript to render the template and place the result in templateTargetDiv:

```
var message = { message: "Hello World!" };

var templateElement = document.getElementById("templateDiv");
var renderElement = document.getElementById("templateTargetDiv");
renderElement.innerHTML = "";

var templateControl = templateElement.winControl;

templateElement.winControl.render(message, renderElement);
```

You are not required to create a target div for the rendering operation. The render method returns a promise that will get a newly created div as a parameter that you can then add to the DOM.

By using reusable templates and data binding, you can easily separate your UI from the data it shows. Instead of retrieving DOM elements and manually setting their properties (which is error-prone and hard to maintain), you use a declarative syntax that results in a more maintainable code.

Supporting scrolling and zooming with CSS3

Your app probably has multiple pages with navigation functionality to move from page to page. However, sometimes you have a single page in which the user can also scroll and zoom. Maybe you have an image or some other content that doesn’t fit directly on-screen and you want the user to navigate through your page.

Scrolling, or panning, is a motion used to slide through some content. Especially with actions like these, you have to think about touch gestures. Users are used to swiping or sliding through an app by using one or more fingers, and you should support it in your apps.

Enabling panning for an area is done through CSS. Suppose that you have container element that is hosting an image as a child:

```
<div class="Container Horizontal">
    <img alt="Cliff" src="/images/Cliff.jpg" />
</div>
```

The following CSS first restricts the size of the container and then defines the different panning options:

```
.Container {
    width: 480px;
    height: 270px;
}

.None {
    overflow: hidden;
}

.Horizontal {
    overflow-x: auto;
    overflow-y: hidden;
}

.Vertical {
    overflow-x: hidden;
    overflow-y: auto;
}

.Unrailed {
    overflow: auto;
    -ms-scroll-rails: none;
}

.Railed {
    overflow: auto;
    -ms-scroll-rails: railed;
}
```

The previous HTML fragment defines the Horizontal class. The result is shown in Figure 3-12.

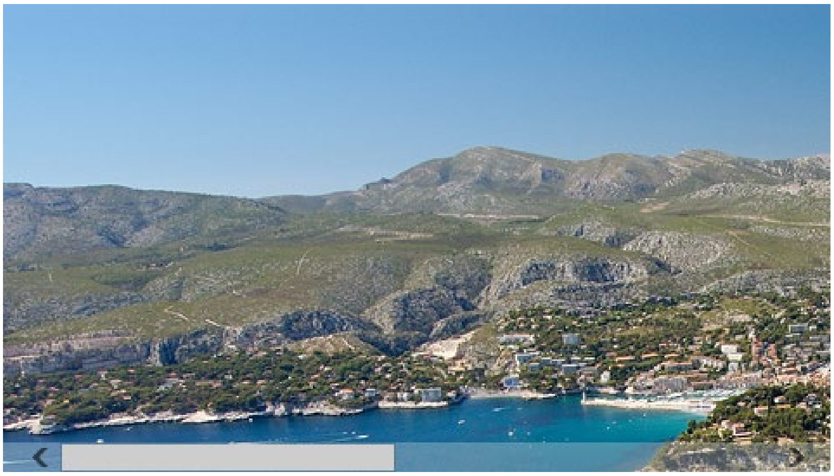


FIGURE 3-12 The image element in a container with horizontal panning

By switching the CSS class, you can implement vertical panning, unrailed (free to move in any direction), or railed (locked to an axis).

The previous code sample allows a user to pan freely through your content. But maybe that isn't what you want. For example, if you have a couple of items that you want the user to pan through one item at a time, you can't use the free mode that you saw before. It would enable a user to stop panning in the middle of an item instead of panning one item at a time.

To help your users pan one item at a time, you can implement snapping. With the -ms-scroll-snap-type property, you can use one of the following snap types:

- **None** Panning and scrolling are unaffected by any defined snap points.
- **Proximity** When panning comes near a snap point, the item is snapped to land exactly on the snap point.
- **Mandatory** Content always lands on a snap point.

Just like panning, zooming is also implemented through CSS. The following CSS allows the user to zoom in and out of an element:

```
.zoomElement {
    overflow: auto;
    -ms-content-zooming: zoom;
    -ms-scroll-rails: none;
    -ms-content-zoom-limit-min: 50%;
    -ms-content-zoom-limit-max: 500%;
}
```

Zooming is enabled by using the -ms-content-zooming property. You also have to specify a minimum and maximum zoom level. A minimum that is smaller than 100 percent allows a user to zoom out, but it leaves empty room in your container element.

You can combine both zooming and panning in one view, which allows a user to zoom in and out and navigate through your element.

When panning and zooming, it's also very natural to use touch gestures. You can control the way a region can be manipulated through touch by using the touch-action CSS property. This property can have one of the following values:

- **Auto, Initial value** Indicates that the Windows Store app using JavaScript determines the permitted touch behaviors for the element.
- **None** The element doesn't permit default touch behaviors.
- **Pan-x** The element permits touch-driven panning on the horizontal axis. The touch pan is performed on the nearest ancestor with horizontally scrollable content.

- **Pan-y** The element permits touch-driven panning on the vertical axis. The touch pan is performed on the nearest ancestor with vertically scrollable content.
- **Pinch-zoom** The element permits pinch-zooming. The pinch-zoom is performed on the nearest ancestor with zoomable content.
- **Manipulation** The element permits touch-driven panning and pinch-zooming. This is the shorthand equivalent of “pan-x pan-y pinch-zoom”.
- **Cross-slide-x** The element permits cross-sliding along the horizontal axis.
- **Cross-slide-y** The element permits cross-sliding along the vertical axis.

Make sure to test these actions and align them to the default behavior that users are accustomed to.

MORE INFO **HTML SCROLLING, PANNING, AND ZOOMING SAMPLE**

Microsoft released a sample that shows how to implement scrolling, zooming, and panning. You can find the sample at *http://code.msdn.microsoft.com/windowsapps/scrolling-panning-and-47d70d4c/*.

Managing text flow and presentation, including overflow

Your app will probably contain text. Depending on the type of app you build, text might even be the most important aspect of your UI.

Making sure that your text is nicely presented on a multitude of devices is therefore very important. The CSS files for Windows Store apps already define standard font styles using the Segoe UI font family. When using the Visual Studio project templates, the WinJS style sheet is referenced and provides the correct font definitions for your app.

Maybe your brand has a particular font that you want to reflect in your app. You can add custom fonts by using the @font-face CSS rule. By including a TTF file in your app package, you can reference it and create a font face like this:

```
@font-face {
    src: url('/images/fonts/MyFont.ttf') format('truetype');
    font-family: MyFont;
    font-style: normal;
    font-weight: 500;
    font-stretch: normal;
    font-variant: normal;
}
```

MORE INFO **CSS TYPOGRAPHY JAVASCRIPT SAMPLE**

For more information about typography, see the sample at *http://code.msdn.microsoft.com/windowsapps/typography-JS-sample-e2df9eb4*.

Another important aspect of presenting text is dealing with overflow. Maybe the content you are trying to present fits nicely on your device, but that doesn't mean all devices have a large enough screen space. Or maybe you can't anticipate the content that users will add and that you need to display.

One way to deal with variations is to use an ellipsis (…). When text is too long for its container, shorten the text and add an ellipsis to the end, signaling the user that there is more content.

CSS has a text-overflow property that you can set to ellipsis to automatically generate this effect. WinJS defines the following class that you can apply to your elements:

```
.win-type-ellipsis {
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
```

This CSS property works only on single-line content, however. If you have multiple lines and you want to add an ellipsis to the last line, you need some custom JavaScript to do this.

Thought experiment

Laying out your content

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 been asked to prepare a presentation for a customer who is looking for suggestions for building a Windows 8 app. The app should be used by internal employees while they are at the office or on the road on multiple devices. They visit restaurants to perform quality checks, which include taking notes and pictures. They are now using a standard desktop application at the office and printed forms when at customer sites.

Look through this chapter and describe the key points you can make to convince your customer that a Windows 8 app would be useful.

Objective summary

- You can use FlexBox, Grid, Multi-column, or Regions layouts.
- Data binding is essential for showing data on the screen. Templates can be used to create reusable pieces of markup that can bind to data and render on-screen.
- Panning can be implemented by using the CSS overflow property. By using -ms-scroll-snap-type of Proximity or Mandatory, you can implement snapping.
- Zooming is implemented by using the - ms-content-zooming: zoom property with a minimum and maximum zoom level.
- WinJS style sheets use the Segoe UI font family by default. You can add your own custom fonts by using CSS font-face.
- Text overflow can be controlled by using the win-type-ellipsis class.

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 an app for a news magazine, and the editors create a standard HTML page that contains all the content. The app should display the articles and mix them with appropriate advertisements. Which layout should you choose?
   - **A.** Flexible Box layout
   - **B.** Grid layout
   - **C.** Multi-column layout
   - **D.** Regions layout
2. You are using a Grid layout in which a user can dynamically move items from one cell to another. However, sometimes a user drops an item on a cell with an existing item, and the dropped item seems to disappear. What should you do?
   - **A.** Add a z-index property to the dropped item with a value larger than the original item.
   - **B.** Add a z-index property to the dropped item with a value smaller than the original item.
   - **C.** Nothing. You can't have multiple items in one cell in a Grid layout.
   - **D.** Add the column-span: 2 property to the target cell.

3. You are showing a grid with items on it to the user. Users can pan around the grid. Users should see only complete cells at any time. What should you do?
    - **A.** Add the -ms-scroll-snap-type property with a value of Proximity.
    - **B.** Add the -ms-scroll-snap-type property with a value of Mandatory.
    - **C.** Add the -ms-scroll-snap-type property with a value of None.
    - **D.** Add the -ms-scroll-snap-type property with a value of Automatic.

Objective 3.3: Create layout-aware apps to handle windowing modes

Although Windows Store apps are normally run in full screen, it isn't the only available windowing mode. Apps can also be viewed side by side to allow multitasking, so your app can run in different sizes that you need to respond to.

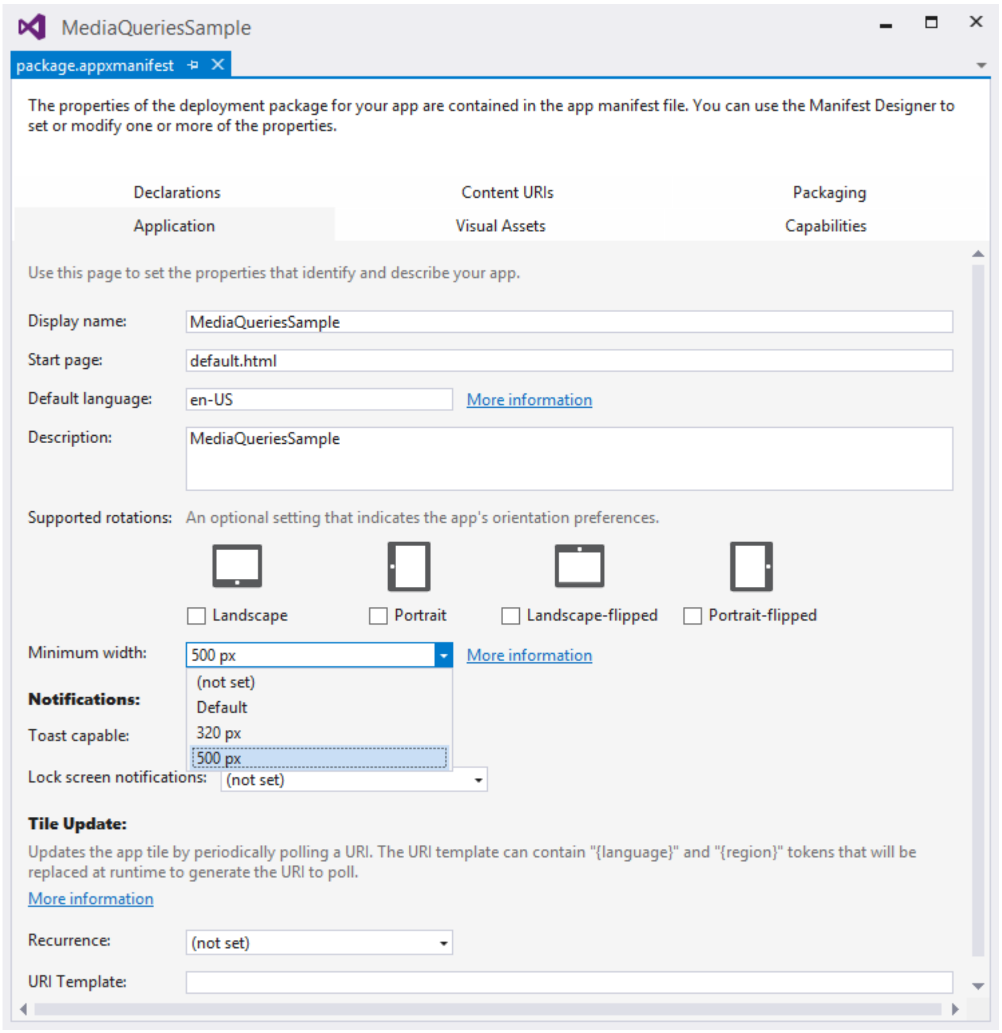
Deciding which widths you support and adapting to changes in windowing mode through CSS and code are the subjects of this objective.

This objective covers how to:

- Use CSS3 media queries to adapt to different devices
- Respond to changes in orientation
- Adapt to new windowing modes by using the ViewManagement namespace
- Manage settings for an apps view

Using CSS3 media queries to adapt to different devices

When you create your app, you have to work with several different screen sizes. The App Manifest Designer shown in Figure 3-13 shows a Minimum Width property that you can set.


MediaQueriesSample

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.

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

Display name: MediaQueriesSample

Start page: default.html

Default language: en-US More information

Description: MediaQueriesSample

Supported rotations: An optional setting that indicates the app's orientation preferences.

☐ Landscape ☐ Portrait ☐ Landscape-flipped ☐ Portrait-flipped

Minimum width: 500 px More information

(not set)
Default
320 px
500 px

Notifications:

Toast capable:

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 3-13 The App Manifest Designer shows the values for the Minimum Width property

By default, this value is not set, so your app can be used in all possible windowing modes. When you set this value to 320px, a user can use your app in all sizes that are larger than 320px. This means that a user running a display of 1024px wide can snap your app to one-third of the screen. When your app doesn’t have any real value to offer in such a small area, you should set the minimum width to 500px.

Testing your app in different resolutions can be difficult when you don't have a lot of devices to test on. Fortunately, you can use the simulator that you can launch from Visual Studio to test your app in different resolutions. Figure 3-14 shows the simulator running an app in snapped mode.

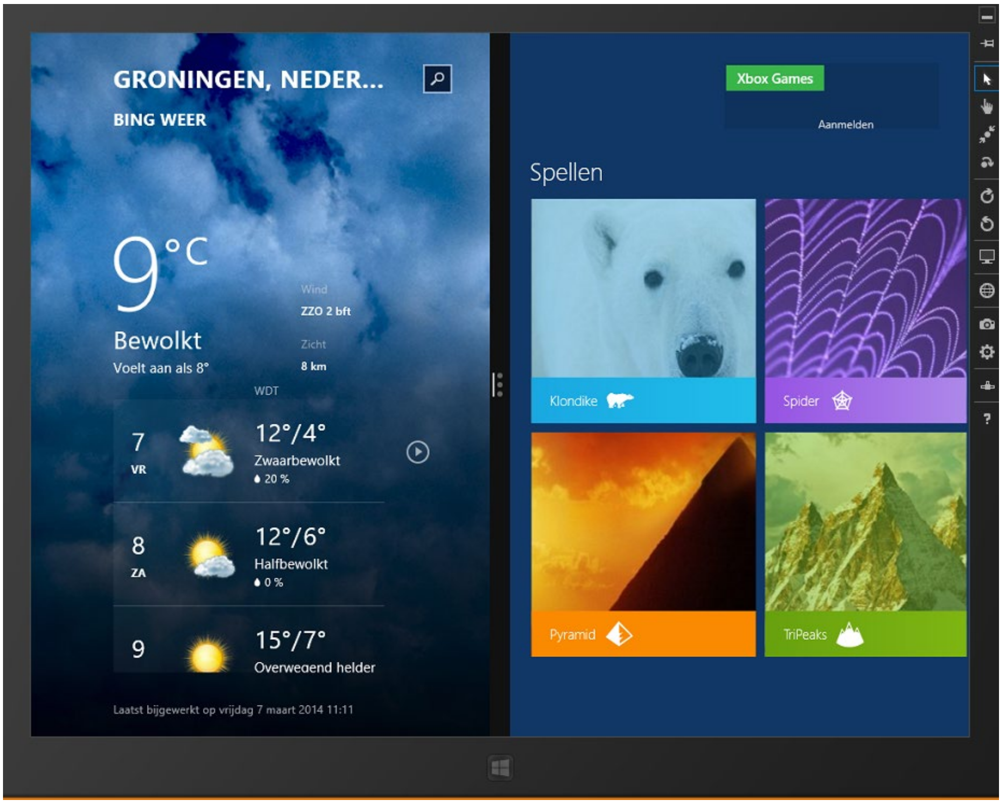


FIGURE 3-14 The simulator running two apps in snapped mode with a resolution of 1024 x 768

You can adapt your app by using CSS media queries. A *media query* is a special CSS selector in which you can check device properties. For example, some websites use CSS media queries to offer a specialized layout for printing.

CSS media queries allow you to query for the dimensions of your device. Look at the following CSS:

```
@media (max-width:499px) {
  …
}
```

All CSS inside the @media block apply only for a width that is smaller than 500px, so you can specify multiple sets of CSS rules that apply to different resolutions.

Windows Store apps always use the full height of your device; the width can vary. Your app should adapt to a width of at least 500px. If it makes sense for your app, you can also add a CSS media query for a width of 320px.

You can keep all the CSS for different resolutions in one file or split them in multiple files and load them conditionally like this:

```
<link href="/css/small.css" rel="stylesheet" media="(max-width: 320px)" />
```

When designing your app layout, you need to take into account the different snapped window modes that you will support. By using layout controls discussed in the previous objective, such as the Grid layout, you can create a very flexible layout.

When working with the Grid layout, items are positioned through CSS, not according to their location in the HTML document. By changing the CSS when a new media query applies, you can change the order and visibility of items.

EXAM TIP

Designing a flexible layout and adapting to different windowing modes is an important part of building apps (and of the exam). Microsoft released a walkthrough of creating a flexible layout at *http://msdn.microsoft.com/en-us/library/windows/apps/jj150600.aspx*. You should study this example before you take the exam.

Responding to changes in orientation

When they use tablet devices, users don't always use your app in landscape mode. Changing the orientation by rotating the screen is supported in Windows Store apps.

To test this behavior, you need a device with an actual sensor; using the simulator doesn't work. To check for changes in orientation, use a class called SimpleOrientationSensor. You get an instance of this class by calling Windows.Devices.Sensors.SimpleOrientationSensor.getDefault(). The object returned by this method call exposes the OrientationChanged event. By subscribing to this event, you can respond to changes in orientation.

```
var orientationSensor = Windows.Devices.Sensors.SimpleOrientationSensor.getDefault();
orientationSensor.addEventListener("orientationchanged", onOrientationChanged);
function onOrientationChanged(e) {
….
}
```

The argument that gets passed to your event handler receives an argument of type SimpleOrientationSensorOrientationChangedEventArg. This object has a property for the current orientation of type SimpleOrientation enumeration that can be one of the following values:

- notRotated
- rotated90DegreesCounterclockwise
- rotated180DegreesCounterclockwise
- rotated270DegreesCounterclockwise
- faceup
- facedown

Remember that your app might receive an orientationchanged event while it is not visible. Responses to those events might change things unexpectedly for users who return to your app.

Instead of blindly subscribing to the orientationchanged event, you can add and remove your event handler, depending on the visibility of your app, like this:

```
function visibilityChangeHandler() {
    if (document.getElementById("scenario1Open").disabled) {
        if (document.msVisibilityState === "visible") {
            sensor.addEventListener("orientationchanged", onDataChanged);
        } else {
            sensor.removeEventListener("orientationchanged", onDataChanged);
        }
    }
}
```

You can attach this event handler to the visibilitychange event on your document:

```
document.addEventListener("visibilitychange", visibilityChangeHandler, false);
```

Instead of waiting for a change in orientation to happen so you can handle the event, you can also ask the SimpleOrientationSensor for the current orientation by calling sensor.getCurrentOrientation().

There is another way to respond to changes in orientation. Just as you can use media queries for handling changes in the width of your window, you can also use media queries for changes in orientation:

```
@media screen and (orientation: landscape) and (max-width: 1024px) {
}

@media screen and (orientation: portrait) and (min-width: 500px) {
}
```

These two queries respond both to changes in the width and the orientation. You can also create a query that responds only to orientation changes.

Adapting to new windowing modes by using the ViewManagement namespace

The Windows.UI.ViewManagement namespace gives you access to classes that can help you respond to changes in windowing modes. These changes might occur because your app gets snapped or because the user loads the on-screen keyboard that hides the bottom part of your app.

You can get information for the current view by calling Windows.UI.ViewManagement. ApplicationView.getForCurrentView(), which gives you an object of type ApplicationView that you can then query for details on the current view.

This class exposes the following properties:

- **AdjacentToLeftDisplayEdge** Indicates whether the current window (app view) is snapped and adjacent to the left edge of the screen
- **AdjacentToRightDisplayEdge** Indicates whether the current window (app view) is adjacent to the right edge of the screen
- **Id** Gets the ID of the window (app view)
- **IsFullScreen** Indicates whether the window (app view) fills the entire screen
- **IsOnLockScreen** Indicates whether the window (app view) is on the Windows lock screen
- **IsScreenCaptureEnabled** Gets or sets whether screen capture is enabled for the window (app view)
- **Orientation** Gets the current orientation (landscape or portrait) of the window (app view) with respect to the display
- **TerminateAppOnFinalViewClose** Indicates whether the app terminates when the last window is closed
- **Title** Gets or sets the displayed title of the window
- **Value** Indicates the state of the current window (app view)

The orientation property can return a value of landscape or portrait. The isFullScreen value returns false when the app is snapped to the left or right part of the screen. By checking adjacentToLeftDisplayEdge or adjacentToRightDisplayEdge, you can see whether the app is on the left or right side of the user's screen. That way, you can change the layout of your page (by aligning elements at the right or left side of your app, for example).

Windows.UI.ViewManagement also offers you access to the InputPane class, which raises notifications when the on-screen input is shown and when it hides:

```
var inputPane = Windows.UI.ViewManagement.InputPane.getForCurrentView();
inputPane.addEventListener("hiding", function () {
});
inputPane.addEventListener("showing", function () {
});
```

Managing settings for an apps view

The Windows.Ui.ViewManagement namespace provides classes and enumerations for managing settings of an application view. As discussed in the previous section, some classes in ViewManagement allow you to get information on the user's configuration; others allow you to change settings.

Windows are defined as app views, and an app view is the displayed portion of a Windows Store app. A user can have up to four different app views displayed on the screen. Those views fill the screen from top to bottom and are distributed from left to right without overlap.

Knowing whether your app is running full screen or snapped to the left or right side of the screen can be important for deciding what functionality or data you want to show on the screen. You get data by calling Windows.UI.ViewManagement.ApplicationView.getForCurrentView().

You can query your current app view for its location like this:

```
var currentAppView = Windows.UI.ViewManagement.ApplicationView.getForCurrentView();
var isFullScreen = currentAppView.isFullScreen;
var isAdjacentToLeftDisplayEdge = currentAppView.adjacentToLeftDisplayEdge;
var isAdjacentToRightDisplayEdge = currentAppView.adjacentToRightDisplayEdge;
```

By reading the orientation property, you can see whether your app is running in portrait or landscape mode:

```
   var winOrientation = Windows.UI.ViewManagement.ApplicationView.getForCurrentView().
orientation;
            if (winOrientation === Windows.UI.ViewManagement.ApplicationViewOrientation.
landscape) {
            } else if (winOrientation === Windows.UI.ViewManagement.
ApplicationViewOrientation.portrait) {
   }
```

You can get information on accessibility settings:

```
var accessibilitySettings = new Windows.UI.ViewManagement.AccessibilitySettings();
output.innerHTML = "High Contrast: " + accessibilitySettings.highContrast + "<br />" +
    "High Contrast Scheme: " + accessibilitySettings.highContrastScheme;
```

This code retrieves the current accessibility settings a user has defined for the whole system. The highContrast variable returns true if the user has enabled high contrast. The scheme property returns the name of the color scheme that is being used.

Another important capability that ViewManagement offers is projection. A typical example of projection is PowerPoint—when you present, you can configure PowerPoint to show your slides on one display while showing your presentation notes on another display.

Windows Store apps support projection with ProjectionManager. You can use it to see whether a second display (a secondary monitor or a projection screen, for example) is available:

```
ViewManagement.ProjectionManager.projectionDisplayAvailable
```

When projecting to another screen, you first need some content to display. The easiest way to do get it is to load an HTML file that is part of your package. This HTML file can reference CSS and JavaScript files that will be loaded with it:

```
    var view = MSApp.createNewView("ms-appx:///html/secondaryView.html");
```

Now that you have a reference to an app view, you can start projecting it:

```
ViewManagement.ProjectionManager.startProjectingAsync(
            view.viewId,
            ViewManagement.ApplicationView.getForCurrentView().id
        );
```

You can also stop the projection:

```
ViewManagement.ProjectionManager.stopProjectingAsync(
            view.viewId,
            ViewManagement.ApplicationView.getForCurrentView().id
        );
```

When projecting to another display, you have to offer the user the option to swap the primary and secondary display. You can do this by calling swapDisplaysForViewsAsync:

```
ViewManagement.ProjectionManager.swapDisplaysForViewsAsync(

    ViewManagement.ApplicationView.getForCurrentView().id,

    MSApp.getViewOpener().viewId);
```

The first parameter is the ID of the projected app view; the second parameter points to the view that opened the current app view. By swapping them, you change the projected and main screen.

MORE INFO **PROJECTION SAMPLE**

A complete example that shows how to start and stop projection and how to swap displays can be found at *http://code.msdn.microsoft.com/windowsapps/Projection-sample-526b3c1d*.

Thought experiment

Adapting to change

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 working on a ToDo app that uses a hub template with different categories and individual lists. Because you just learned about all the different windowing modes, you are thinking about what they could mean for your app.

1. **Which snapped modes do you want to support? What are you going to show?**
2. **Which technology should you use to change your UI?**

Objective summary

- CSS3 media queries can be used to adapt your app to different windowing modes when it gets snapped. You can use your app manifest to configure the sizes your app supports.
- You can respond to changes in orientation by using media queries or the SimpleOrientationSensor class.
- The ViewManagement namespace gives you access to the current ApplicationView object. This object has information about the orientation, size, and alignment of your app window.
- ViewManagement allows you to project a secondary view to a second display.

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 using media queries to adapt your app to different windowing layouts. Which dimensions can you support? (Choose all that apply.)
   - **A.** 200px
   - **B.** 320px
   - **C.** 500px
   - **D.** 1024px

2. You want to change the functionality of your app when a user changes the orientation of the device. How can you do this? (Choose all that apply.)
   - **A.** Use a media query with a selector of orientation.
   - **B.** Use the SimpleOrientationSensor class to be notified of changes in orientation.
   - **C.** Listen for the visibilitychange event.
   - **D.** Listen to the orientationchanged event on the ApplicationView object.
3. You want to make sure that your app can be used with a size of only 500px or larger. What should you do?
   - **A.** Add a media query that blacks out the screen when the app gets smaller.
   - **B.** Listen to the window resized event and cancel the event when the app gets too small.
   - **C.** Configure the app manifest for a minimum size of 500px.
   - **D.** Add media queries for only the supported window sizes.

Objective 3.4: Design and implement the app bar

Windows Store apps have some clear design guidelines regarding the layout and placement of your elements. One important control in your apps is the app bar, which is used for placing commands that are local to the current page or for your whole app. It helps users easily find the commands they are looking for.

This objective covers how to:

- Determine what to put on the app bar based on app requirements
- Style and position app bar items
- Design the placement of controls on the app bar
- Handle AppBar events

Determining what to put on the app bar based on app requirements

The design guidelines for Windows Store apps lead you to a clean, well-designed app without distractions of the main goal of your app. However, completely ignoring commands does not work.

Users need commands for navigation or for specific actions in your app. In Figure 3-15, the Weather app shows both the bottom and the top app bars.

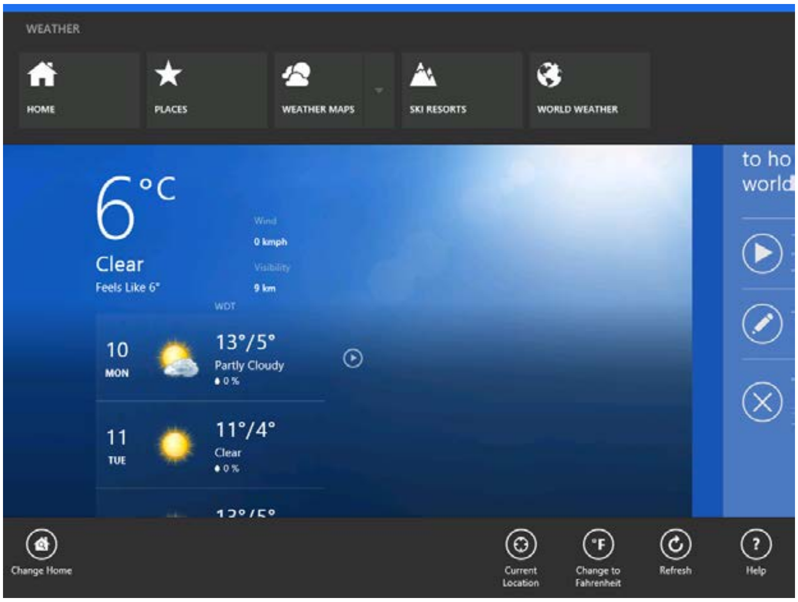


FIGURE 3-15 The app bars in the Weather app

The top bar is used for navigation among various sections of your app. You can use it to offer users quick access to the home screen and other parts of your app.

The bottom app bar contains commands that invoke actions in your app. The difference between commands on the left and right of the bottom app bar is about their applicability to the current page. Commands available in all pages are placed on the right; commands useful only in the current context are placed on the left.

This grouping helps users become familiar with your app. Users can expect global commands to always be in the same location at the right side of your app bar. When they look for specific commands, they have to look only at the left part of the app bar.

You can further help users by making sure the app bar automatically appears when a user selects an item that has associated actions. This happens on your Start screen when you select a tile: The app bar opens and shows commands to uninstall the app or configure its tile.

Microsoft guidelines state that you shouldn't put critical commands on the app bar (for example, the Take A Picture command in a camera app). This command should be added to the apps page so users can easily access it.

You also shouldn't put commands that are related to settings and configuration in your app bar. The function of letting users log on or change account or global app settings should be in the Settings charm.

By using a Menu flyout, you can place commands such as copy, paste, and cut in a menu that directly relates to selected text or items. Those commands also shouldn't be placed on the app bar.

Styling and positioning app bar items

You create an app bar by using the WinJS.UI.AppBar control. Add a div set to this control, and an empty app bar is then added to your app.

A simple app bar with one button can look like this:

```
<div id="myAppBar" data-win-control="WinJS.UI.AppBar">
    <button data-win-control="WinJS.UI.AppBarCommand" data-win-options=
       "{id:'cmdAdd',label:'Add',icon:'add',section:'global',tooltip:'Add item'}">
    </button>
</div>
```

This code creates an app bar with one button, as shown in Figure 3-16.

A lot of the styling is handled for you by the AppBar and AppBarCommand controls. Because you added a button with an Add icon, you see a nice round button with a + icon. Using these default icons gives a consistent interface that is immediately recognizable by a user.

You can add multiple types of commands to an app bar:

- Button
- Toggle
- Flyout
- Separator
- Content

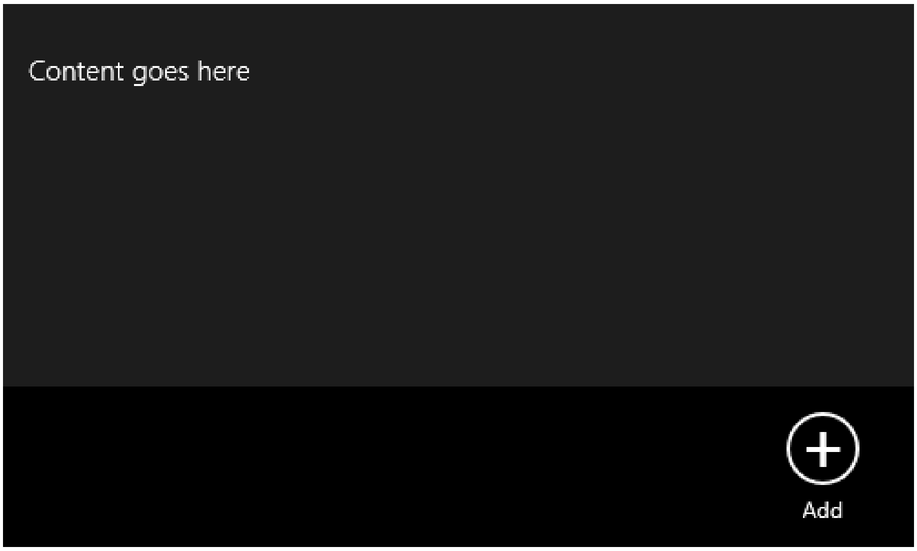


FIGURE 3-16 The simple app bar with an Add button

A button exists of a label, a tooltip, and an icon. There is a long list of available icons for styling buttons (see the following note). You can also add your own icons by using a 40px x 40px PNG file with a transparent background.

MORE INFO **AVAILABLE ICONS FOR APP BAR COMMANDS**

The WinJS library defines a long list of icons that you can use in your app. You can find this list at *http://msdn.microsoft.com/en-us/library/windows/apps/hh770557.aspx*.

To add a toggle, use the following markup:

```
<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id:'cmdToggle',
    type:'toggle',
    label:'Toggle',
    icon:'audio',
    section:'global',
    tooltip:'Toggle'}"></button>
```

This is also an HTML button, but now you use a type of toggle to create a button that a user can press and release to toggle some action.

You can also add a flyout to a button on the app bar. First, define the flyout in markup outside of the app bar:

```
<div id="exampleFlyout" data-win-control="WinJS.UI.Flyout" aria-label="{Example
flyout}">
    <div>This is an example AppBarCommand of type 'flyout'.</div>
    <button id="exampleFlyoutButton">Example flyout</button>
</div>
```

Add a button of type flyout to your app bar and link it to your flyout:

```
<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id:'flyoutButton',
    type:'flyout',
    label:'Flyout',
    icon:'openpane',
    section: 'global',
    flyout:select('#exampleFlyout')}"></button>
```

By using the content app bar command, you can add a div that you can fill with custom content like this:

```
<div data-win-control="WinJS.UI.AppBarCommand"
     data-win-options="{id: 'textfield',
 type: 'content',
 label:'Text field',
 section: 'selection' }">
    <input type="text" value="Text" />
</div>
```

This code adds a text box to your app bar. You can add all HTML elements that you want, but remember that an app bar is not meant for complicated commands. If you want to add custom content to your app bar, consider your design and make sure it is really what you want.

Finally, the separator is used to group items together and visually separate those groups. To add a separator, use an HTML hr element:

```
<hr data-win-control="WinJS.UI.AppBarCommand"
    data-win-options="{id:'separator',
type:'separator',
section:'global'}" />
```

By combining these different command types, you have many app bar choices for users.

Designing the placement of controls on the app bar

As discussed in the previous section, commands can be placed and positioned on the left or right side of an app bar by using the section property on a command.

Section can have a default value of global or selection. Global, which positions commands on the right side of the app bar, is meant for commands that always show on all pages in an app. The following code shows a toggle that is placed in the global section of the app bar:

```
<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id:'cmdToggle',
    type:'toggle',
    label:'Toggle',
    icon:'audio',
    section:'global',
    tooltip:'Toggle'}"></button>
```

Selection is meant for commands that are placed on the left side of your app bar. Those commands have specific meaning in the current user's context. For example, the following code shows how to place a text field on the selection part of the app bar:

```
<div data-win-control="WinJS.UI.AppBarCommand"
     data-win-options="{ id: 'textfield',
 type: 'content',
 label:'Text field',
 section: 'selection' }">
    <input type="text" value="Text" />
</div>
```

In addition to selection and global, you can also use a custom layout, which allows you to use your own HTML to create the layout for your app bar. You can add HTML and WinJS controls to a custom section of your app bar. You create such a section by using content like this:

```
<div data-win-control="WinJS.UI.AppBarCommand" data-win-options="{ type: 'content' }">
    <p>
        You can include a wide variety of HTML inside of a 'content' AppBarCommand, <br
/>
        including HTML and some WinJS controls.
    </p>
</div>
```

EXAM TIP

When you are asked on the exam to position items on the app bar, make sure that you read the requirements carefully. Look for information that suggests whether the command should be globally available or is linked to items on the current page.

Handling AppBar events

You can easily attach handlers for the commands you place on your app bar. You can get an instance of the app bar by using the winControl property. By calling getCommandById on your app bar, you can then get access to the individual commands and attach event handlers:

```
var appBar = document.getElementById("createAppBar").winControl;
appBar.getCommandById("cmdAdd").addEventListener("click", doClickAdd, false);
```

The app bar also raises events. You can listen to one of these events:

- onafterhide occurs immediately after the AppBar is hidden.
- onaftershow occurs after the AppBar is shown.
- onbeforehide occurs before the AppBar is hidden.
- onbeforeshow occurs before a hidden AppBar is shown.

The following code sample shows how to attach to these events:

```
var appBar = document.getElementById('myAppBar').winControl;
appBar.addEventListener('afterhide', function () {
    WinJS.log
        && WinJS.log("after hide");
});

appBar.addEventListener('aftershow', function () {
    WinJS.log
        && WinJS.log("after show");
});
```

```
appBar.addEventListener('beforehide', function () {
    WinJS.log
        && WinJS.log("before hide");
});

appBar.addEventListener('beforeshow', function () {
    WinJS.log
        && WinJS.log("before show");
});
```

You can control app bar showing and hiding programmatically. You can show the app bar by calling this:

```
document.getElementById('idofappbar').winControl.show();
```

By using the sticky property, you can control whether the app bar is automatically closed whenever the user clicks outside of it:

```
var appBar = document.getElementById('myAppBar').winControl;
appBar.sticky = true;
appBar.show();
```

Thought experiment

Adding some commands

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.

Open the Weather app on your Windows 8 device. Navigate through the app and open the app bar for each page.

1. **What's the difference between the commands on the left side and the commands on the right side of your app bar?**
2. **Is the app consistent on every page?**

Objective summary

- The app bar is used in Windows Store apps to provide a common location for placing commands. Commands can be both global or for the current context.
- You can use a button, flyout, content, separator, or toggle command. You can use predefined styles and icons when creating commands.
- Contextual commands should be placed on the left side of your app bar; global commands should be on the right. By using a separator, you can group commands. You should avoid overloading the app bar with commands.

- You can attach event handlers for individual commands on the app bar and to the app bar itself for when it is shown and hidden.

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 a ToDo App. When the user selects a specific item, you want to show a command on the app bar for completing the item. Where should you position this command?
   - **A.** On the left side of the app bar
   - **B.** On the right side of the app bar
   - **C.** In the middle of the app bar
   - **D.** In a custom section of the app bar
2. You want to add a flyout to your app bar. What should you do? (Choose all that apply.)
   - **A.** Create a button with a data-win-control of WinJS.UI.AppBarCommand.
   - **B.** Add a div with a data-win-control of WinJS.UI.Flyout.
   - **C.** Create a div with a data-win-control of WinJS.UI.AppBar.
   - **D.** Add a div with a data-win-control of WinJS.UI.AppBarCommand.
3. You want to make sure that the app bar stays visible when a user selects another item on your page. What should you do?
   - **A.** Listen for the beforehide event and cancel the event in the event handler.
   - **B.** Set the sticky property of your app bar to true.
   - **C.** Listen for the afterhide event and cancel the event in the event handler.
   - **D.** Set the sticky property of your app bar to false.

Objective 3.5: Apply CSS styling

Windows Store apps written with HTML, CSS, and JavaScript run in Internet Explorer 11, which enables you to use all CSS techniques that Internet Explorer 11 supports. You can style your apps just as you would with a regular website.

Most of the content of this objective has been discussed in other parts of the book. In this objective, you learn about gradients in detail; other content is a quick recap and a reference to other sections in the book.

This objective covers how to:

- Implement gradients
- Implement Grid layouts
- Implement zooming
- Implement scroll snapping
- Implement media queries

Implementing gradients

CSS3 gradients let you display a smooth transition from one color to another without having to specify all the details about the colors in-between. The gradient, which is generated by Internet Explorer 11, creates a very smooth transition from the first color to the second.

You can use two types of gradients:

- Linear
- Radial

A linear gradient goes from one point to another: left/right, up/down, or diagonally. A radial gradient is defined by its center and expands in a circular path.

To create a linear gradient, you have to define a minimum of two colors:

```
#grad {
    background: linear-gradient(red, blue);
    width: 100px;
    height: 100px;
}
```

This code results in the div shown in Figure 3-17.

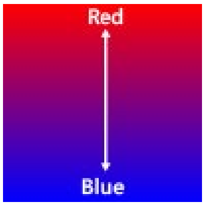


FIGURE 3-17 The div with a linear gradient from red to blue

You can change the direction of your linear gradient by adding a direction property like this:

```
background: linear-gradient(to bottom right, red , blue);
```

You can use any of the predefined values such as to bottom, to top, to right, to left, to bottom right, and so on.

If those predefined directions are not what you're looking for, you can specify an angle. The angle is specified as an angle between a horizontal line and the gradient line, going counterclockwise. In other words, 0deg creates a bottom-to-top gradient, whereas 90deg generates a left-to-right gradient:

```
background: linear-gradient(180deg, red, blue);
```

You can use as many colors as you want:

```
background: linear-gradient(red, green, blue);
```

Instead of using predefined colors such as red, green, or white, you can also use the rgba function to specify the exact components of all colors. This function also allows you to specify a value from 0 (fully transparent) to 1 (completely solid):

```
background: linear-gradient(to right, rgba(255,0,0,0), rgba(255,0,0,1));
```

A radial gradient is created by using the following syntax:

```
#grad {
    background: radial-gradient(red, green);
    width: 100px;
    height: 100px;
}
```

This code creates the gradient shown in Figure 3-18.

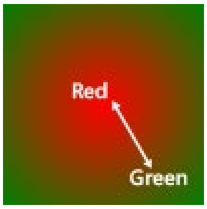


FIGURE 3-18 The div with a radial gradient from red to green

This gradient is an ellipse and has its center in the middle of the containing div. You can set the shape to a circle:

```
background: radial-gradient(circle, red, yellow, green);
```

Implementing Grid layouts

Grid layouts are discussed in Objective 3.2. A CSS3–based Grid layout uses rows and columns that define cells. You can place items in those cells or let them span multiple cells.

The following code shows a complete example:

```
<div id="myGrid">
    <div id="item1">Item 1</div>
    <div id="item2">Item 2</div>
    <div id="item3">Item 3</div>
    <div id="item4">Item 4</div>
    <div id="item5">Item 5</div>
</div>
#myGrid {
    display: -ms-grid;
    background: gray;
    border: blue;
    -ms-grid-columns: auto 1fr 2fr;
    -ms-grid-rows: 200px auto;
}
    #myGrid div {
        background: lightgray;
        border: red solid 1px;
    }

#item1 {
    background: maroon;
    -ms-grid-row: 1;
    -ms-grid-column: 1;
}

#item2 {
    -ms-grid-row: 1;
    -ms-grid-column: 2;
}

#item3 {
    -ms-grid-row: 1;
    -ms-grid-column: 3;
}

#item4 {
    -ms-grid-row: 2;
    -ms-grid-column: 1;
}
```

```
#item5 {
    -ms-grid-row: 2;
    -ms-grid-column: 2;
    -ms-grid-column-span: 2;
}
```

Implementing zooming

Zooming, which is discussed in Objective 3.2, is performed by using the -ms-content-zooming: zoom CSS property like this:

```
.zoomElement {
    overflow: auto;
    -ms-content-zooming: zoom;
    -ms-scroll-rails: none;
    -ms-content-zoom-limit-min: 100%;
    -ms-content-zoom-limit-max: 500%;
```

Specify a minimum and maximum zoom limit, and make sure that the content is set to overflow so it stays within its container.

Implementing scroll snapping

Scroll snapping is important when implementing scrolling and panning (it is discussed in detail in Objective 3.2).

By using the -ms-scroll-snap-type property with a value of proximity or mandatory, you can force content to snap to a specific position when a user is panning or scrolling through it.

An example of a mandatory snapping is shown here:

```
.MandatorySnapInterval {
    -ms-scroll-snap-type: mandatory;
    -ms-scroll-snap-points-x: snapInterval(0%, 100%);
}
```

Proximity snapping can be done with the following code:

```
.ProximitySnapList {
    -ms-scroll-snap-type: proximity;
    -ms-scroll-snap-points-x: snapList(100%, 200%, 300%, 400%, 500%);
}
```

Implementing media queries

Using CSS media queries to change your CSS for different screen sizes is discussed in Objective 3.3. You use a media query like this:

```
@media (max-width:499px) {
 …
}
```

The CSS in this block applies only when your app is less than 500px wide. You can use it to create different CSS settings when your app is snapped or when it is on devices with a low resolution.

EXAM TIP

Media queries are very important for creating apps that run on a variety of devices. Make sure that you understand how they work and that you have some practical experience.

Thought experiment

Using CSS

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.

All effects mentioned in this objective use CSS. However, you can also create those effects by using JavaScript and HTML.

1. **Explain why you want to use CSS for those effects.**
2. **Put the CSS techniques in this objective in order, from most likely to use in your app to less likely to use in your app.**

Objective summary

- A gradient can be linear or radial. By defining multiple colors, the browser creates a smooth transition from one color to the next.
- Grid layouts are perfect for creating flexible layouts that can easily adapt to different sizes. You define a grid with rows and columns and then place items into cells.
- Zooming can be done by using the -ms-content-zooming: zoom; property with a minimum and maximum zoom level.

- Scroll snapping allows you to restrict a user to scrolling a specific amount at a time (mandatory snapping) or by snapping to a certain position when a user gets close (proximity snapping).
- Media queries are used to respond to changes in screen size and apply different CSS rules, depending on the dimensions.

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 want to create a gradient from top to bottom that goes from transparent white to orange to red. Which syntax should you use?
   - **A.** background: linear-gradient(rgba(0,0,0,0), rgba(255, 165, 0,0.5),red);
   - **B.** background: linear-gradient(to top, rgba(0,0,0,0), rgba(255, 165, 0,0.5),red);
   - **C.** background: linear-gradient(to bottom, white, orange, red);
   - **D.** background: linear-gradient(to bottom, rgba(0,0,0,0), rgba(255, 165, 0,0.5),red);
2. You have Grid layout with four columns and five rows. The header should be placed horizontally from left to right across all columns. Which property should you use?
   - **A.** -ms-grid-columns: auto;
   - **B.** -ms-grid-column-span: 4;
   - **C.** -ms-grid-row-span: 5;
   - **D.** z-index: 4;
3. You want to enable zooming on an image you have on your page. What should you do? (Choose all that apply.)
   - **A.** Set the image in a container div.
   - **B.** Apply the -ms-content-zooming: zoom property to the div.
   - **C.** Apply the -ms-content-zooming: zoom property to the image.
   - **D.** Configure the -ms-content-zoom-limit-min and -ms-content-zoom-limit-max properties.

Answers

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

Objective 3.1: Thought experiment

The Weather app uses a Repeater control, a ListView control, and flyouts, as well as the more basic controls such as images, buttons, and text elements.

Objective 3.1: Review

1. **Correct answer:** D
   - **A. Incorrect:** An itemTemplate property can't be used for interactive items.
   - **B. Incorrect:** The type of data source doesn't influence the way a user interacts with the items.
   - **C. Incorrect:** This event doesn't exist.
   - **D. Correct:** With a templating function, you can use JavaScript to add additional event handlers.
2. **Correct answers:** B, C, D
   - **A. Incorrect:** The List object has a method for creating groups for the data it has.
   - **B. Correct:** This is used to determine the group to which an item belongs.
   - **C. Correct:** This is used to order items.
   - **D. Correct:** This is used to create data such as the group title that can then be used when rendering the groups.
3. **Correct answer:** D
   - **A. Incorrect:** This interface is used when creating a WinRT object for streaming data to a WebView control.
   - **B. Incorrect:** This is used to send data from the webpage in the WebView control to the app.
   - **C. Incorrect:** This is the event that is raised when a webpage sends data to the app.
   - **D. Correct:** This method is used to invoke a script on the webpage loaded in the WebView control.

Objective 3.2: Thought experiment

Windows Store apps can run on a multitude of devices. You can easily create a flexible layout by using a Grid, FlexBox, Multi-column, or Regions layout.

You can use templates and bindings to show data about restaurants and other activities the employee has to do.

Images can be easily incorporated in the app; with CSS3, you can implement image scrolling and zooming.

Objective 3.2: Review

1. **Correct answer:** D
   - **A. Incorrect:** Mixing regular content and an HTML document is most easily done by using a Regions layout. FlexBox is used for layouts in which the relative size of items is important.
   - **B. Incorrect:** Although a Grid layout is very flexible, it would require a lot of manual work to mix the HTML and custom content.
   - **C. Incorrect:** A Multi-column layout is used to flow content automatically to multiple columns and configure page and item breaks. It doesn't mix content from different sources.
   - **D. Correct:** Mixing of regular content and an HTML document is most easily done by using a Regions layout.
2. **Correct answer:** A
   - **A. Correct:** This places the newly dropped item on top of the other item.
   - **B. Incorrect:** A smaller value doesn't work because it would place the item below the already existing item.
   - **C. Incorrect:** You can have multiple items in one cell and use the z-index to arrange them.
   - **D. Incorrect:** This property should be added to an item to let it span multiple cells. In this case, items fit into one cell.
3. **Correct answer:** B
   - **A. Incorrect:** Proximity is used for snapping to a position when a user gets close to it. It doesn't force the user to pan one item at a time.
   - **B. Correct:** Mandatory allows you to make sure that a user can pan one item at a time.
   - **C. Incorrect:** A value of None allows the user to pan around freely.
   - **D. Incorrect:** A value of Automatic does not exist.

Objective 3.3: Thought experiment

1. You can support both 320px- and 500px-wide screens. In the 320px mode, you can view only the categories in a list view or a single ToDo list.
2. Media queries are very well suited for this.

Objective 3.3: Review

1. **Correct answers:** B, C
   - A. **Incorrect:** The smallest size is 320px.
   - B. **Correct:** 320px is a supported size for apps.
   - C. **Correct:** 500px is a supported size for apps.
   - D. **Incorrect:** 1024px isn't a size you have to explicitly support. Everything larger than 500px can be used.
2. **Correct answers:** B, C
   - A. **Incorrect:** With a media query, you can change only CSS rules; you can't change functionality.
   - B. **Correct:** This allows you to respond to orientation changes in JavaScript.
   - C. **Correct:** You have to use this event to make sure that nothing is changing when your app is not visible.
   - D. **Incorrect:** This event doesn't exist.
3. **Correct answer:** C
   - A. **Incorrect:** Although possible, it is definitely not an optimal solution. By configuring the manifest, an app closes when it gets smaller than 500px.
   - B. **Incorrect:** This event can't be canceled.
   - C. **Correct:** This makes sure that a user can't resize your app to less than 500px.
   - D. **Incorrect:** Adding media queries for specific sizes doesn't limit the app to those sizes; it limits only the sizes that CSS rules apply to.

Objective 3.4: Thought experiment

1. Commands on the left side are specific to the page you are on; commands on the right side are global.
2. Not completely. Some pages have different global commands than other pages.

Objective 3.4: Review

1. **Correct answer:** A
    - **A.** **Correct:** Commands on the left side are for contextual commands.
    - **B.** **Incorrect:** Commands on the right side are for global commands. The commands available when an item is selected are not globally available.
    - **C.** **Incorrect:** You shouldn't align items to the middle of the app bar.
    - **D.** **Incorrect:** A command associated with a selected item should be on the left side. Using a completely custom layout is not required for this scenario.
2. **Correct answers:** A, B, C
    - **A.** **Correct:** This button activates the flyout.
    - **B.** **Correct:** The HTML for the flyout should be in a div on the page.
    - **C.** **Correct:** The app bar is a div with a WinJS.UI.AppBar WinJS control.
    - **D.** **Incorrect:** A flyout should be opened from a button.
3. **Correct answer:** B
    - **A.** **Incorrect:** You can't cancel this event.
    - **B.** **Correct:** Setting sticky to true ensures that the app bar isn't easily dismissed.
    - **C.** **Incorrect:** You can't cancel this event.
    - **D.** **Incorrect:** A value of false for sticky is the default. You should set it to true for this scenario.

Objective 3.5: Thought experiment

1. Browsers are optimized for using CSS. They can run native code when applying CSS rules and even use hardware acceleration.
2. Grid layout, media queries, scroll snapping, zooming, gradients.

Objective 3.5: Review

1. **Correct answer:** D
    - **A.** **Incorrect:** This line misses the correct direction setting.
    - **B.** **Incorrect:** The direction should be from top to bottom, not bottom to top.
    - **C.** **Incorrect:** This misses the alpha settings that require using the rgba function.
    - **D.** **Correct:** This has both the correct direction and alpha settings.

2. **Correct answer:** B
    - **A. Incorrect:** This creates a grid with one column that is autosized.
    - **B. Correct:** Column span lets one item span across multiple columns.
    - **C. Incorrect:** You need a column, not a row span.
    - **D. Incorrect:** A z-index is used to place items on top of each other.
3. **Correct answers:** A, B, D
    - **A. Correct:** An element needs a container to zoom in on it.
    - **B. Correct:** The div should have the zoom property.
    - **C. Incorrect:** The div, not the image, should have the zoom property.
    - **D. Correct:** A minimum and maximum zoom limit should be specified.

This page intentionally left blank

CHAPTER 4

Program user interaction

User participation is the most important part of your app. Your app can be built on a great idea with a beautiful user interface, but it won't become the next big hit if it can't be easily used by users.

This chapter is all about user interaction. You start by learning how to allow users to use touch and a keyboard and mouse to interact with your app. By using gestures and supporting different input styles, you can ensure that users feel comfortable when using your app.

The chapter discusses using navigation in an app, including how to create an app that makes it easy to find what users need and how to create a page structure that can be easily maintained and extended.

Finally, you learn how to get users to interact with your app when they are not directly using it. Using a tile that shows regular updates and invites users to launch your app and using toast notifications that keep users informed of changes in your app help to ensure that they come back.

This chapter focuses on 20 percent of the exam. Make sure that you understand how to use touch, how to create pages for your navigation, and how to use tiles and toasts to inform users of updates. Try to experiment with the ideas from this chapter and the Microsoft samples to make sure you understand these subjects.

Objectives in this chapter:

- Objective 4.1: Manage input devices
- Objective 4.2: Design and implement navigation in an app
- Objective 4.3: Create and manage tiles
- Objective 4.4: Notify users by using toast

Objective 4.1: Manage input devices

A Windows Store app offers you a unique possibility to create an app to run on a diverse set of devices. Users can use touch, a mouse and keyboard, and a stylus to work with your app.

When users start using multiple apps, they begin to expect certain behavior. Windows has built-in gestures that they can use, and your app needs to support them. In this objective, you see the types of input your app can receive and how you can support them.

This objective covers how to:

- Capture gesture library events
- Create custom gesture recognizers
- Listen to mouse events or touch gestures
- Manage stylus input and inking
- Handle drag-and-drop events

Capturing gesture library events

A Windows Store app runs on devices with a mouse and keyboard, touch devices, and devices that use a stylus. All these input styles are well supported in Windows Store apps. It is your job to ensure that your app uses those input devices to reach the largest audience possible.

These actions, which are called gestures, can be performed by a finger, fingers, pen/stylus, mouse, and so on. For example, invoking a command with a single finger touch is equivalent to clicking it with the mouse or a stylus, or selecting it and pressing Enter on a keyboard.

In a Windows Store app, all forms of input are referred to as pointer input. By unifying input mechanisms, you can create apps to handle all types of input. Pointers contain extra data based on the device that created it, such as the mouse button that created the click event or the number of fingers that created a swipe.

On a higher level, Windows offers you gestures that can arise from any input source. By using gestures, you don't have to focus on raw pointer data.

The basic gestures that Windows 8 relies on are shown in Table 4-1.

TABLE 4-1 Gestures in Windows 8

Name	Type	Description
Tap	Static gesture	One finger touches the screen and lifts up.
Press and hold	Static gesture	One finger touches the screen and stays in place.
Slide	Manipulation gesture	One or more fingers touch the screen and move in the same direction.
Swipe	Manipulation gesture	One or more fingers touch the screen and move a short distance in the same direction.
Turn	Manipulation gesture	Two or more fingers touch the screen and move in a clockwise or counterclockwise arc.
Pinch	Manipulation gesture	Two or more fingers touch the screen and move closer together.

Fortunately, WinJS controls already implement support for mouse, keyboard, touch, and stylus input. When using those controls, you automatically benefit from the work done by Microsoft to support all types of users.

For example, the semantic zoom control (discussed in the next objective) supports different kinds of inputs to allow zooming and panning through its content. The control processes pointer and gesture data so you can use it without thinking about those events.

If you want more control, start using the gesture events, which include MSGestureTap and MSGestureHold; and sequences of gestures such as MSGestureStart, MSGestureChange, and MSGestureEnd.

At the lowest level is the pointer data, which includes unified pointer events such as pointerdown, pointermove, and so on. You can parse user input directly and respond differently to different types of input.

Subscribing to gesture events is easy. The following example uses two divs—the first div can be manipulated, and the second one is for logging:

```
<div id="targetContainer" style="width: 100px; height: 100px; border: 1px solid blue"">
</div>
<div id="log" style="overflow:auto;width:100px;height:100px;border:1px solid red;">
</div>
```

In JavaScript, you can listen for the gesture events with the following code (you can register the event handlers in the onactivated event handler):

```
var divElement = document.getElementById("targetContainer");
divElement.addEventListener("MSGestureTap", function () {
    log("tap");
});
divElement.addEventListener("MSGestureHold", function (e) {
    log("hold");
});
divElement.addEventListener("MSGestureStart", function () {
    log("start");
});
divElement.addEventListener("MSGestureChange", function (e) {
    log("change");
});
divElement.addEventListener("MSGestureEnd", function () {
    log("end");
});
divElement.addEventListener("MSInertiaStart", function () {
    log("start");
});
```

The log method logs the message to the div:

```
function log(message) {
    var log = document.getElementById("log");
    log.innerHTML += message + "<br />";
}
```

The last step is to hook up the pointer events to an MSGesture object:

```
var gestureObject = new MSGesture();
gestureObject.target = divElement;
divElement.gestureObject = gestureObject;
divElement.addEventListener("pointerdown", function (e) {
    e.target.gestureObject.addPointer(e.pointerId);
}, true);
```

When you run this application, you see that tapping, holding, and moving with a mouse or your finger fires gesture events that are logged to the div.

Of course, just logging the events isn't very interesting. If you use the following code for the gesture change event, you can move the div around:

```
divElement.addEventListener("MSGestureChange", function (e) {
    var elt = e.target;
    var m = new MSCSSMatrix(elt.style.msTransform);

    elt.style.msTransform = m.translate(e.translationX, e.translationY);
});
```

The event argument that gets passed to the event handler contains a lot of data, such as translation, velocity, rotation, time, and scale. All those values are used to determine the type of gesture a user is making. Touching the screen, moving within a certain time threshold, and releasing is a *tap*. Touching the screen, moving beyond the time threshold, and releasing is a *hold*.

In this example, the movement is created by using an MSCSSMatrix object. Matrices are mathematical objects used for describing translations, rotations, and scaling. By creating more-complex matrices, you can create complex animations with gestures.

Why have you used regular JavaScript events such as click until now? The run time automatically translates pointer events into plain HTML events, which affects performance in a minor way. So it is best to use the pointer and gesture events when developing custom controls with specific gestures.

EXAM TIP

Make sure that you understand why you can use regular HTML events in Windows Store apps by translating pointer events into HTML events. However, when implementing performance-sensitive code, avoid using the HTML events by processing the pointer events yourself.

Testing gestures is the easiest when done on a touch-enabled device. However, for quick testing or when you don't have a touch device you can also use the simulator that's part of Visual Studio. On the right side of the simulator are multiple buttons (see Figure 4-1). The first few buttons can be used for simulating touch, pinching, and rotation. Other buttons can be used to change the screen resolution, location data, and network capabilities.

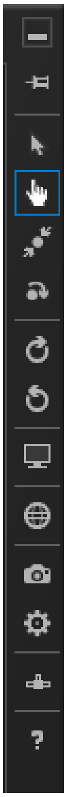


FIGURE 4-1 The buttons for the Visual Studio simulator running a Windows Store app

By using the simulator, you can easily test applications on a regular device. You launch the simulator from Visual Studio. To run your Windows Store app in the simulator, select Simulator from the list next to the Start Debugging button on the debugger toolbar (see Figure 4-2).

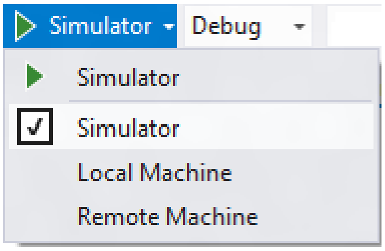


FIGURE 4-2 Selecting the Visual Studio simulator to run a Windows Store app

Creating custom gesture recognizers

Gestures such as swipe, rotate, pinch, and stretch don't exist as one single action. Instead, a user starts a gesture, processes some actions, and finishes. Recognizing those gestures and making sense of what a user is trying to do is the task of a *gesture recognizer*.

This object can be found at Windows.UI.Input.GestureRecognizer. You can listen for pointer events, feed them to the gesture recognizer, and then process the events that the gesture recognizer raises.

You can create and configure a gesture recognizer like this:

```
var gr = new Windows.UI.Input.GestureRecognizer();
gr.showGestureFeedback = false;
gr.gestureSettings =
    Windows.UI.Input.GestureSettings.tap |
    Windows.UI.Input.GestureSettings.doubleTap |
    Windows.UI.Input.GestureSettings.rightTap |
    Windows.UI.Input.GestureSettings.hold |
    Windows.UI.Input.GestureSettings.holdWithMouse;
```

With the gestureSettings property, you define the types of gestures you want to recognize. In addition to the gestureSettings property, you can use other properties for specific settings such as crossSlideThresholds, crossSlideExact, and crossSlideHorizontally. You can listen for the following types of gestures:

- None
- Tap
- DoubleTap
- Hold
- HoldWithMouse
- RightTap
- Drag
- ManipulationTranslateX
- ManipulationTranslateY
- ManipulationTranslateRailsX
- ManipulationTranslateRailsY
- ManipulationRotate
- ManipulationScale
- ManipulationTranslateInertia
- ManipulationRotateInertia
- ManipulationScaleInertia
- CrossSlide
- ManipulationMultipleFingerPanning

You can create your own gestures by listening to combinations of these gesture types. After configuring your gesture recognizer, you can then add event handlers:

```
gr.addEventListener('manipulationstarted', function (e) { });
gr.addEventListener('manipulationupdated', function (e) { });
gr.addEventListener('manipulationcompleted', function (e) { });
gr.addEventListener('tapped', function (e) { });
gr.addEventListener('righttapped', function (e) {});
```

These events are called whenever the gesture recognizer translates the pointer events you feed it into a specific gesture.

To supply the gesture recognizer with pointer events, subscribe to the pointer events on your element and pass them to the gesture recognizer like this:

```
var target = document.getElementById("targetContainer");

target.addEventListener('pointerdown', function (evt) {
    var pp = evt.getCurrentPoint(target);
    gr.processDownEvent(pp);
}, false);
target.addEventListener('pointermove', function (evt) {
    var pps = evt.getIntermediatePoints(target);
    gr.processMoveEvents(pps);
}, false);
target.addEventListener('pointerup', function (evt) {
    var pp = evt.getCurrentPoint(target);
    gr.processUpEvent(pp);
}, false);
target.addEventListener('pointercancel', function (evt) {
    var pp = evt.getCurrentPoint(target);
    gr.processUpEvent(pp);
}, false);
target.addEventListener('wheel', function (evt) {
    var pp = evt.getCurrentPoint(target);
    gr.processMouseWheelEvent(pp, evt.shiftKey, evt.ctrlKey);
}, false);
```

This code extracts the correct data from the pointer event and calls the matching method on the gesture recognizer.

Remember that creating all the gesture recognizers upfront doesn't scale well when you have many gesture objects. Instead, you can create objects dynamically, starting on pointerdown and destroying them on MSGestureEnd.

MORE INFO **INPUT GESTURES AND MANIPULATIONS WITH GESTURERECOGNIZER**

Microsoft created a sample application that demonstrates how to use a gesture recognizer. You can find the sample at *http://code.msdn.microsoft.com/windowsapps/Manipulations-and-gestures-362b6b59*.

Listening to mouse events or touch gestures

Because of the way Windows translates pointer data into gestures and finally into regular HTML Document Object Model (DOM) events, you don't have to do very specific things to make your app usable by a mouse when it is already optimized for touch.

As Microsoft states, you should design the app touch first because when everything works with touch, a mouse also works. You can listen for pointer or gesture events, or just use the built-in controls that are optimized for different input styles.

Look for the use of a mouse wheel; when you have an element, attach a handler for the mouse wheel like this:

```
element.addEventListener("wheel", onMouseWheel, false);
```

Inside your event handler, set the pointerId to 1 and pass it to the pointerdown event handler:

```
function onMouseWheel(e) {
    e.pointerId = 1;
    onPointerDown(e);
}
```

Now you can use the mouse wheel to manipulate your object by zooming or rotating it in combination with keyboard presses.

When working with pointer events, you can check the pointerType property to see whether the input was generated by touch, mouse, or pen actions:

```
elm.addEventListener("pointerdown", handleDown, false);
function handleDown(evt) {
  if (evt.pointerType == "mouse") {
    // Do something for mouse input only
  } else {
    // Do something for non-mouse input
  }
}
```

Managing stylus input and inking

Besides using a mouse, keyboard, and touch input, users can also use a stylus. Because of the pointer– and gesture–based input mechanisms, you don't have to do anything special to enable users to use a stylus for regular input such as clicking or holding items.

One scenario in which users use a stylus is when inking. An HTML canvas element makes it easy to listen to raw pointer events and use the stylus to draw on the canvas. But inking goes a step farther: Ink is a data structure that maintains the actual input data such as pressure, angle, and velocity, so ink remembers how an image was drawn (so you can apply techniques such as handwriting recognition).

Inking starts with creating an instance of the Windows.UI.Input.Inking.InkManager class and an input element. By listening to pointerdown, pointermove, and pointerup events, you can pass those events to InkManager, which then creates InkStroke objects that you can render.

MORE INFO **INKING SAMPLES**

The Windows software development kit (SDK) contains two inking samples. One features a complete application with handwriting recognition; the other is a simplified version that focuses on just processing the inking. The Input: Simplified ink sample can be found at *http://code.msdn.microsoft.com/windowsapps/Input-simplified-ink-sample-11614bbf*. The Input Ink sample can be found at *http://code.msdn.microsoft.com/windowsapps/Ink-App-sample-61abaec3*.

Handling drag-and-drop events

When implementing dragging, it is easiest to use the built-in controls that WinJS offers you. The ListView control has support for dragging since WinJS 2.0; and ItemContainer (used in ListView) also supports dragging. Drag-and-drop support is implemented on the standard HTML5 drag-and-drop capabilities.

Follow these steps to implement dragging with the ListView control:

1. Set itemsDraggable to true on ListView.
2. Handle the itemdragstart event and configure the data you want to transfer.
3. Handle the dragover event on the target and signal that you accept the drop.
4. Handle the drop event on the target and transfer the data.

Implementing dragging and dropping looks like the code that follows. If you have an element ListView and dropTarget, you can add the required event handlers. The first one is dragstart:

```
listView.addEventListener("itemdragstart", function (eventObject) {
    eventObject.detail.dataTransfer.setData("Text", JSON.stringify(eventObject.detail.
dragInfo.getIndices()));
});
```

This event handler configures the data you want to transfer. In this case, it uses the index of the item that you will drag. In addition to itemdragstart, you can use itemdragend:

```
listView.addEventListener("itemdragend", function (eventObject) {
    dragging = false;
});
```

That’s all you have to do for the source of the drag transfer. The drop target needs to be configured to allow drops by using the event.preventDefault method:

```
var dropTarget = element.querySelector("#myDropTarget");
dropTarget.addEventListener("dragover", function (eventObject) {
    if (dragging) {
        // Allow HTML5 drops
        eventObject.preventDefault();
    }
});
```

By using dragenter and dragleave, you can respond to the draggable item entering and leaving your drop target:

```
dropTarget.addEventListener("dragenter", function (eventObject) {
    if (dragging) {
        WinJS.Utilities.addClass(dropTarget, "drop-ready");
    }
});
dropTarget.addEventListener("dragleave", function (eventObject) {
    WinJS.Utilities.removeClass(dropTarget, "drop-ready");
});
```

Finally, there is the drop event handler, which is the handler that processes the actual dragging operation by removing the item from the source and adding it to the target or by some other operation. In this case, the index of the item is retrieved, and a message is displayed:

```
dropTarget.addEventListener("drop", function (eventObject) {
    // Get indicies -> keys of items that were dropped
    WinJS.Utilities.removeClass(dropTarget, "drop-ready");
    if(dragging) {
        var indexSelected = JSON.parse(eventObject.dataTransfer.getData("Text"));
        if (Array.isArray(indexSelected) && typeof indexSelected[0] === "number") {
            var listview = document.querySelector("#listView").winControl;
            var ds = listview.itemDataSource;

            ds.itemFromIndex(indexSelected[0]).then(function (item) {
                WinJS.log && WinJS.log("You dropped the item at index " + item.index +
", "
                + item.data.title, "sample", "status");
            });
        }
    }
});
```

MORE INFO **LISTVIEW DRAG-AND-DROP SAMPLE**

The Windows SDK contains a sample that shows different ways to implement drag with a ListView control. The previous code was taken from scenario 2 in this sample. You can find the complete sample at *http://code.msdn.microsoft.com/windowsapps/HTML-ListView-reorder-and-ffd280d9*.

Because the dragstart and dragend events are defined by the HTML5 specification, you can also use them on an ItemContainer control. Enabling ItemContainer as a drag source is easy:

```
myItemContainer.addEventListener("dragstart", function (eventObject) {
     var dragData = {
|                     sourceId: myDragContent.id,
                      data: myItemTitle.innerText,
                      imgSrc: myImg.src
                    };
     eventObject.dataTransfer.setData("Text", JSON.stringify(dragData));
 });
```

ListView also supports some specialized events such as itemdragenter and itemdragdrop. These events get more data passed to them than the default drag-and-drop events. For example, the event argument passed to itemdragdrop tells you the specific index where a user wants to drop an item through the event.detail.insertAfterIndex property. Nothing prevents you from using the drag-and-drop HTML5 elements, especially when you implement dragging between standard HTML controls and a ListView control. But these special events are useful when you implement dragging between WinJS controls.

Thought experiment

Managing input

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 a ToDo app to manage lists of tasks and share them with others. You are wondering about the different types of inputs and which ones you should support.

Answer the following questions:

1. **What types of input can you expect?**
2. **Should you differentiate between input devices?**

Objective summary

- All input in Windows Store apps is mapped to raw pointer data, which contains specific info about the type of device (such as a mouse, keyboard, touch, or stylus) that created the input.
- Gestures use the pointer data to create complex input actions such as panning, holding, and tapping.
- A gesture recognizer can be configured to convert pointer data to specific custom gestures for use in your app.
- Because of the way Windows merges different input devices into pointer data, you can design your app touch first and (as a result) support other input devices.
- If required, you can differentiate pointer input by the device that created it and can then respond appropriately.
- Inking is implemented by ink data structures that not only contain the final image but also contain all actions used to create it. These structures can be used to implement complex processing such as handwriting recognition.
- Dragging is implemented by using the standard HTML5 events. WinJS controls such as ListView implement dragging out of the box.

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 are the standard Windows 8 gestures? (Choose all that apply.)
   - **A.** Tap
   - **B.** Double-tap
   - **C.** Press and hold
   - **D.** Slide
   - **E.** Swipe
   - **F.** Turn
   - **G.** Pinch
2. You are listening for the MSGestureTap event, but no events are firing. What have you forgotten?
   - **A.** You need to subscribe to the click event and convert it into a gesture.
   - **B.** You need to initialize a gesture recognizer to recognize tap gestures.
   - **C.** You should listen for the MSGestureChange event and check its type property for tapping gestures.
   - **D.** You need to listen for the pointerdown event and send it to the MSGesture object.
3. You want to configure ListView as a drag source. Which events should you handle? (Choose all that apply.)
   - **A.** itemdragstart
   - **B.** itemdragend
   - **C.** dragover
   - **D.** dragenter
   - **E.** dragleave
   - **F.** drop

Objective 4.2: Design and implement navigation in an app

Your app probably has more than one page; most apps contain multiple pages and offer users a way to navigate between those pages. This objective discusses how to implement navigation so it meets your app's requirement.

You also learn about semantic zoom, an authentically digital way to let a user navigate through a group of content. Finally, you see how to load HTML fragments into your app.

This objective covers how to:

- Handle navigation events, check navigation properties, and call navigation functions by using the WinJS.Navigation namespace
- Design navigation to meet app requirements
- Use semantic zoom
- Load HTML fragments

Handling navigation events, checking navigation properties, and calling navigation functions by using the WinJS.Navigation namespace

A typical web application exists of different HTML files and uses hyperlinks to navigate between pages. Each page defines its own set of JavaScript and CSS files and is completely isolated from other pages. By keeping track of session data on the web server or using HTML5 features such as local storage, you can transfer state from one page to another.

Moving from page to page also means that the browser shows a white page at the beginning of loading a new page. After the HTML and CSS are parsed, the new content shows.

You are used to loading data on navigation when browsing the web. Web developers try to make page loading as fast as possible, but sometimes you just have to wait.

One of the Microsoft design principles discussed in Chapter 1 is that an app should be fast and fluid. So Windows Store apps use a different approach from traditional webpages, which is more like the asynchronous JavaScript and XML (AJAX) approach that modern web applications use.

Instead of loading a completely new page when a user navigates, load all content upfront and use JavaScript to update the DOM instantaneously. This pattern is called a single-page application (SPA). You start with a single HTML page and then use JavaScript to load new content and show it on your start page.

This architecture is not something explicitly required when a Windows Store app is built. You can implement navigation any way you want, and you are not required to use the WinJS controls that implement page navigation. However, using those controls saves you a lot of work and helps you create a fast and fluid app.

Creating an SPA requires two important elements:

- A component to keep track of navigation history
- A component to load HTML pages with CSS and JavaScript

The first is offered to you by WinJS.Navigation, which is part of the WinJS library and implements a basic navigation stack. It keeps track of a list of URIs and exposes properties such as canGoBack, canGoForward, and state. You can manipulate the stack by calling methods such as forward, back, and navigate.

Whenever you go from one URI to another, the Navigation class raises events that you can listen to. By itself, the Navigation class implements only half of the navigation framework. If no one listens to the events that the Navigation class raises, nothing else happens.

The PageControlNavigator class can help in this situation. It listens to navigation events and updates the DOM accordingly by adding and removing items. The PageControlNavigator class is not a part of WinJS; it is created by the project templates in Visual Studio, and its code is included in your app. Having the code allows you to customize the PageControlNavigator class according to your requirements and is the missing link between WinJS.Navigation and custom pages.

You can see all those elements in action when you create a new project based on the Navigation App template, which creates default.html, default.js, and default.css files just as the Blank template does. The template also adds navigator.js, which contains PageControlNavigator, and adds your first page with a corresponding CSS and JavaScript file called home.

Your default.html file contains the following markup in its body tag:

```
<div id="contenthost" data-win-control="Application.PageControlNavigator"
data-win-options="{home: '/pages/home/home.html'}"></div>
```

This control hosts the app content. When loading, it navigates to your home.html page. Your home page uses the home.js file with the following content:

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

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

The WinJS.UI.Pages.define method is used to create a new Page. The URI used here should be an exact match of the one you use on your PageControlNavigator object.

A WinJS.UI.Page defines a couple of methods that you can override:

- **Init** Called before elements from the page control have been created.
- **Processed** The Page control automatically calls WinJS.UI.processAll. When that's complete, the processed method runs.
- **Ready** Called after the page has been added to the DOM.
- **Error** Called if an error occurs in loading or rendering the page.
- **Unload** Called when navigation has left the page. By default, WinJS automatically disposes of controls on a page when that page is unloaded.

- **UpdateLayout** Called in response to the window.onresize event, which signals changes between various view states.

The default templates implement the ready method, which is the method you use most often. However, you can use the other methods in your Page objects if required.

The home.html file contains the following markup:

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

    <!-- WinJS references -->
    <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

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

References to the WinJS CSS and JavaScript files are there only so you can view the page outside of the containing div in the default html file. JavaScript and CSS files that are already loaded are not loaded again.

The markup defines a header with a WinJS.UI.BackButton. This button subscribes to the WinJS.Navigation events and makes sure that it shows up only when you can actually navigate back to something.

The default.js file contains the basic code to navigate to the first page and saves the history whenever an app is suspended:

```
nav.history = app.sessionState.history || {};
nav.history.current.initialPlaceholder = true;
nav.navigate(nav.location || Application.navigator.home, nav.state);
```

This code is used in promises to make sure the application stays responsive. It navigates to the page saved in your history or the home page that you defined in the markup and then passes along any state.

You just learned the basics of navigation in WinJS apps. By using the PageControlNavigator, WinJS.Navigation, and the Page objects, you can easily implement an SPA design in which you can split your application into separate pages that are automatically loaded into your default.html file.

Navigating to a page is nothing more than calling WinJS.Navigation.navigate and passing the correct URI. You can also call WinJS.Navigation.back and WinJS.Navigation.forward to move backward or forward in the navigation stack. Calling these methods raises the correct events, making PageControlNavigator load your Page objects.

EXAM TIP

Because all project templates except the Blank template implement navigation, you should definitely study those samples to become familiar with the way navigation works in Windows Store apps. All Windows SDK samples also use navigation to show different scenarios, so you have plenty of examples to learn from.

Designing navigation to meet app requirements

The first chapter of this book showed you how to choose the correct project templates to start your app. Each project template has its own different navigation styles.

The Blank template doesn't implement any navigation at all, whereas the Hub and Grid templates use navigation to enable movement between an overview, groups, and individual items.

Tapping a group header invokes a WinJS.Navigation.navigate method that sends you to the correct group. Tapping an item directly takes you to a details page in which you can use the back button to navigate to the previous page.

If those navigation styles don't suit your particular app, you can choose a Flat style that has no hierarchy. In Internet Explorer, you can move between different tabs that you have opened, for example. You can implement such a style by calling WinJS.Navigation.navigate and then navigating directly to a specific page. Because there is no hierarchy, users don't expect a back button to return to their previous location.

When designing your navigation strategy, remember that because you are using a single-page architecture, CSS and JavaScript files are loaded after you a reach a page and are then kept in the app's scope. Having all JavaScript objects kept in scope means that you might have conflicts with JavaScript variables and functions that are used on different pages but are loaded side by side into your app. Using namespaces and classes to encapsulate JavaScript elements is very important to create a maintainable app.

The same is true when it comes to CSS. If a page loads a CSS file, those styles are then applied everywhere in your app. You can avoid this problem by scoping items correctly. The dark and light CSS files in WinJS already do this for you. All styles that are conflicting between those files are applied scoped under a dark or light selector: win-ui-light and win-ui-dark, respectively. Apply those classes to the top-level elements on your pages to ensure that you can mix both dark and light CSS files in one app without creating conflicts.

If you have conflicting CSS rules on different pages, you should also scope them to a specific selector so each page can choose which rules to use.

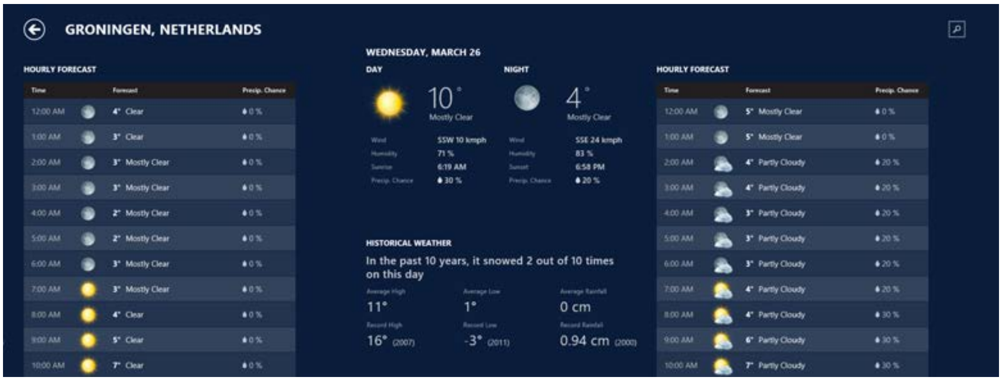
Another important aspect is performance. The PageControlNavigator class removes Page objects from the DOM completely whenever you navigate to another URI, which frees memory and makes sure that all data is unloaded upon navigation.

However, if a user navigates through a set of data and often returns to the same page, the same page has to be loaded over and over again. Fortunately, you can change it to suit your particular app needs because PageControlNavigator is part of the project. Maybe you want to keep the HTML in memory and process the data binding only on child items, or you might need to keep the complete overview page with all data in memory and only show and hide it. The scenario you choose depends on your app; be aware of the possibilities.

Using semantic zoom

Semantic zoom is a control implemented in WinJS that enables a user to easily switch between different views of the same data. Semantic zoom is not like zooming in and out to view the same data but on a different scale. Instead, semantic zoom embraces the concept of being authentically digital. In the digital world, you can achieve unique scenarios that aren't possible in the real world.

For example, when you look at a forecast in the weather app, you see a detailed view of the next few days (see Figure 4-3).



GRONINGEN, NETHERLANDS

HOURLY FORECAST

Time	Forecast	Precip. Chance
12:00 AM	4° Clear	0 %
1:00 AM	3° Clear	0 %
2:00 AM	3° Mostly Clear	0 %
3:00 AM	3° Mostly Clear	0 %
4:00 AM	2° Mostly Clear	0 %
5:00 AM	2° Mostly Clear	0 %
6:00 AM	3° Mostly Clear	0 %
7:00 AM	3° Mostly Clear	0 %
8:00 AM	4° Clear	0 %
9:00 AM	5° Clear	0 %
10:00 AM	7° Clear	0 %

WEDNESDAY, MARCH 26

DAY		NIGHT	
10° Mostly Clear		4° Mostly Clear	
Wind	SSW 10 kmph	Wind	SSE 24 kmph
Humidity	71 %	Humidity	83 %
Sunrise	6:19 AM	Sunset	6:58 PM
Precip. Chance	30 %	Precip. Chance	20 %

HISTORICAL WEATHER

In the past 10 years, it snowed 2 out of 10 times on this day

Average High	Average Low	Average Rainfall
11°	1°	0 cm
Record High	Record Low	Record Rainfall
16° (2007)	-3° (2011)	0.94 cm (2000)

HOURLY FORECAST

Time	Forecast	Precip. Chance
12:00 AM	5° Mostly Clear	0 %
1:00 AM	5° Mostly Clear	0 %
2:00 AM	4° Partly Cloudy	20 %
3:00 AM	4° Partly Cloudy	20 %
4:00 AM	3° Partly Cloudy	20 %
5:00 AM	3° Partly Cloudy	20 %
6:00 AM	3° Partly Cloudy	20 %
7:00 AM	4° Partly Cloudy	20 %
8:00 AM	4° Partly Cloudy	30 %
9:00 AM	6° Partly Cloudy	30 %
10:00 AM	7° Partly Cloudy	30 %

FIGURE 4-3 The weather app shows a detailed forecast for the next few days

What do you expect when you zoom out of this view? A real-world experience would show all elements on the screen smaller so additional days could fit. But that's not what the weather app does. When you zoom out from an individual day, you see what's shown in Figure 4-4.

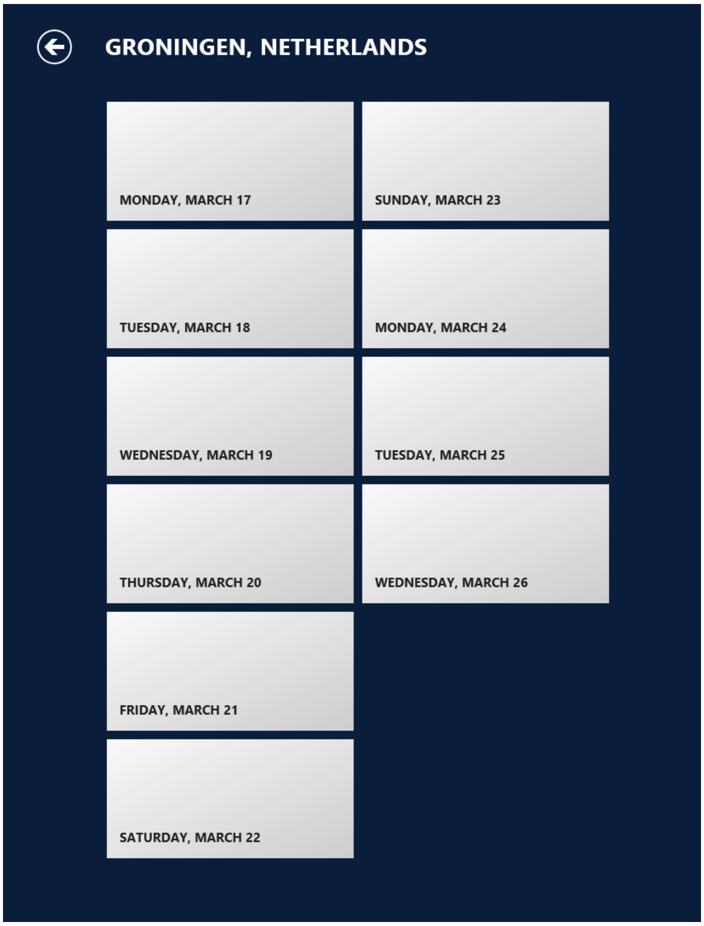


FIGURE 4-4 The zoomed-out view of the forecast for the next few days

Semantic zoom adds extra functionality to your application by letting users zoom out to get a different view of your data.

EXAM TIP

Semantic zoom is more than just zooming out and seeing a larger part of some content. Semantic zoom shows content on multiple logical levels.

By tapping one of the items, the user zooms in again and views the details of that item. When zooming out, you can group items, show headers and categories, or use some other layout to give a different view of the data. For example, when viewing a report with data, you could zoom out to switch to a chart showing a condensed version of the data.

Semantic zoom is easily implemented with the SemanticZoom control. This control allows users to switch between two different views of the data by using familiar gestures such as pinch and stretch, using Ctrl+scroll wheel, or pressing Ctrl+plus (+) or Ctrl+minus (-).

A SemanticZoom control contains two controls: one for the zoomed-in view and one for the zoomed-out view. These controls need to implement the IZoomableView interface. Currently, ListView is the only WinJS control that implements this interface.

To use ListView in a SemanticZoom control, you need an IListDataSource with grouped data such as a WinJS.Binding.List. If you have this data available, you can then add a SemanticZoom control like this:

```
<div id="semanticZoomDiv" data-win-control="WinJS.UI.SemanticZoom">
    <!-- The zoomed-in view. -->
    <div id="zoomedInListView"
        data-win-control="WinJS.UI.ListView"
        data-win-options="{ itemDataSource: myData.groupedItemsList.dataSource,
                            itemTemplate: select('#mediumListIconTextTemplate'),
                            groupHeaderTemplate: select('#headerTemplate'),
                            groupDataSource: myData.groupedItemsList.groups.dataSource,
                            selectionMode: 'none',
                            tapBehavior: 'none',
                            swipeBehavior: 'none' }"></div>
    <!--- The zoomed-out view. -->
    <div id="zoomedOutListView"
        data-win-control="WinJS.UI.ListView"
        data-win-options="{ itemDataSource: myData.groupedItemsList.groups.dataSource,
                            itemTemplate: select('#semanticZoomTemplate'),
                            selectionMode: 'none',
                            tapBehavior: 'invoke',
                            swipeBehavior: 'none' }"></div>
</div>
```

Two WinJS.UI.ListViews are defined: one has an ID of zoomedInListView, and the other has an ID of zoomedOutListView. They use the same data source, but with different templates and options.

The SemanticZoom control exposes one event: onzoomchanged, which is raised whenever the control zooms in or out. You can subscribe to this event if you want to be notified when the user changes views.

The SemanticZoom control also has a couple of important properties, such as enableButton (true if you want to show the zoom-out button inside of the control), locked (to enable or disable zooming), zoomedOut (true if the control is currently zoomed out), and zoomFactor. ZoomFactor is used to determine the amount of scaling done when going from one view to another. Lower values make the effect more visible.

MORE INFO **COMPLETE SEMANTICZOOM CONTROL SAMPLE**

A working example of the SemanticZoom control with two ListView controls can be found at *http://code.msdn.microsoft.com/windowsapps/ListView-grouping-and-6d032cc1/*.

ListView is the only WinJS control that implements IZoomableView, but you can implement this interface yourself. The IZoomableView interface defines the following methods:

- **beginZoom** Initiates semantic zoom on the custom control
- **configureForZoom** Initializes the semantic zoom state for the custom control
- **endZoom** Terminates semantic zoom on the zoomed-in or zoomed-out child of the custom control
- **getCurrentItem** Retrieves the current item of the zoomed-in or zoomed-out child of the custom control
- **getPanAxis** Retrieves the panning axis of the zoomed-in or zoomed-out child of the custom control
- **handlePointer** Manages pointer input for the custom control
- **positionItem** Positions the specified item within the viewport of the child control when panning or zooming begins
- **setCurrentItem** Selects the item closest to the specified screen coordinates

Define a property named zoomableView on your control and let that property return an object that implements all the methods in the IZoomableView interface.

Methods such as getPanAxis can return only a value of horizontally or vertically. Methods such as setCurrentItem get a pair of coordinates (x,y) and have to map them to an item on the screen. The beginZoom and endZoom methods switch from one view to another.

Although not an easy undertaking, implementing custom controls to use with Semantic-Zoom, such as calendars or graphs, can improve the usability of your app and give users a unique insight into your data.

MORE INFO **CUSTOM CONTROL WITH SEMANTICZOOM SAMPLE**

You can find a sample of implementing a custom control for SemanticZoom in the Windows SDK at *http://code.msdn.microsoft.com/windowsapps/SemanticZoom-for-custom-4749edab/*.

Loading HTML fragments

The navigation elements you looked at in this objective are based on Pages that get loaded into the DOM. If you want more control over how the HTML is loaded, you can use the WinJS.UI.Fragments namespace.

WinJS.UI.Fragments.renderCopy takes an URI pointing to an HTML file and copies the rendered content into an element you specify:

```
WinJS.UI.Fragments.renderCopy("/html/myfragment.html",
    this.basicFragmentLoadDiv)
    .done(
        function () {
       // success
        },
        function (error) {
       // error
        }
    );
```

This code automatically sets basicFragmentLoadDiv to the rendered HTML. If you want to process the HTML before sending it to the DOM (or without even sending it to the DOM), you can use a done function that takes the fragment as an argument:

```
WinJS.UI.Fragments.renderCopy("/html/myfragment.html")
     .done(function (fragment) { });
```

When the fragment you load contains WinJS controls, call WinJS.UI.processAll(targetElement), which converts all regular HTML elements that have a data-win-control attribute into WinJS controls. The Windows SDK sample shows how to do this with the following code:

```
fragmentsWithControls: function () {
    var that = this;
    this.fragmentsWithControlsDiv.innerHTML = "";
   // Read fragment from the HTML file and load it into the div.  Note
    // WinJS.UI.Fragments.renderCopy() returns a promise which we attach a done() call
    // in order to perform additional processing or handle errors that may have
    // occurred during the renderCopy() action.
    // Passing the DOM element as the second argument will get renderCopy to parent
    // the fragment to that DOM element automatically.
    WinJS.UI.Fragments.renderCopy("/html/3_FragmentsWithControls_Fragment.html",
        this.fragmentsWithControlsDiv)
        .done(function() {
            // After the fragment is loaded into the target element,
            // WinJS.UI.processAll() needs to be called to activate the
            // controls and process options records.
            WinJS.UI.processAll(that.fragmentsWithControlsDiv);
            WinJS.log && WinJS.log("successfully loaded fragment", "sample", "status");
        },

        function(error) {
            WinJS.log && WinJS.log("error loading fragment: " + error, "sample",
"error");
        }
    );
}
```

The reference to that.fragmentsWithControlsDiv comes from the first line in the fragmentsWithControls method:

```
var that = this;
```

Capturing the this parameter is required to make sure that you still reference it when your promise returns. When the div is passed to processAll, WinJS initializes any controls that are inside your HTML fragment.

JavaScript and CSS files that are referenced from the fragment are automatically loaded. CSS rules are then applied to your HTML, and you can call JavaScript methods that are defined in the loaded JavaScript.

MORE INFO **LOADING HTML FRAGMENTS SAMPLE**

The Windows SDK contains an example that shows different ways to load an HTML fragment. You can find it at *http://code.msdn.microsoft.com/windowsapps/Fragments-91f66b07*.

Thought experiment

Navigating through 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 asked to give a presentation to your colleagues on how navigation in a Windows Store app works.

Answer the following questions:

1. **Why is the single-page architecture important?**
2. **What is the relationship between WinJS.Navigation, PageControlNavigator, and WinJS.Page?**
3. **Is semantic zoom a form of navigation? When should you use it?**

Objective summary

- Windows Store apps can follow a single-page architecture.
- WinJS.Navigation keeps a navigation history and has methods such as navigate, forward, and back that raise navigation events.
- PageControlNavigator is created as a part of the default project templates and listens for the navigation events that WinJS.Navigation raises. It then processes the actual navigation by loading the Page and placing it in the DOM.

- WinJS.Page objects can be used to define separate HTML documents with JavaScript and CSS files that can be loaded through navigation.
- The SemanticZoom control allows you to easily implement semantic zoom with a ListView or with a custom control by implementing IZoomableView.
- You can load HTML fragments by using the WinJS.UI.Fragments namespace.

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 want to navigate one page back. Which class should you use?
   - **A.** WinJS.Navigation
   - **B.** PageControlNavigator
   - **C.** WinJS.UI.Pages
   - **D.** WinJS.UI.BackButton
2. You want to implement semantic zoom on a set of data. Which steps should you take? (Choose all that apply.)
   - **A.** Add a WinJS.UI.SemanticZoom control.
   - **B.** Add two WinJS.UI.ListView objects with templates.
   - **C.** Implement IZoomableView on your data.
   - **D.** Group your data.
3. You use the light theme on one page and the dark theme on another page in your app. However, when a user navigates through your app, the two styles seem to conflict. What should you do?
   - **A.** Change default.html to default-light.html and add a default.dark.html. Use them to reference the correct theme and to load any pages.
   - **B.** Use WinJS.UI.fragments to load HTML without any CSS data.
   - **C.** Use the win-ui-light and win-ui-dark CSS classes to scope the CSS styles to the correct pages.
   - **D.** Change the PageControlNavigator object to remove all link elements when navigating between pages.

Objective 4.3: Create and manage tiles

When you look at the Start screen of your Windows 8 device, one thing immediately catches your eye: tiles. Each app is represented by a tile, and tiles are not just icons! Tiles are alive with content. Tiles show updates and other important information in the user's Start screen without having to run the app.

Having a great tile gets your users' attention and ensures that they return to your app. This objective discusses everything you can do with tiles.

This objective covers how to:

- Create and update tiles and tile contents
- Create and update badges (the TileUpdateManager class)
- Respond to notification requests
- Choose an appropriate tile update schedule based on app requirements

Creating and updating tiles and tile contents

Windows Store apps use various sizes and types for their tiles, which come in small, medium, wide, and large sizes. A tile can be static, or it can be a live tile receiving notifications from your app or from some external web service.

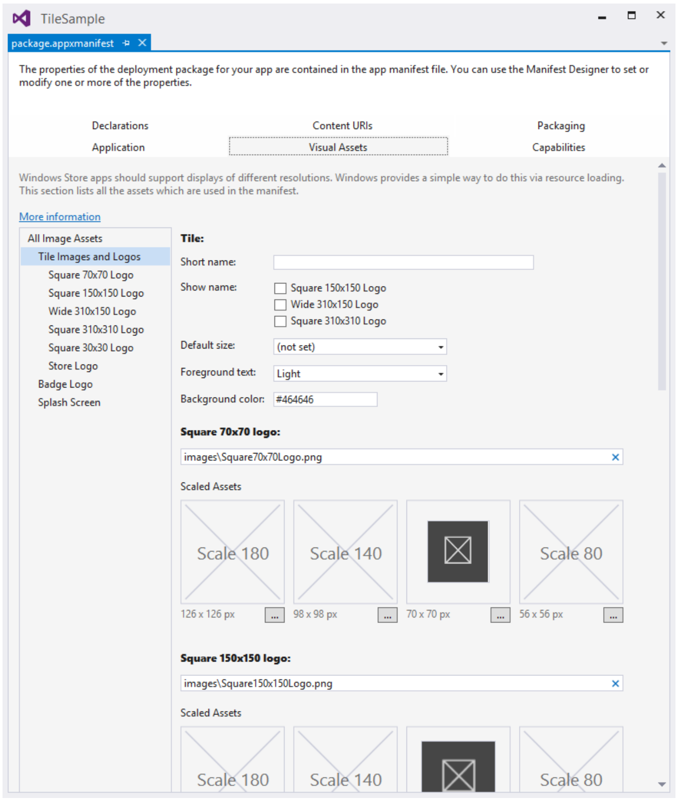
Users can configure the size of the tile, which should always support a small and medium width. If your app supports live tiles, you can also choose to use the large and wide options. Users can also disable and enable notifications on a live tile.

When your app is installed from the store, it begins with a default static tile. You define this tile in the app manifest. When you start sending notifications, the tile gets updated and can show all kinds of content.

The App Manifest Designer shown in Figure 4-5 shows how to configure tile sizes.

By adding assets for the 70 x 70 (small), 150 x 150 (medium), 310 x 150 (wide), and 310 × 310 large) settings, you configure which tile sizes your app supports (the 30 x 30 setting shown in Figure 4-5 is used for icons used in Windows Explorer. It is not used for tile sizes). Windows uses the images you supply as the default image for your tile until it starts receiving notifications.

Figure 4-6 shows the four tile sizes you can use in your app.



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.

Windows Store apps should support displays of different resolutions. Windows provides a simple way to do this via resource loading. This section lists all the assets which are used in the manifest.

Tile:

Short name:

Show name: ☐ Square 150x150 Logo ☐ Wide 310x150 Logo ☐ Square 310x310 Logo

Default size: (not set)

Foreground text: Light

Background color: #464646

Square 70x70 logo:

images\Square70x70Logo.png

Scaled Assets

Square 150x150 logo:

images\Square150x150Logo.png

Scaled Assets

FIGURE 4-5 The App Manifest Designer shows tile settings and assets for selection

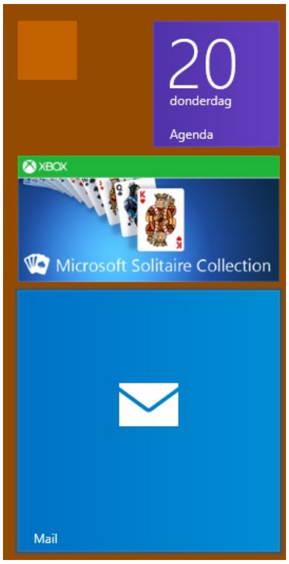


FIGURE 4-6 The four different tile sizes: small, medium, wide, and large

Tiles become interesting when you start updating them. You can choose from predefined XML templates and populate them with custom data to create a live tile.

EXAM TIP

Tiles don't have a free format. When you design a tile during the exam, make sure that you use one of the predefined layouts.

Those XML templates have support for all tile sizes and support combinations of text and images. An example of a wide tile template with an image and text is this:

```
<tile>
    <visual version="2">
        <binding template="TileWide310x150ImageAndText01" fallback="TileWideImageAndTe
xt01">
            <image id="1" src=""/>
            <text id="1"></text>
        </binding>
    </visual>
</tile>
```

MORE INFO **TILE TEMPLATES**

You can find a list of tile templates at *http://msdn.microsoft.com/en-us/library/windows/apps/hh761491.aspx*. Here you find examples and the XML for all templates defined by Windows.

By using this template, you can add content for the image and text, and then use it as a new tile template. The following code shows you how to do this from JavaScript:

```
var notifications = Windows.UI.Notifications;

var template = notifications.TileTemplateType.tileWide310x150ImageAndText01;
var tileXml = notifications.TileUpdateManager.getTemplateContent(template);

var tileTextAttributes = tileXml.getElementsByTagName("text");
tileTextAttributes[0].appendChild(tileXml.createTextNode("Hello World!"));

var tileImageAttributes = tileXml.getElementsByTagName("image");
tileImageAttributes[0].setAttribute("src", "ms-appx:///images/myimage.png");
tileImageAttributes[0].setAttribute("alt", "red graphic");

var tileNotification = new notifications.TileNotification(tileXml);

var currentTime = new Date();
tileNotification.expirationTime = new Date(currentTime.getTime() + 600 * 1000);

notifications.TileUpdateManager.createTileUpdaterForApplication()
                                .update(tileNotification);
```

This code uses the Windows.UI.Notification namespace to start updating the apps tile. All XML templates are defined by Windows; you don't have to create them yourself. Instead, you can use the TileTemplateType enumeration to get the names of the existing templates and then pass this name to TileUpdateManager.getTemplateContent to get the XML for the template.

Now that you have the XML, you have to update the attributes for the image and the text. Because you know the schema up front, you can query for the correct items and update them.

The next step is to create a TileNotification based on your XML and set an expiration time on your notification. The result can then be used to update the tile.

In this example, a tile update for a 310 x 150 tile is created, but you can't be sure that a user specified that tile size for the Start screen. When sending updates, you should send updates for all the sizes that your app supports. An exception is the 70 x 70 tile; it is a shrunk version of the 150 x 150 that doesn't support live tile updates.

You can send multiple sizes at once by retrieving the XML for the additional items and then appending this XML to your result. That way, you send the content for different tile sizes in one update to Windows:

```
var squareTemplate = notifications.TileTemplateType.tileSquare150x150Text04;
var squareTileXml = notifications.TileUpdateManager.getTemplateContent(squareTemplate);
var squareTileTextAttributes = squareTileXml.getElementsByTagName("text");
squareTileTextAttributes[0].appendChild(squareTileXml.createTextNode("Hello World! "));
var node = tileXml.importNode(
               squareTileXml.getElementsByTagName("binding").item(0), true);
tileXml.getElementsByTagName("visual").item(0).appendChild(node);
```

This code creates a new small XML template and then appends the XML to the overall result.

When required, you can also clear the content of your tile:

```
Windows.UI.Notifications.TileUpdateManager.createTileUpdaterForApplication().clear();
```

This command reverts to the original tile asset that you specified in your app manifest.

MORE INFO **WINDOWS SDK TILE SAMPLE**

The Windows SDK contains a sample of working with tiles. You can find it at *http://code.msdn.microsoft.com/windowsapps/app-tiles-and-badges-sample-5fc49148*.

Although working with the XML templates directly can be difficult, you can use the NotificationsExtensions library in your code. This is a C# library that creates a wrapper around the XML templates so you can use them more easily. This library is used in the Windows SDK samples, so you can use it in your own code. You need to add a reference to your project that points the library. After that, you can use the following code:

```
var tileContent = NotificationsExtensions.TileContent.TileContentFactory.
                        createTileSquare310x310Text09();
tileContent.textHeadingWrap.text = "Hello World!";
```

Instead of working directly with XML and locating the correct elements, the NotificationsExtensions library allows you to create a new tile and use properties to set its content. Although this information is not a part of the exam, knowing about it can be very useful.

Windows supports secondary tiles, which can be pinned to the Start screen by users, but they deep-link into your applications. For example, the People app has a regular tile, but you can also create secondary tiles that go straight to a person.

Creating a secondary tile is done with the Windows.UI.StartScreen.SecondaryTile class. This class expects a title, arguments that are passed to your app when the user activates your app through the secondary tile, the default logo, and the tile size:

```
var tile = new Windows.UI.StartScreen.SecondaryTile("tile_id",
    "Title",
    "activiation arguments for this tile",
    square150x150Logo,
    Windows.UI.StartScreen.TileSize.Square150x150);
```

Now you can further configure your tile with additional sizes and then show a confirmation dialog box to users, asking whether they want to pin the tile to the Start screen. Without user consent, you can never add a secondary tile. You normally show this dialog box when the user presses a button on the app bar or somewhere in your layout. Make sure to position the dialog box correctly so it displays near the button:

```
var selectionRect = document.getElementById("pinButton").getBoundingClientRect();
tile.requestCreateForSelectionAsync({
                                      x: selectionRect.left,
                                      y: selectionRect.top,
                                      width: selectionRect.width,
                                      height: selectionRect.height
                                    },
                                      Windows.UI.Popups.Placement.below).
    done(function (isCreated) {
    if (isCreated) {
        // success
    } else {
       // error
    }
});
```

Unpinning a secondary tile is done by specifying the ID you used when creating it:

```
var tileToDelete = new Windows.UI.StartScreen.SecondaryTile("tile_id ");

tileToDelete.requestDeleteAsync();
```

This code shows a flyout to the user requesting permission to remove the secondary tile. As with requesting permissions to create a tile, you also need to position the delete flyout correctly.

Now that there is a secondary tile, the user launches the app by using this tile and expects to end up in the right place. You now use the activation arguments that you specified when creating the tile. When your app launches, check for those arguments in the activated event and take action on them:

```
function activated(eventObject) {
        if (eventObject.detail.kind ===
                  Windows.ApplicationModel.Activation.ActivationKind.launch) {
            if (eventObject.detail.arguments !== "") {
                // arguments contains the arguments that you specified when creating the
                // secondary tile. You can parse them here and show the correct page in
                // your app to the user.
                }));
            } else {
                // default behavior of loading the start page
            }
        }
    }
```

MORE INFO **SECONDARY TILE SAMPLE**

You can find an example showing how to create and use secondary tiles at *http://code.msdn.microsoft.com/windowsapps/secondary-tiles-sample-edf2a178/*.

Creating and updating badges (the TileUpdateManager class)

Figure 4-7 shows the Windows SDK Tiles JS sample tile pinned to the Start screen. In the bottom-right corner, you see the number 5. This number is called a badge.

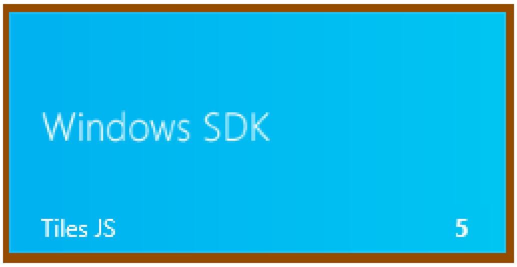


FIGURE 4-7 The Tiles JS sample showing the title, name, and a badge update

A badge is a number or a glyph used to indicate the app's status in some way (for example, the number of unread email messages or an alarm that is set). A badge is not part of the XML templates that you saw in the previous sections; it is an overlay on your tile that is updated through its own application programming interface (API).

To create a number badge, use the following code:

```
var notifications = Windows.UI.Notifications;
var badgeType = notifications.BadgeTemplateType.badgeNumber;
var badgeXml = notifications.BadgeUpdateManager.getTemplateContent(badgeType);
var badgeAttributes = badgeXml.getElementsByTagName("badge");
badgeAttributes[0].setAttribute("value", "42");
var badgeNotification = new notifications.BadgeNotification(badgeXml);
notifications.BadgeUpdateManager
             .createBadgeUpdaterForApplication().update(badgeNotification);
```

Specify which type of badge you want to create: number or glyph. In this case, you create a number badge. Like tiles, badges use an XML schema that you retrieve and then update. By using BadgeUpdateManager, you can then send your new badge notification to the Start screen.

To create a glyph badge, use the following code:

```
var badgeType = notifications.BadgeTemplateType.badgeGlyph;
var badgeXml = notifications.BadgeUpdateManager.getTemplateContent(badgeType);
var badgeAttributes = badgeXml.getElementsByTagName("badge");
badgeAttributes[0].setAttribute("value", "newMessage");
var badgeNotification = new notifications.BadgeNotification(badgeXml);
notifications.BadgeUpdateManager
             .createBadgeUpdaterForApplication().update(badgeNotification);
```

The available badge glyphs are described in Table 4-2.

TABLE 4-2 Badge glyphs

Status	Glyph	XML
none	No badge shown	<badge value="none"/>
activity	🔄	<badge value="activity"/>
alarm	🔔	<badge value="alarm"/>
alert	✱	<badge value="alert"/>
available	🟢	<badge value="available"/>
away	🟠	<badge value="away"/>
busy	🔴	<badge value="busy"/>

Status	Glyph	XML
newMessage	✉	<badge value="newMessage"/>
paused	⏸	<badge value="paused"/>
playing	▶	<badge value="playing"/>
unavailable	●	<badge value="unavailable"/>
error	⊗	<badge value="error"/>
attention	❗	<badge value="attention"/>

Responding to notification requests

Notifications are created by your app when the user processes some piece of code. Until now, you have used a local notification.

You have four different options when working with both tile and badge updates: local, scheduled, periodic, and push. You can choose one or combine multiple options that best suit your app.

Scheduled notifications

A *scheduled notification* is created inside your app when the user runs it. Instead of creating a notification that takes effect immediately, you can schedule a notification for a specific time. Calendar appointments or any feature events that you want to appear at a specific date and time are good examples.

Start with the code you saw previously to create a new tile update:

```
var notifications = Windows.UI.Notifications;
 var template = notifications.TileTemplateType.tileWide310x150ImageAndText01;
 var tileXml = notifications.TileUpdateManager.getTemplateContent(template);
 var tileTextAttributes = tileXml.getElementsByTagName("text");
 tileTextAttributes[0].appendChild(tileXml.createTextNode("Hello World! My very own tile 
notification"));
 var tileImageAttributes = tileXml.getElementsByTagName("image");
 tileImageAttributes[0].setAttribute("src", "ms-appx:///images/myimage.png");
 tileImageAttributes[0].setAttribute("alt", "red graphic");
```

This code creates a tile of 310 x 150 with text and an image, and sets the appropriate attributes.

Now you can specify a schedule for when the tile update should appear:

```
var currentTime = new Date();
var startTime = new Date(currentTime.getTime() + 3 * 1000);
var scheduledTile = new Windows.UI.Notifications.
                          ScheduledTileNotification(tileXml, startTime);
scheduledTile.id = "Future_Tile";
var tileUpdater = Windows.UI.Notifications.
                        TileUpdateManager.createTileUpdaterForApplication();
tileUpdater.addToSchedule(scheduledTile);
```

This code creates a start time that is three seconds in the future. After those three seconds have passed, your tile updates.

You can give your notifications an ID so you can later identify them and possibly remove them from the schedule by calling the following:

```
var scheduledTiles = Notifications.TileUpdateManager.
                 createTileUpdaterForApplication().getScheduledTileNotifications();
```

This code returns a list of scheduled notifications that you can inspect. By looping through them, you can inspect properties such as ID, content, and delivery time. You can remove an item from the schedule by calling this:

```
Notifications.TileUpdateManager.
        createTileUpdaterForApplication().removeFromSchedule(item);
```

Periodic notifications

Local and scheduled notifications are based on data your app has locally. Your app can decide to add a notification, either immediately or sometime in the future.

But what if your app doesn’t have the data for an update? If your app is connected to a back end, the back end could keep track of all data and know when an update should be scheduled.

When using *periodic notifications*, you configure your app to look at a URI at a specified interval. That URI is hosted somewhere by you (in Microsoft Azure, for example) and sends the same XML templates you saw before to your app. While testing your notifications, you can use a local HTTP server hosted in Internet Information Services (IIS), for example, that returns your data.

Suppose that your XML template is located at *http://yourdomain.com/tile.xml*. Now you can create code to poll your website and see whether any updates are available:

```
var notifications = Windows.UI.Notifications;
var recurrence = notifications.PeriodicUpdateRecurrence.hour;
var url = new Windows.Foundation.Uri("http://yourdomain.com/tile.xml");
notifications.TileUpdateManager.createTileUpdaterForApplication().
             startPeriodicUpdate(url, recurrence);
```

This code polls your URL once per hour and processes any available updates. You can also specify multiple URIs to poll and create a queue of tile updates that Windows cycles through.

All you have to do is enable queuing the first time the user runs your app:

```
notifications.TileUpdateManager.createTileUpdaterForApplication().
      enableNotificationQueue(true);
```

This code enables the queue. Any updates that you now add to the TileUpdateManager are added to the queue. It works for local, scheduled, polled, and pushed notifications.

By default, the queue follows a first-in, first-out (FIFO) approach: The oldest update is removed when a new update comes in. But maybe that's not what you want. If you have a couple of tile updates that belong to different categories (for example, your user's high score and the overall high score), make sure that you don't get duplicate values (two tiles—one with the old user's high score and one with the new high score).

You can overwrite the FIFO behavior by using tags:

```
tileNotification.tag = "userHighScore";
```

Now when a new update comes in with the same tag, the old update with that tag is removed.

When using the enableNotificationQueue method, the queue is enabled for all tile sizes. You can enable the queue for specific sizes by calling a method that specifies the size, such as enableNotificationQueueForSquare150x150, enableNotificationQueueForWide310x150, and enableNotificationQueueForSquare310x310.

MORE INFO **SCHEDULED NOTIFICATIONS SAMPLE**

The Windows SDK contains a sample that shows you how to schedule notifications at *http://code.msdn.microsoft.com/windowsapps/scheduled-notifications-da477093*.

Push notifications

Poll notifications originate from the client, who regularly checks your web service for updates. The polling request might return with no updates or an update might be already waiting quite some time before the client polls for changes.

Although polling is easy to implement, it's clear why using it isn't an ideal situation. A better approach is to use a *push notification*, in which the server notifies the client when there is a new update, resulting in almost immediate updates and in fewer useless requests, saving bandwidth and resources.

Windows Store apps can use the Notification Client Platform (NCP) to request support for push notifications. The NCP then asks the Windows Push Notification Services (WNS) to create a notification channel. This process returns a URI that you can pass to your web service to establish push communication between your app and your service.

Before you can send notifications, you have to register your app with the Windows Store Dashboard, which gives you a set of credentials that your cloud service can use to authenticate with WNS and make sure it has access to send notifications to your app.

Implementing push notifications requires you to have server-side code to send the actual notifications.

An alternative to using the NCP is SignalR, which is a .NET-based framework with both client and server components that use HTML5 WebSockets to create real-time communication between a client and a server. SignalR is very easy to use from JavaScript and allows you to write real-time apps with relatively ease. SignalR is outside the scope of the exam, but it's very interesting. If you want to know more, you can start at *http://www.asp.net/signalr*.

MORE INFO **PUSH NOTIFICATIONS SAMPLE**

You can find a detailed example showing you how to perform the client-side steps to enable push notifications at *http://code.msdn.microsoft.com/windowsapps/push-and-periodic-de225603/*.

Choosing an appropriate tile update schedule based on app requirements

Now that you have seen the different options for updating your tile and badge, it is easier to choose the correct option for your app.

Table 4-3 lists the different delivery methods that you can use.

TABLE 4-3 Delivery methods and their uses

Delivery Method	Use With	Description
Local	Tile, badge, toast	A set of API calls that sends notifications while your app is running, directly updating the tile or badge, or sending a toast notification
Scheduled	Tile, toast	A set of API calls that schedules a notification in advance to update at the precise time you specify
Periodic	Tile, badge	Notifications that update tiles and badges regularly at a fixed time interval by polling a cloud service for new content
Push	Tile, badge, toast, raw	Notifications sent from a cloud server, even if the app isn't running

Local, scheduled, and push notifications run at their own time. When using periodic notifications, however, you have to specify the time interval for the app to poll the service.

Microsoft clearly states that you should poll no more than once every 30 minutes. If your content is more time-sensitive than 30 minutes, switch to push notifications. It is just as important to remove content that is out of date, especially when your service is unreachable.

You can choose between the following five update frequencies:

- Half hour
- Hour
- Six hours
- Twelve hours
- Daily

You might start by immediately choosing the half hour update mechanism so that your content is always as up to date as possible. However, polling places load both on your client and service, and your client uses battery power each time it polls the service. Your service has to process all requests coming from all devices running your app. This schedule needs to be a balance between resource usage and customer requirements. Think about your specific situation and make an informed decision on what is acceptable for your app.

EXAM TIP

An update frequency of 30 minutes might sound attractive, but is not always the best schedule. Make sure that you understand the requirements posed in the exam question before choosing a frequency.

Thought experiment

Using tiles

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 building your ToDo app and are thinking about possible scenarios for using tiles, badges, and notifications.

List a scenario for each of the following:

1. **A primary small, medium, large, and wide tile**
2. **Secondary tiles**
3. **Notifications**
4. **Badges**

Objective summary

- Users can place a tile that represents your app on their Start screen. Tiles can have different sizes and layouts that you can configure.
- You use the Windows.UI.Notifications namespace to access TileUpdateManager and TileNotification.
- Notifications can be local, scheduled, periodic, or push.
- Secondary tiles can be used to give users direct access to a specific page in your app. You can also specify notifications for secondary apps.
- Badges, which are shown on tiles, can be a number or a glyph giving specific information to the user.
- The schedule you use to update your tiles has to be a balance between resource usage and customer requirements.

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 have created a secondary tile. When users click the tile, they're taken to the Start page of your app instead of to the specific page they linked to. How can you fix this?
   - **A.** Attach a custom event handler to each secondary tile that Windows runs when the user activates the tile.
   - **B.** Listen for the activated event and see whether the ActivationKind is secondaryTile.
   - **C.** When creating a tile, attach a secondaryTile launch event to it.
   - **D.** Listen for the activated event and see whether arguments were supplied.
2. You want to show the number of open tasks together with a task icon on your tiles badge. Which badge template should you use?
   - **A.** None; this is not possible
   - **B.** badgeGlyph
   - **C.** badgeNumber
   - **D.** badgeNumberGlyph

3. Your tile needs to show breaking news. Which delivery method should you use?
   A. Local
   B. Scheduled
   C. Periodic
   D. Push

Objective 4.4: Notify users by using toast

Tiles and notifications are a great way to put you prominently on the user's Start screen, as you learned in the preceding objective. But does that always mean that a user sees your updates and opens your app?

Toast notifications take the whole notification idea one step farther. A toast pops up on the users' screens no matter where they are at that moment: on the Start screen, in another app, or on the desktop.

A *toast* is an invitation to a user to open your app. Toasts are similar to tiles and should be used in combination with tiles and badges. A user can dismiss a toast (or not even see it), so you shouldn't use it for critical functionality.

If used correctly, however, toasts can ensure that users return to your app. This objective shows you what is possible with toast notifications.

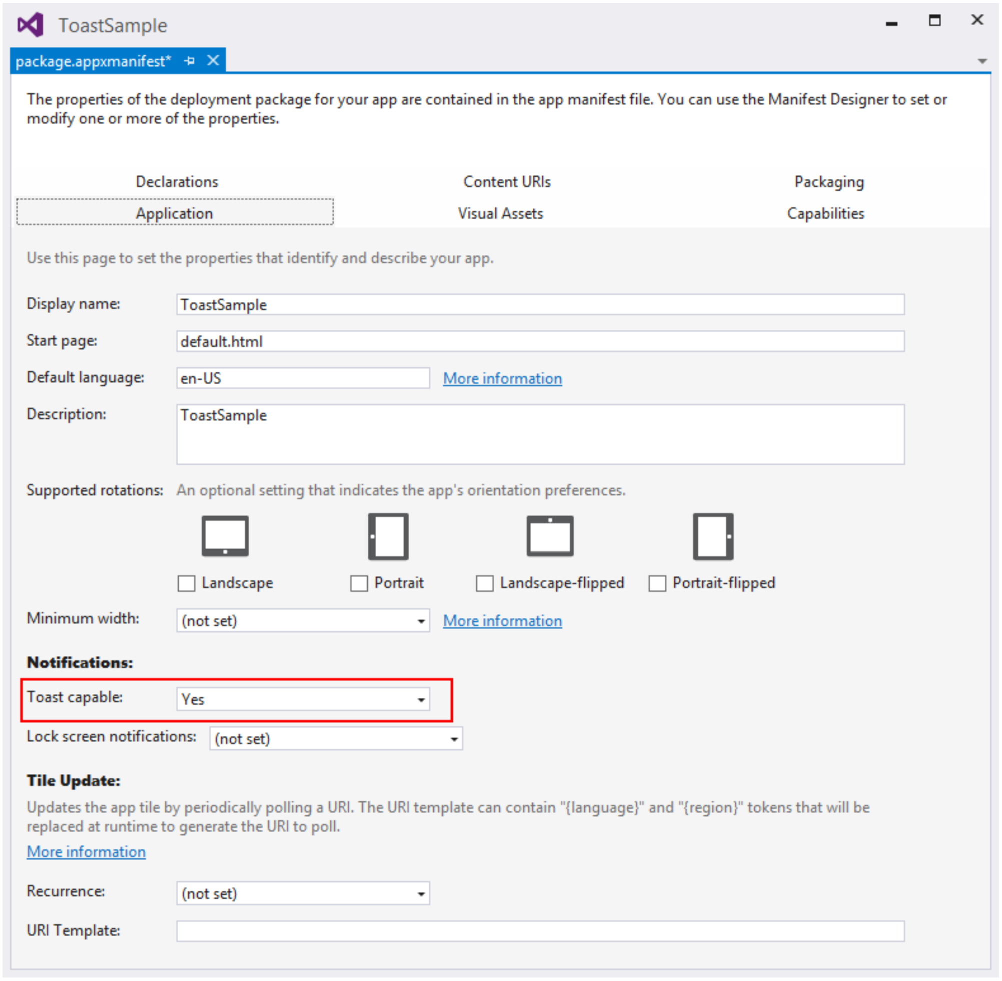
This objective covers how to:

- Enable an app for toast notifications
- Populate toast notifications with images and text by using ToastUpdateManager
- Play sounds with toast notifications
- Respond to toast events
- Control toast duration
- Configure and use Microsoft Azure Mobile Services for push notifications

Enabling an app for toast notifications

Because toast notifications can be shown on the Start screen, desktop, and in other apps, a user should explicitly allow showing toasts. Just as with other integration features that your Windows Store app uses, you enable toasts by using the app manifest.

Figure 4-8 shows the App Manifest Designer with the option to enable toast notifications.



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:	ToastSample
Start page:	default.html
Default language:	en-US
Description:	ToastSample

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: Yes

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.

Recurrence: (not set)

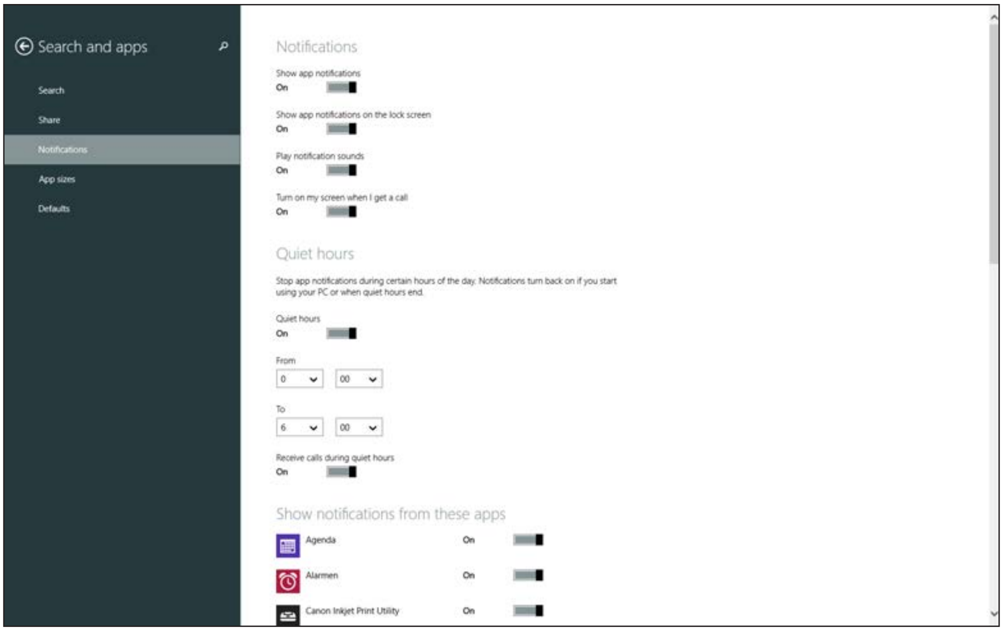
URI Template:

FIGURE 4-8 Enabling toast notifications

When you enable this option, a user can configure your app to allow or disallow the use of toasts by going to Notifications in the PC settings, as shown in Figure 4-9.

Enabling toast capability in your app manifest is the first step of implementing toasts.

You can also configure your app to be shown on the lock screen. When a user places your app on the lock screen, toasts are also shown on the lock screen.



Notifications

Show app notifications
On

Show app notifications on the lock screen
On

Play notification sounds
On

Turn on my screen when I get a call
On

Quiet hours

Stop app notifications during certain hours of the day. Notifications turn back on if you start using your PC or when quiet hours end.

Quiet hours
On

From
0 00

To
6 00

Receive calls during quiet hours
On

Show notifications from these apps

Agenda	On
Alarmen	On
Canon Inkjet Print Utility	On

FIGURE 4-9 The Notifications settings, where a user can configure notifications for your app

Populating toast notifications with images and text by using ToastUpdateManager

Similar to tiles, toasts are based on XML templates that define the layout of the toast. Using the same XML templates in all apps creates a uniform experience for the user and makes it easier to recognize a toast.

Toasts have two types of XML templates:

- Toasts with one or multiple lines of text
- Toasts with a combination of an image and text

As you do with tiles, start by loading the correct XML template:

```
var notifications = Windows.UI.Notifications;
var template = notifications.ToastTemplateType.toastImageAndText01;
var toastXml = notifications.ToastNotificationManager.getTemplateContent(template);
```

This template uses both an image and some text. Now that you have the XML, you can set the text and image:

```
var toastTextElements = toastXml.getElementsByTagName("text");
toastTextElements[0].appendChild(toastXml.createTextNode("Hello World!"));

var toastImageElements = toastXml.getElementsByTagName("image");
toastImageElements[0].setAttribute("src", "ms-appx:///images/logo.scale-100.png");
toastImageElements[0].setAttribute("alt", "logo");
```

As you do with secondary tiles, you have to specify arguments that respond to the user activating your app from the toast. Users expect to open the app in the context of the toast they just saw:

```
toastXml.selectSingleNode("/toast").setAttribute("launch",
                           '{"type":"toast","param1":"12345","param2":"67890"}');
```

Finally, you can show the toast:

```
var toast = new notifications.ToastNotification(toastXml);
var toastNotifier = notifications.ToastNotificationManager.createToastNotifier();
toastNotifier.show(toast);
```

Those are the requirements to show a toast notification with a default duration and no sounds.

This toast generates the following XML:

```
<toast launch="{'type':'toast','param1':'12345','param2':'67890'}">
    <visual>
        <binding template="ToastImageAndText01">
            <image id="1" src="ms-appx:///images/logo.scale-100.png" alt="logo" />
            <text id="1">Hello World!</text>
        </binding>
    </visual>
</toast>
```

MORE INFO **TOAST TEMPLATE CATALOG**

You can find the XML for all the toast templates at *http://msdn.microsoft.com/en-us/library/windows/apps/hh761494.aspx*.

As you do with tiles, you can manipulate the XML directly or use the NotificationsExtensions library from C# to configure your toasts.

Playing sounds with toast notifications

To really get a user's attention, you can play a sound when your toast notification appears. Windows defines a list of sounds that you can use (this list can't be extended). You can use nonlooping sounds such as IM, Mail SMS, or Reminder. You can also use a looping sound, which requires that your toast duration is set to long (see the section "Controlling toast duration," later in this chapter). Looping sounds are an alarm or incoming call.

Short nonlooping sounds signal that something has happened, whereas looping sounds really try to get the users' attention because someone is waiting for the user.

Audio is specified in the XML template as a separate node. By default, the XML templates don't contain the audio element, so you have to add it.

The following code shows an example of attaching an audio tag to a toast XML template:

```
var template = Windows.UI.Notifications.ToastTemplateType.toastImageAndText01;
var toastXml =
    Windows.UI.Notifications.ToastNotificationManager.getTemplateContent(template);
var toastNode = toastXml.selectSingleNode("/toast");

var toastAudioElements = toastXml.getElementsByTagName("audio");
toastAudioElements[0].setAttribute("src", "ms-winsoundevent:Notification.IM");
toastAudioElements[0].setAttribute("loop", "false");
toastNode.appendChild(toastAudioElements);
```

The audio element is created as a completely new element, initialized with a sound and set to nonlooping. You add it to the toast XML so you can configure additional options and create a toast from it.

If you want a looping sound, set the loop attribute to true:

```
toastAudioElements[0].setAttribute("loop", "true");
```

If you want to disable sounds, you can set the silent attribute to true:

```
toastAudioElements[0].setAttribute("silent", "true");
```

Responding to toast events

When a toast appears, a couple of things can happen. First, a user can click your toast to activate your app. A user can also choose to dismiss the toast, or the toast might time out.

You can respond to those events and take appropriate actions. Ensuring that your app can be activated from your toast is the most important step.

When creating your toast, specify the arguments that you want to receive whenever the user activates your app from the toast like this:

```
toastXml.selectSingleNode("/toast").setAttribute("launch",
                             '{"type":"toast","param1":"12345","param2":"67890"}');
```

Inside your activated event, you can check to see whether arguments are present:

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
        var launchString = args.detail.arguments;
        if (launchString)
        {
            var toastArgs = JSON.parse(launchString);
            // Use the arguments to show the correct page to the user
        }

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

If a user doesn't activate your app through the toast, a dismissed event is raised with an argument, telling you why the toast was dismissed. You subscribe to this event on the toast object that's returned from the ToastNotification constructor:

```
var toast = new notifications.ToastNotification(toastXml);
toast.addEventListener("dismissed", function (e) {
     switch (e.reason) {
         case notifications.ToastDismissalReason.applicationHidden:
             break;
         case notifications.ToastDismissalReason.userCanceled:
             break;
         case notifications.ToastDismissalReason.timedOut:
             break;
     }
```

The applicationHidden event is raised whenever your own application explicitly hides a toast notification:

```
toastNotifier.hide(notification);
```

Controlling toast duration

Toast notifications come in two durations:

- The standard toast takes seven seconds and can play a brief sound.
- The long-duration toast takes 25 seconds and can optionally play a looping sound.

Standard toasts, which are the default, quickly grab the attention of your users. If users miss the toast, no harm is done.

Long-duration toasts really get the users' attention. For example, an incoming phone call means that another person is waiting for your user to answer, so you should use a long-duration toast.

To configure a toast as long-duration, set the duration attribute on the toast XML element:

```
var template = notifications.ToastTemplateType.toastImageAndText01;
var toastXml = notifications.ToastNotificationManager.getTemplateContent(template);
var toastNode = toastXml.selectSingleNode("/toast");
toastNode.setAttribute("duration", "long");
```

Configuring and using Microsoft Azure Mobile Services for push notifications

Microsoft Azure Mobile Services is a back end for apps hosted on Microsoft Azure for you by Microsoft. It's easy to get started. You can use Mobile Services for push notifications and as a general back end for your whole app. In this objective, you learn about push notifications; Chapter 5, "Manage security and data," discusses using Mobile Services as a back end.

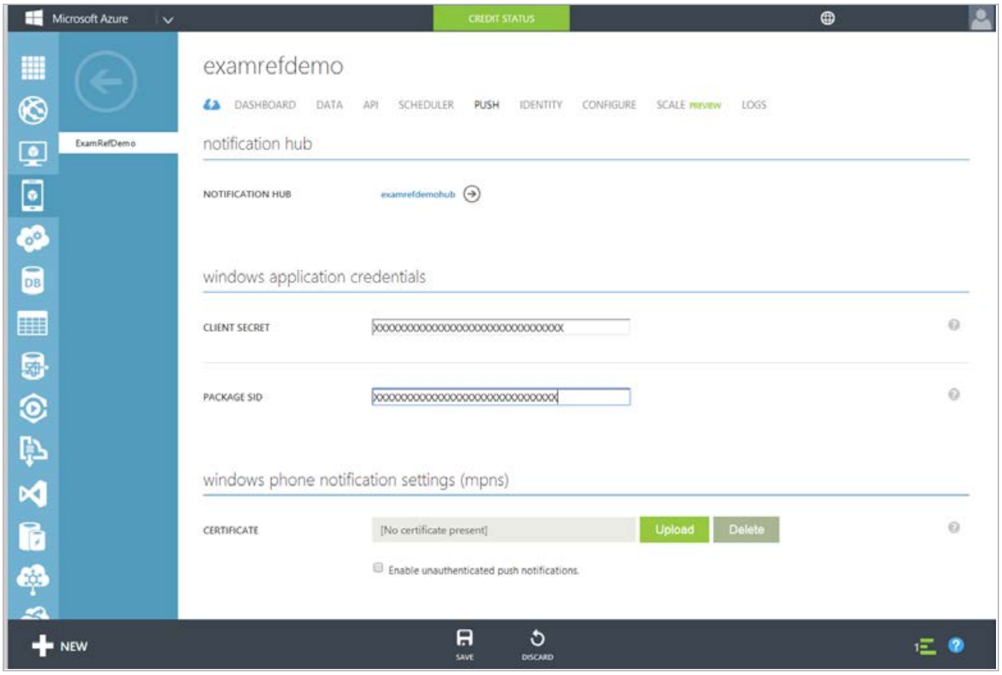
As you saw in Objective 4.3, push notifications require a bit of setup. First, you create a Mobile Service. After registering your app in the store, you can get the required credentials to link your Mobile Service and app for push notifications.

With Mobile Services, you can respond to user actions and run a script (at the time of this writing, it can be JavaScript or C#, although the latter is only in preview). This script can then send a push notification.

EXAM TIP

Microsoft Azure Mobile Services was added as an exam requirement for the new Windows 8.1 exam as of November 2013. Make sure you get some hands-on experience with Mobile Services before taking the exam.

Creating a Mobile Service is easy. If you don't have an Azure account, you can create a free trial to experiment. Figure 4-10 shows a newly created Azure Mobile Service with the application credentials set to the credentials of your app (you can get these credentials from the Windows Dev Center).



examrefdemo

notification hub

NOTIFICATION HUB examrefdemohub

windows application credentials

CLIENT SECRET xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

PACKAGE SID xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

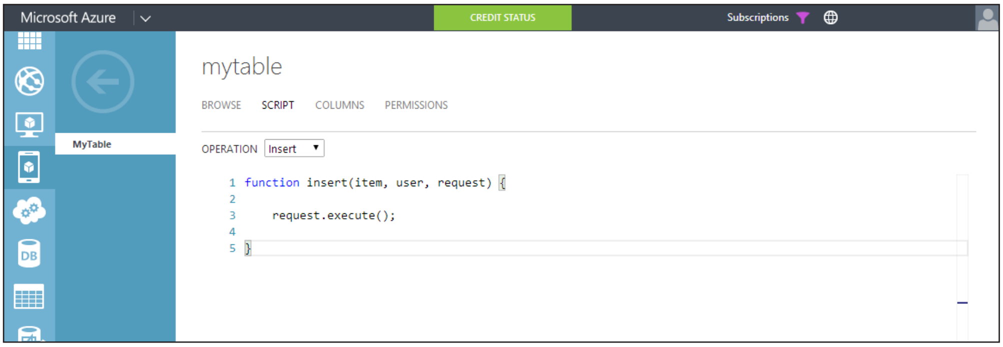
windows phone notification settings (mpns)

CERTIFICATE [No certificate present]

☐ Enable unauthenticated push notifications.

FIGURE 4-10 Microsoft Azure Mobile Services configured for push notifications on a Windows Store app

To send out push notifications, you can add some script to the Mobile Service to respond to the actions. Those scripts can be added through the Azure Management Portal. When selecting your mobile service you can navigate to the specific table and then add scripts that run whenever data is modified in the table. You can see this in Figure 4-11.



OPERATION Insert

```
function insert(item, user, request) {

    request.execute();

}
```

FIGURE 4-11 The Microsoft Azure portal showing the Script section on a Mobile Services table

For example, the following script runs whenever a new record is inserted in the Mobile Services table and sends out a push notification:

```
function insert(item, user, request) {
var payload = '<?xml version="1.0" encoding="utf-8"?><toast><visual>' +
    '<binding template="ToastText01">  <text id="1">' +
    item.text + '</text></binding></visual></toast>';

request.execute({
    success: function() {
        push.wns.send(null,payload, 'wns/toast', {
            success: function(pushResponse) {
                console.log("Sent push:", pushResponse);
                request.respond();
                },
                error: function (pushResponse) {
                    console.log("Error Sending push:", pushResponse);
                    request.respond(500, { error: pushResponse });
                    }
                });
            }
        });
}
```

That's all you have to do for the server side of your app. For the client side, you have to connect your app to the mobile service and open a channel:

```
var client = new WindowsAzure.MobileServiceClient(
    "https://myservice.azure-mobile.net/",
    "secret key"
);
```

```
Windows.Networking.PushNotifications
    .PushNotificationChannelManager
    .createPushNotificationChannelForApplicationAsync()
    .then(function (channel) {
        client.push.registerNative(channel.uri);
    }, function (error) {
        var message = "Registration failed: " + error.message;
        var dialog = new Windows.UI.Popups.MessageDialog(message);
        dialog.showAsync();
    });
```

That's all you have to do. When the app is first launched, it opens the channel and registers itself for toast notifications.

MORE INFO **STEP-BY-STEP ENABLING PUSH NOTIFICATIONS SAMPLE**

The Microsoft Azure documentation contains a walkthrough on how to get started with push notifications in Mobile Services at *http://www.windowsazure.com/en-us/documentation/articles/mobile-services-javascript-backend-windows-store-javascript-get-started-push/*.

Thought experiment

Using some toast

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 working on a ToDo app in which multiple people will collaborate on lists of tasks that they can finish or assign to someone else. Describe the scenarios in which you would use toast notifications for this app.

Objective summary

- Toast notifications need to be enabled in the app manifest.
- Toasts are based on XML templates that you fill with data. ToastNotificationManager is used to get those XML templates and send out toast notifications.
- You can use short nonlooping sounds and looping sounds with your toast notification.
- You can attach arguments to your toasts that you can parse in the activated event of your app. You can also listen for the dismissed event when the user doesn't click your toast.
- A toast notification can have a standard duration of 7 seconds or a longer duration of 25 seconds.
- Microsoft Azure Mobile Services are great back end for an app and feature built-in capabilities for push notifications.

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 showing toast notifications to the user, but you have trouble determining which toast a user clicks when your app is launched. What should you do?
   - **A.** Attach a launch attribute to your toast XML with a string containing your arguments.
   - **B.** Give the toast a unique ID and retrieve the ID when your app is launched.
   - **C.** Attach a unique launch event to each toast you create.
   - **D.** Use the NotificationsExtensions library to have more control over your toast.
2. You want to play a looping alarm sound with your toast, and you already have an audio element called toastAudioElements. Which steps should you take to configure your sound? (Choose all that apply.)
   - **A.** toastAudioElements[0].setAttribute("src", "ms-winsoundevent:Notification.Reminder");
   - **B.** toastAudioElements[0].setAttribute("loop", "true");
   - **C.** toastAudioElements[0].setAttribute("src", "ms-winsoundevent:Notification.Looping.Alarm");
   - **D.** toastNode.setAttribute("duration", "long");
3. For which purpose(s) can you use toast notifications? (Choose all that apply).
   - **A.** To inform users of an upcoming appointment in a calendar app
   - **B.** To inform users that their login credentials are invalid and your chat app can't connect
   - **C.** To inform users that contacts came online while they used your chat application
   - **D.** To let users listen to a preview of an audio file that you distribute through your app

Answers

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

Objective 4.1: Thought experiment

1. You can expect touch, keyboard, mouse, and stylus input.
2. Normally this isn't required. When you want different actions to happen when a user uses another type of input device, check for the type of input device and adjust your actions accordingly.

Objective 4.1: Review

1. **Correct answers:** A, C, D, E, F, G
   - **A.** **Correct:** Tapping is a quick touch and lift.
   - **B.** **Incorrect:** Double-tapping or clicking is no longer part of the standard gestures in Windows 8. Double-tapping was available in the early Windows 8 releases, but was removed in the final release.
   - **C.** **Correct:** Press and hold is a standard gesture in which a finger touches the screen and stays on it for a time period longer than a certain threshold (to differentiate it from tapping).
   - **D.** **Correct:** Slide is moving in one direction with one or more fingers.
   - **E.** **Correct:** Swiping is like a slide, but on a short distance.
   - **F.** **Correct:** Turning or rotating is a standard gesture with two or more fingers in which they move in a clockwise or counterclockwise direction.
   - **G.** **Correct:** Pinching is the standard gesture used for zoom line operations.
2. **Correct answer:** D
   - **A.** **Incorrect:** The click event is raised by WinJS when it translates pointer events into HTML-specific events. It doesn't fire gesture events.
   - **B.** **Incorrect:** Gesture recognizers are used to create custom gestures.
   - **C.** **Incorrect:** The MSGestureChange event is a different type of gesture event. It doesn't fire as long as you haven't hooked up the pointer events.
   - **D.** **Correct:** By listening for pointer events, you can then pass them on to a gesture object that raises gesture events in return.

3. **Correct answers:** A, B

    A. **Correct:** The itemdragstart event is used to configure the data you want to transfer.

    B. **Correct:** The itemdragend event is fired on the drag source element when the drag-and-drop operation ends.

    C. **Incorrect:** The dragover event is fired on the target element while the item is being dragged over the element.

    D. **Incorrect:** The dragenter event fires when an element that's being dragged enters the target element.

    E. **Incorrect:** The dragleave event fires when an element that's being dragged leaves the target element.

    F. **Incorrect:** The drop event is the final event fired in a drag-and-drop operation on the target element.

Objective 4.2: Thought experiment

1. Single-page architecture enables fast navigation between pages. It also enables you to keep state around without having to save and restore state on each navigation.
2. WinJS.Navigation keeps track of history and fires navigation events, WinJS.Page defines separate pages by URI, and PageControlNavigator links the two together by listening to navigation events and showing the correct WinJS.Page objects.
3. Because semantic zoom allows you to quickly switch between two views of data, you might call it a form of navigation. You use it to show different views of the same data to a user.

Objective 4.2: Review

1. **Correct answer:** A

    A. **Correct:** WinJS.Navigation has a back method that updates the history and fires the correct events.

    B. **Incorrect:** PageControlNavigator responds to events raised by WinJS.Navigation. It is not used to initiate navigation.

    C. **Incorrect:** WinJS.UI.Pages is used to define new page objects.

    D. **Incorrect:** BackButton is a WinJS control that listens for events raised by WinJS.Navigation. It shows a back button when the WinJS.Navigation object has a history of URIs. You shouldn't use it to navigate back programmatically.

2. **Correct answers:** A, B, D
   - **A. Correct:** The SemanticZoom control is essential for implementing semantic zoom.
   - **B. Correct:** The two ListViews should be placed in the SemanticZoom control: one for the zoomed-in view; one for the zoomed-out view.
   - **C. Incorrect:** IZoomableView is already implemented by ListView. You don't have to implement it yourself.
   - **D. Correct:** To create two different views of your data, group the data so you can view headers, groups, and items.
3. **Correct answer:** C
   - **A. Incorrect:** Within the single-page architecture, you don't use two different starting pages.
   - **B. Incorrect:** Loading HTML fragments yourself still loads all CSS and JavaScript, so it doesn't resolve the conflicts.
   - **C. Correct:** By scoping the CSS rules, you can avoid conflicts.
   - **D. Incorrect:** Although technically possible, it isn't the best solution. It complicates code and decreases performance.

Objective 4.3: Thought experiment

1. The small tile can show your the logo of your app, but it can't show any notifications. Medium can show notifications in various templates. Wide and large support the most templates.
2. A secondary tile can deep-link to a specific task or task list and show status updates.
3. Notifications can be used for showing new or open tasks or task updates.
4. Badges can show the number of new or open tasks or a glyph showing that new content has arrived.

Objective 4.3: Review

1. **Correct answer:** D
   - **A. Incorrect:** This is not possible; check the activated event for any specific arguments.
   - **B. Incorrect:** The ActivationKind secondaryTile does not exist.
   - **C. Incorrect:** You can use the activated event. There is no specific event for secondary tile launches.
   - **D. Correct:** By checking for arguments in your activated handler, you know whether the user selected a secondary tile.

2. **Correct answer:** A
    - **A.** **Correct:** Badges don't support a combination of glyphs and numbers.
    - **B.** **Incorrect:** This shows only a glyph, not a number and a glyph.
    - **C.** **Incorrect:** This shows only a number, not a glyph.
    - **D.** **Incorrect:** The combination of both a glyph and a number is not allowed in a badge.
3. **Correct answer:** D
    - **A.** **Incorrect:** Local updates are generated by your app when it runs. If the user isn't using your app, the tile doesn't get updated with news.
    - **B.** **Incorrect:** With a scheduled update, you specify a local update to occur at some time in the future.
    - **C.** **Incorrect:** Polling the service for updates happens at 30-minutes intervals, which is not suited for breaking news.
    - **D.** **Correct:** Push notifications are very fast and leave the service in control of the update.

Objective 4.4: Thought experiment

Toast notifications can show users that a task was finished or a new task is assigned to them. They help users stay up to date without having to open your app.

Objective 4.4: Review

1. **Correct answer:** A
    - **A.** **Correct:** The launch attribute on your toast XML is passed to your activated event as part of the arguments.
    - **B.** **Incorrect:** A toast can't have a unique ID that you then map to the specific context in which it was created. Instead, you can attach data to the toast XML.
    - **C.** **Incorrect:** You can't attach a unique handler to each toast for activation.
    - **D.** **Incorrect:** The NotificationsExtensions library is a C# wrapper around the XML templates. It doesn't offer any extra features besides those of direct XML manipulation; it is just easier to use.
2. **Correct answers:** B, C, D
    - **A.** **Incorrect:** The Reminder sound is a short sound that can't be looped.
    - **B.** **Correct:** You need to configure the audio to loop.
    - **C.** **Correct:** The Alarm sound can be looped.
    - **D.** **Correct:** Looping can be done only on a toast with a long duration.

3. **Correct answer:** A
   - **A. Correct:** An upcoming appointment is a perfect scenario for a toast notification.
   - **B. Incorrect:** Errors such as expired credentials shouldn't be shown in a toast that a user can easily miss. You should show them in-app.
   - **C. Incorrect:** While inside your application, you shouldn't use any toasts.
   - **D. Incorrect:** Using toasts to preview audio is not possible; you can use only Windows audio sounds.

This page intentionally left blank

CHAPTER 5

Manage security and data

Data is an essential part of a lot of apps. Maybe your app stores data locally, roams it to other devices, or stores it on a back end such as Azure Mobile Services. Choosing the correct data strategy is an important part of the development of your app. And when you know where your data is stored, you need ways to access your data. This objective focuses on choosing the correct data access strategy, retrieving data, and working with the data within your UI.

The second part of this objective is about security. Making sure users can authenticate with your app and that you can manage access to sensitive data is important for most apps.

This chapter focuses on content that comprises 20 to 25 percent of your exam. Choosing the correct data access strategy is not an exact science. Make sure you understand the different options and understand when they can be useful. Try to experiment with the different security options and make sure you know how to use CredentialPicker and PasswordVault.

Objectives in this chapter:

- Objective 5.1: Choose a data access strategy
- Objective 5.2: Retrieve data remotely
- Objective 5.3: Implement data binding
- Objective 5.4: Manage Windows authentication and authorization
- Objective 5.5: Manage web authentication

Objective 5.1: Choose a data access strategy

The data your app uses has to be stored somewhere, so Windows Store apps offer you options for storing data both locally and remotely. Depending on your app requirements, choose a data access strategy that uses one or a combination of the different data storage options you have.

This objective discusses the different options and scenarios.

This objective covers how to:

- Choose the appropriate data access strategy (file based; web service; remote storage, including Microsoft Azure storage and Azure Mobile Services) based on requirements

Choosing the appropriate data access strategy based on requirements

When deciding on a data access strategy, you need to store data somewhere. In essence, you have to decide whether to store data locally or remotely. Both options have advantages and disadvantages that you need to be aware of, such as performance and security. The following sections will discuss both options to help you choose the correct option for your scenario.

Local data

Local data, such as images or other data files that are distributed with your app and are always present on a user's device can be stored in your app package. Local data, which is well-suited for static data that should be loaded relatively quickly, can be loaded by using the ms-appdata:// protocol that you've seen in previous samples.

Local data is stored in three folders:

- LocalFolder
- RoamingFolder
- TemporaryFolder

The local folder should be used for any data that you want to be preserved between app sessions. Data stored in memory will be lost when the app closes. Local data will be preserved on the device and can be loaded when needed. The local folder should be used whenever you have data that isn't not portable to other devices or that is too big to sync between devices.

The temporary folder is managed by Windows and can be cleaned up at any time by both the system or by a user explicitly running a cleanup action. Because of this you can't be certain if the data you saved in here is still available the next time you want to load it. The temporary folder can be used as a cache.

The roaming folder is for data that should be automatically synced between a user's different devices, which is very useful for storing data locally and making it accessible on multiple devices. A roaming folder can't be shared between different users.

Roaming data does not happen instantly. The user's device checks factors such as available bandwidth and user activity to determine when it should roam data. The size of roaming data is also limited and should be kept below the quota found in the RoamingStorageQuota property.

The following code shows how to write some data to the roaming folder:

```
var roamingFolder = Windows.Storage.ApplicationData.current.roamingFolder;
var filename = "sampleFile.txt";
roamingFolder.createFileAsync(filename,
                              Windows.Storage.CreationCollisionOption.replaceExisting)
    .done(function (file) {
        return Windows.Storage.FileIO.writeTextAsync(file, "file content");
    });
```

The createFileAsync method takes a filename and a *CreateCollisionOption* parameter. The CreateCollectionOption parameter is an enum with one of the following values:

- **GenerateUniqueName** Creates the new file or folder with the desired name, or automatically appends a number if a file or folder already exists with that name
- **ReplaceExisting** Creates the new file or folder with the desired name, and replaces any file or folder that already exists with that name
- **FailIfExists** Creates the new file or folder with the desired name, or returns an error if a file or folder already exists with that name
- **OpenIfExists** Creates the new file or folder with the desired name, or returns an existing item if a file or folder already exists with that name

In the previous example, replaceExisting is used. This option makes sure that no error occurs if the file already exists. It also lets you start with a clean, empty file if the file does exist.

File-based actions are always asynchronous because you don't want to block the app. In this case, you first create the file and then write some text to it. By using the writeBytesAsync method, you can also write some binary data to a file.

Reading data from your roaming folder is easy:

```
roamingFolder.getFileAsync(filename)
            .then(function (file) {
                return Windows.Storage.FileIO.readTextAsync(file);
            }).done(function (text) {
                // use the text in your app
});
```

Similar to writing to a file, you first receive a reference to the file and then read the content as text. Opening the file and readings its content is also done asynchronously, but it is easy to write the code using promises.

The following example demonstrates how to couple different input/output operations together. The sample reads a binary file (an image) and writes it to the roaming folder:

```
var roamingFolder = Windows.Storage.ApplicationData.current.roamingFolder;
var imageFilename = new Windows.Foundation.Uri("ms-appx:///images/logo.scale-100.png");
var outputFilename = "output.dat";
```

```
roamingFolder.createFileAsync(outputFilename,
                              Windows.Storage.CreationCollisionOption.replaceExisting)
    .done(function (outputFile) {
        return Windows.Storage.StorageFile.getFileFromApplicationUriAsync(imageFilena
me).done(function (inputFile) {
            return Windows.Storage.FileIO.readBufferAsync(inputFile).done(function
(buffer) {

                var bytes = new Uint8Array(buffer.length);
                var dataReader = Windows.Storage.Streams.DataReader.fromBuffer(buffer);
                dataReader.readBytes(bytes);
                dataReader.close();

                outputDiv.innerText = outputFile.path;

                return Windows.Storage.FileIO.writeBytesAsync(outputFile, bytes);
            });
        });
    });
```

The final promise receives a buffer of data and then converts that data to an array that can be stored in the output file. All methods return a promise to make sure that the code runs asynchronously and doesn't block the UI.

So which scenarios should you choose for storing data in a folder? The temporary folder should be used only for data that can be removed after you finish with it. Windows checks to see whether the file is still in use and deletes it after you're done.

The local folder can be used to store persistent data that is applicable only to the user's current device. Because the data is local, you can store files that are larger than those in the roaming folder.

Roaming data is a very popular option. By storing data in this folder, you give users a consistent experience across different devices. Roaming data is stored in the cloud (Microsoft Azure) and is synced to all the devices of a user where your app is installed. Roaming saves users from having to set up each app to their specifications and enables them to work on the same data set on whichever device they use.

When storing data in a folder, remember that the lifetime of the data is bound to the lifetime of the app. When you store data in a local folder, the data is removed when the user removes the app. Roaming data is kept around in the cloud for a specific time interval (30 days at the time of this writing). So don't store data in a folder if it is valuable to the user and needs to be easily accessed.

Because storing data in a local or roaming folder is easy to do and saves you from creating your own custom back end, it's a very viable option for many apps. Even if not all data is suited to be stored in a folder, you can always choose a hybrid model in which you offload some data to a cloud solution and other data to the folders in your app.

EXAM TIP

Make sure you understand the difference between the local folder and the roaming folder.

Remote data

Although they can be a good data storage solution, folders can't fulfill the requirements of more complex apps. Apps that allow multiple users to work together, or apps that are a front end to some centralized data that can be accessed through a website or through other apps on other platforms, need a remote way to store data.

Microsoft Azure is a perfect solution for your apps' back end. Azure is a Microsoft cloud solution that offers you a wide variety of services. For apps, the huge benefit of Azure is that you don't have to invest in buying servers or infrastructure upfront. When you release your app, you probably don't know how much income you will generate, so buying hardware upfront is difficult. If your app suddenly becomes very popular, Azure is also there to help. By giving you the option to automatically scale your back end to the number of users and pay only for what you use, Azure gives you a very cost-efficient solution that can handle any number of users.

The Azure services that are the most interesting when it comes to apps are *Azure Mobile Services* and *Azure storage*.

Azure storage offers complex features such as tables and queues, which are out of scope for this exam. Azure block blob storage can be seen as an enormous hard drive in the cloud, in which you can store data that can be securely delivered to clients. When it comes to storing large data files such as videos, Azure block blob storage is a perfect solution.

The following code shows how to upload a file to Azure blob storage. In this case, the credentials of the Azure Storage Account are removed from the listing. You need to add your own credentials if you want to test this example.

The example consists of two parts: one in C#, the other in JavaScript. The C# code is used to create what's known as a Shared Access Signature. This is a URL that can be used from your JavaScript to access the Blob storage. The URL contains access rights and an expiration date.

The C# code is as follows:

```
private async Task<string> _GetSaS()
{
  var storageAccountString = String.
Format("DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}",

accountName, accountKey);
    var storageAccount = CloudStorageAccount.Parse(storageAccountString);
    var client = storageAccount.CreateCloudBlobClient();

    CloudBlobContainer container = client.GetContainerReference("mycontainer");
    await container.CreateIfNotExistsAsync();
    var blob = container.GetBlockBlobReference("myimage.png");

    var sas = blob.GetSharedAccessSignature(
        new SharedAccessBlobPolicy()
        {
            Permissions = SharedAccessBlobPermissions.Write |
SharedAccessBlobPermissions.Read | SharedAccessBlobPermissions.List,
            SharedAccessExpiryTime = DateTime.UtcNow.AddMinutes(30),
        });

    return string.Format(CultureInfo.InvariantCulture, "{0}{1}", blob.Uri, sas);
}
```

In this case, a Shared Access Signature is created that lasts for 30 minutes and allows the user to write and read data. The container name, mycontainer, is created in Azure Storage with a Block Block named myimage.png.

The JavaScript code calls this C# code and then uploads a file:

```
function uploadData() {
    var ajaxRequest = new XMLHttpRequest();

    var imageFilename = new Windows.Foundation.Uri("ms-appx:///images/logo.scale-100.
png");

    Windows.Storage.StorageFile.getFileFromApplicationUriAsync(imageFilename).
done(function (inputFile) {
        return Windows.Storage.FileIO.readBufferAsync(inputFile).done(function (buffer)
{

            var bytes = new Uint8Array(buffer.length);
            var dataReader = Windows.Storage.Streams.DataReader.fromBuffer(buffer);
            dataReader.readBytes(bytes);
            dataReader.close();

            new CORSSupport.AzureCommon().getSas().done(function (url) {
                try {
                    ajaxRequest.open('PUT', url, true);
                    ajaxRequest.setRequestHeader('Content-Type', 'image/jpeg');
                    ajaxRequest.setRequestHeader('x-ms-blob-type', 'BlockBlob');
                    ajaxRequest.send(bytes);
                }
                catch (e) {
                    outputDiv.innerText = "can't upload the image to server.\n" +
e.toString();
                }
            });
        })
    });
}
```

This code reads an image file from the package, converts it to a byte array, and then uploads the data to Azure by sending a put request.

You can use authentication on your blob storage or you can create public blobs that can be referenced by anyone. Azure blob storage is primarily about storing data and retrieving it from all over the world.

If you need a back end that contains some actual logic, Azure Mobile Services can be a solution. (Chapter 4 discussed using Azure Mobile Services for sending push notifications to your app, and it can be used for toast or tile updates.)

You can also use Azure Mobile Services to store the data for your app. It automatically creates a table schema for your data and enables you to perform Create, Read, Update, Delete (CRUD) operations on your data. By writing custom JavaScript or C#, you can add code to your back end that extends to those CRUD operations. You can even add completely custom operations to your back end and then use other Azure features such as blob storage from within your Mobile Services code.

The beauty of the cloud is that you pay only for what you use. Mobile Services is completely managed for you by Azure, and you can configure it to scale when the user load increases. You then have a back end to handle the load if your app suddenly becomes very popular.

After you create a Mobile Service, it is easy to connect to it from JavaScript. The following code sample shows how to add a new ToDo item to a Mobile Services back end:

```
var todolistClient = new WindowsAzure.MobileServiceClient(
                "https://todolist.azure-mobile.net/",
                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
var item = { description: {'todo item'}};
var todoTable = todolistClient.getTable('TodoItem');
    todoTable.insert(todoItem).done(function (item) {
        // item is added
    });
};
```

MORE INFO **CREATING A MOBILE SERVICE**

For more information on how to use the Azure portal to create a Mobile Service, see *http://azure.microsoft.com/en-us/documentation/articles/mobile-services-windows-store-javascript-get-started-data*.

A complete discussion of how Azure works is outside the scope of this book and the exam. What's required is a basic understanding of what Azure offers with its Mobile Services and storage. Understanding that Mobile Services is a perfect way to create a back end for your app and knowing its possibilities will help you during the exam.

You should also understand the differences between storing data locally (in a roaming folder) or remotely.

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 considering the Weather app that is installed by Windows. Although you don't know how the app is implemented, can you determine what data is stored locally and which is stored remotely?

Objective summary

- Local data is stored in folders, and you have a local folder that is available only on the device in which it is created. A temporary folder can also be used; its data is removed when an app no longer uses it. Roaming data is automatically synced between devices on which users have your app, enabling you to create a seamless experience when a user moves from one device to another.
- You can remotely store data, such as videos and images that you store in Azure storage. You can also create a back end by using Azure Mobile Services, which enables you to store data and create custom actions that your app can use.

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 allow users to select large video files in your app and add custom effects to them. Users expect that video files will sync to all their devices. Where do you store the data?
   - **A.** Roaming folder
   - **B.** Table in Azure Mobile Services
   - **C.** Local folder
   - **D.** Azure blob storage

2. Your app allows users to add text and other effects as metadata to an image they select and then share this image with other users so they can rate it. Which features do you use? (Choose all that apply.)

   A. Table in Azure Mobile Services

   B. Azure blob storage

   C. Roaming folder

   D. Local folder

3. Why is it important to always use asynchronous code when using I/O operations like storage or web service calls?

   A. It ensures that the external I/O call finishes more quickly.

   B. It ensures that your app stays responsive while waiting for the I/O operation.

   C. It allows the web service or I/O device to do other work while your app is waiting for the response.

   D. It ensures that the user can still use the Internet or I/O device for other applications while your app is sending a request.

Objective 5.2: Retrieve data remotely

When you store your data remotely, it's important to know how to get to that data. The Windows Library for JavaScript (WinJS) offers features that you can use to call external web services. However, depending on the communication style used by those web services, ensure that you connect to them in the correct way.

This objective discusses the options you have for remote connectivity and how to use them optimally.

This objective covers how to:

- Use XHR or HttpClient to retrieve web services
- Set appropriate HTTP verb for REST
- Handle progress of data requests
- Consume SOAP/WCF services
- Use WebSockets for bidirectional communication

Using XHR or HttpClient to retrieve web services

Your Windows Store app with HTML, JavaScript, and Cascading Style Sheets (CSS) runs in Internet Explorer, so you can use the features that Internet Explorer offers when you access remote web services.

In regular web applications, you often use a technology called Asynchronous JavaScript and XML (AJAX) when loading data from a web service. Browsers implement support for making asynchronous web service calls by using an object called XmlHttpRequest. This object allows you to process a web service call from JavaScript and process the result when it returns. In web applications, XmlHttpRequest can be used to load some data and update a part of the page without doing a full page load.

You can still use the XmlHttpRequest object with Windows Store apps, but that is a very rudimentary way of working with AJAX requests. To help you, Microsoft implemented WinJS.xhr and later released HttpClient, which is based on WinJS.xhr.

The following code shows how to process a call with WinJS.xhr:

```
WinJS.xhr({ url: "http://www.microsoft.com" }).done(
    function complete(result) {
          }
      );
```

This code processes an asynchronous call to *www.microsoft.com* and gets an object of type XMLHttpRequest as a result.

Although this process might look simple, sending requests becomes more difficult when you start adding headers, cookies, or other custom settings. WinJS.xhr doesn't have any strong typing, so all values that you pass to it are plain text. The same is true for retrieving data: You know that the lastModified header contains a date, but WinJS returns it as a plain string—leaving all the parsing up to you.

Fortunately, Microsoft introduced a new library that makes handling remote web service calls much easier: HttpClient. HttpClient was introduced for all Windows Store platforms: JavaScript, C# and C++.

EXAM TIP

For the exam, you need to understand both WinJS.xhr and HttpClient. However, for real-world scenarios, you should always choose HttpClient. HttpClient is more modern and has more functionality in Windows Store apps.

In JavaScript, HttpClient is essentially a strong wrapper around WinJS.xhr; in a modern application, you should use HttpClient for the best programming experience. The previous sample code of retrieving the Microsoft website with WinJS.xhr can be changed to the following to use HttpClient:

```
var hc = new Windows.Web.Http.HttpClient();
var uri = new Windows.Foundation.Uri("http://www.microsoft.com");
hc.defaultRequestHeaders.userAgent.parseAdd("ie");
hc.getStringAsync(uri).done(
    function complete(result) {
});
```

Start by instantiating a new HttpClient object. Instead of using a string-based URL, construct a URI object. This code also adds a header value setting the user agent to Internet Explorer. This avoids a response from the server stating that your app is an automated process and that you can't access the site. You can then use the getStringAsync method of HttpClient to retrieve the Microsoft website directly as a string.

Although this basic use of HttpClient doesn't differ much from the WinJS.xhr method, some differences become apparent when you start using more complex functionality.

If you use HttpClient.getStringAsync, you directly retrieve the result as a string. Calling getStringAsync is a shorthand way to use GetAsync, which returns an object of type HttpResponseMessage.

The HttpResponseMessage object has several important properties:

- **Content** Gets or sets the content of the HTTP response message on the HttpResponseMessage object
- **Headers** Gets the collection of HTTP response headers associated with the HttpResponseMessage that was sent by the server
- **IsSuccessStatusCode** Gets a value that indicates whether the HTTP response was successful
- **ReasonPhrase** Gets or sets the reason phrase, which is typically sent with the status code by servers
- **RequestMessage** Gets or sets the request message that led to this response message
- **Source** Gets the source of the data received in HttpResponseMessage
- **StatusCode** Gets or sets the status code of the HTTP response
- **Version** Gets or sets the HTTP protocol version used on the HttpResponseMessage object

The content property is one that you will often use to get the result of an HTTP request. Content can be a buffer, string, stream, or name/value data. You can even define custom content.

The following code snippet uses HttpClient to process a Get request and parse the content as XML:

```
var uri = new Windows.Foundation.Uri("http://www.microsoft.com/");

var hc = new Windows.Web.Http.HttpClient();
hc.defaultRequestHeaders.userAgent.parseAdd("ie");
var httpPromise = hc .getAsync(uri)
    .then(function (response) {
        response.ensureSuccessStatusCode();
        return response.content.readAsStringAsync();
    }).then(function (responseBodyAsText) {
        var parser = new window.DOMParser();
        var xml = parser.parseFromString(responseBodyAsText, "text/xml");
    });

httpPromise.done(function () {
}, function (error) { });
```

This code shows a typical pattern for using HttpClient. First, it's important to call response.ensureSuccessStatusCode() to make sure that your request returned successfully. If this call fails, the error handler in the "done" part of the promise processes.

In this case, it is getAsync, so you get a full HttpResponseMessage in which you can decide how you want to work with the content. Parsing it as a string and then funneling it through the DOMParser as XML is one option. Depending on your requirements, you can use the data any way you want.

Adding cookies to a request is a common scenario. HttpClient supports it because of its extensible pipeline. A request flows through several filters that comprise the pipeline of your request. HttpBaseProtocolFilter is the base filter that you can add to a request. You can use this object to retrieve a CookieManager to work with your app's cookies:

```
var bpf = new Windows.Web.Http.Filters.HttpBaseProtocolFilter();
var cookieManager = bpf.cookieManager;
var cookie = new Windows.Web.Http.HttpCookie("myCookieName", ".mydomain.com", "/");
cookie.Value = "myValue";
cookieManager.setCookie(cookie);
var httpClient = new Windows.Web.Http.HttpClient(bpf);
```

This example adds a cookie for your domain with a custom value. The filter is then added to HttpClient and is used in all requests processed through this HttpClient instance.

You can create custom filters and add them to the pipeline. The filters have to be written in C++ and are outside the scope of the exam. However, when you find yourself executing the same code over and over on a request, think of filters and how they can help you centralize code. Microsoft made a sample available in which it has created a retry filter that automatically retries the request whenever a 503 status code is returned from the server.

MORE INFO **HTTPCLIENT SAMPLE**

The HttpClient sample shows how to implement a customer filter, but also demonstrates other uses of HttpClient. You can find it at *http://code.msdn.microsoft.com/windowsapps/HttpClient-sample-55700664*.

Until now, the discussion has been how to process a Get request for data. The following sections discuss what other options you have and how they map to HttpClient.

Setting appropriate HTTP verbs for REST

Getting data from a web service is probably one of the most-used web service features in an app. However, with that data you also want to Create, Update, Read, and Delete (CRUD).

HttpClient offers support for working with web services to process these action types. But to understand how it does so, you should know a little more about how HTTP works.

The nature of the HTTP protocol used for the Internet (and thus for web services) is built around URIs that describe resources and HTTP verbs that describe your actions. Services that are based on the HTTP verbs and URIs are called *Representational State Transfer (REST)* services.

For example, just by looking at the URL *http://mydomain.com/people/john*, you can guess that it represents getting some information about people named John. In this case, you are using the Get HTTP verb to retrieve some data.

The available HTTP verbs are listed in Table 5-1.

TABLE 5-1 HTTP verbs

Verb	CRUD Action	Behavior
Post	Create	Inserts a new entity to the URI
Get	Read	Retrieves one or more entities or some other data that is identified by the URI of the request
Put	Update	Replaces an entity that is identified by the URI. This verb requires that all fields on the entity be specified, regardless of how many change
Patch	Update	Transmits a partial change to the entity identified by the URI in which only identifiers and modified fields are specified
Delete	Delete	Specifies that a given URI be deleted
Head	N/A	Retrieves just the message headers identified by the URI
Options	N/A	Represents requests for information about the communication options available on the target resource

The most popular HTTP verbs are *Post, Get, Put,* and *Delete*, which match the typical Create, Read, Update, and Delete (CRUD) actions that you want to perform on your data.

HttpGet

Most query or retrieval operations are implemented as HttpGet actions. The number of methods on a web service that are implemented as HttpGet actions are usually the highest. For any type of data, all the items in the data set that allow for individual retrieval by the key usually have to be retrieved. Table 5-2 shows a basic REST scheme for retrieval centered on a given model (in practice, a web service can expose several different models, so you usually find several different HttpGet methods).

TABLE 5-2 HttpGet retrieval on a fictional Baz object

URI	Action
/api/Bazs	Gets a list of all Bazs
/api/Bazs/keyvalue	Gets and instance of Baz by key field
/api/Bazs?attributename=attributevalue	Gets an instance of Baz by attribute

The HttpClient class supports several methods that you can use for Get operations:

- **GetAsync(Uri)** Sends a Get request to the specified URI as an asynchronous operation.
- **GetAsync(Uri, HttpCompletionOption)** Sends a Get request to the specified URI with an HTTP completion option as an asynchronous operation. The HTTP completion option specifies whether you want the request to finish when the headers are read or when the complete content is read.
- **GetBufferAsync** Sends a Get request to the specified URI and returns the response body as a buffer in an asynchronous operation.
- **GetInputStreamAsync** Sends a Get request to the specified URI and returns the response body as a stream in an asynchronous operation.
- **GetStringAsync** Sends a Get request to the specified URI and returns the response body as a string in an asynchronous operation.

HttpDelete

Arguably the easiest of the HTTP verbs to identify, HttpDelete requests are straightforward to use. The most common way to delete a record is to specify a unique key and use it to identify and delete the record.

A web service typically provides some form of feedback about a Delete request. There are three possible outcomes, assuming that the request is correctly formed and processed:

- The first outcome is a successfully processed request that has an HttpStatusCode of OK (200). A valid response is returned to the client, and information from the request can be included.
- The next outcome is an HttpStatusCode of Accepted (202), which indicates that the request was processed and accepted, but is still pending.
- The last outcome is an HttpStatusCode of No Response (204).

To send a Delete request to your REST service, you can use the DeleteAsync method on your HttpClient. If your service is truly RESTful, you can probably use an URI such as *http://mydomain.com/people/id*, where *id* is the unique ID of the entity that you are trying to delete. Because of the correct HTTP verb, the REST service knows what you are trying to do.

HttpPost

When you want to insert new data, you usually use the HttpPost verb. If any exam question or requirement specifies that a new record be created, it probably necessitates an HttpPost operation.

HttpClient has a PostAsync method that you can use to process a Post request. This method takes both a URI and the content you want to send. You use it like this:

```
var stringContent = new Windows.Web.Http.HttpStringContent("The content to post");
var uri = new Windows.Foundation.Uri("http://mydomain.com/people");
var httpClient = new Windows.Web.Http.HttpClient();
var httpPromise = httpClient.postAsync(uri, stringContent).done(function () {
}, function (error) { });
```

A correctly implemented REST service returns an HttpStatusCode of Created (201) when the insertion of the new data is successful. The response also contains the location of the new resource, which enables you to immediately get the feedback from which you can find out any details about the entity that was just added.

HttpPut

The HttpPut verb is used for operations that correspond to *upserts*, inserting for new records and updating for existing records. A side-effect of upserts is that the method should be *idempotent* (if you call the method once or 100 times with the same data, there should be no meaningful difference in the side-effects of calling it 1 or 100 times).

HttpPut is also supported by the HttpClient with the method PutAsync. This method also takes a URI and the content you want to send.

EXAM TIP

Make sure you understand how the HTTP verbs Get, Put, Delete, and Post map to Create, Read, Update, and Delete. You can expect exam questions that ask you to choose the correct verb based on the requirements.

Handling progress of data requests

Executing web service requests can be one of the most time-consuming actions in your app. Even if the request takes only one second, the user is probably waiting for the request to finish. To make sure that your app is still fast, fluid, and shows the user what's happening, you can implement progress reports for your requests.

The following code shows how to handle progress updates:

```
var uri = new Windows.Foundation.Uri("http://www.microsoft.com");
var httpClient = new Windows.Web.Http.HttpClient();
httpClient.getAsync(uri).done(function (response) {
},
function error(result) {
},
function progress(progress) {
    WinJS.log && WinJS.log("Progress: " + progress.stage);
});
```

This code processes a regular Get request with an extra method for errors and for progress notifications. The progress object passed to your handler contains a property named stage that contains a number that maps to certain stages in your request. The values and their meanings are shown in Table 5-3.

TABLE 5-3 Progress stages

HttpProgress.stage Numeric Value	HttpProgress.stage Meaning
10	Detecting proxy
20	Resolving name
30	Connecting to server
40	Negotiating Secure Sockets Layer (SSL)
50	Sending headers
60	Sending content
70	Waiting for response
80	Receiving headers
90	Receiving content

By mapping these values to something understandable, you can give your users an idea of what's happening in your app.

Consuming SOAP/WCF services

REST services are built on the idea of HTTP verbs and URIs to work with resources. Simple Object Access Protocol (SOAP) services work differently. Instead of accessing one URI with multiple verbs, different URIs directly map to specific actions. So instead of calling *http://mydomain.com/people* with a Get verb, you call *http://mydomain.com/getpeople*. The *getpeople* segment maps to the name of the method defined in the service.

SOAP communication is based on XML. Instead of sending plain JavaScript Object Notation (JSON) to the server, you send what's called a SOAP envelope that contains the data in XML format. An example SOAP message looks something like this:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://mydomain.com/stock">
      <m:StockName>Microsoft</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

This envelope describes that you want to call the GetStockPrice method on the specified URL and that you pass a StockName parameter with a value of Microsoft.

You can construct those envelopes by hand and use HttpClient to send the data. Another option that doesn't require you to deal with XML in JavaScript is to add a Windows Runtime (WinRT) component project and then add a service reference to the Windows Communication Foundation (WCF) service you want to use. By then exposing this service to your JavaScript project, you have an easier way to work with your service. A typical service method in your WinRT component might look like this:

```
public sealed class Service
{
    public  Windows.Foundation.IAsyncOperation<string> GetDataAsync()
    {
        var service = new Service1Client(new BasicHttpBinding(),
            new EndpointAddress("http://localhost:7214/Service1.svc"));

        return service.GetDataAsync().AsAsyncOperation();

    }
```

This method uses the service proxy that Visual Studio can generate for WCF services and exposes it as an AsyncOperation.

MORE INFO **WCF SERVICES**

Creating your own WCF services is outside of the scope of this book. For more information, see *http://msdn.microsoft.com/en-us/library/dd456779.aspx*.

EXAM TIP

Working with WCF can be difficult. Make sure that you understand the conceptual difference between a REST service and a SOAP service. Remember that SOAP services are based on XML envelopes that contain the data while REST is based on a URI scheme with HttpVerbs.

Using WebSockets for bidirectional communication

Real-time communication between your app and a server is often required. For example, you might be developing a multiplayer game or working on another type of app that requires users to work together and see updates in real time.

A common solution to real-time scenarios has been to regularly ask the server about available updates by using a JavaScript timer and sending an AJAX request to the server. In this situation, your app is constantly busy sending requests and processing the responses, but an update might not be available. So there is a load on the server, especially when multiple users are running your app, and most of their update requests are not filled.

This type of communication is also one-directional. The client app asks the server for updates, but the server can't send some data to the client.

Fortunately, technology has improved and there is a new solution: WebSockets, which is a protocol that provides bidirectional communication between client and server. The initial request is created over HTTP, but the request is upgraded to a TCP-based protocol after the handshake.

WebSockets enables a fast bidirectional communication path between an app and a back-end server. They can send each other updates when they become available without having to frequently check to see if there are updates available.

Building the server side of a WebSockets server is outside the scope of this exam. Microsoft provides an example with several scripts that automatically set up a web server so you can test your WebSockets code.

MORE INFO **WEBSOCKETS SAMPLE**

For the complete implementation of all the required methods, see the WebSockets sample at *http://code.msdn.microsoft.com/wpapps/Connecting-with-WebSockets-643b10ab*.

Working with WebSockets in your app means that you first construct a WebSocket and then start listening for incoming messages. You can also use the opened WebSocket to send messages to the server.

Creating a WebSocket is easy:

```
var webSocket = new Windows.Networking.Sockets.MessageWebSocket();
```

You have to start listening for incoming messages or for the WebSocket to be closed:

```
webSocket.onmessagereceived = function (args) {};
webSocket.onclosed = function (args) {};
```

Now you can start the WebSocket by passing an URI to the connectAsync method. This URI starts with *ws://* for plain WebSocket connections or *wss://* for secured WebSocket connections:

```
webSocket.connectAsync(uri);
```

If the connection is successful, you start listening for incoming messages. You send messages through the outputStream property of the WebSocket. By writing data to this stream, you can send messages to the server:

```
var messageWriter = new Windows.Storage.Streams.DataWriter(webSocket.outputStream);
messageWriter.writeString("message");
messageWriter.storeAsync();
```

This code shows you how to work with WebSockets on a low level. Microsoft is currently working on a library called SignalR, which encapsulates a lot of the work of building both clients and server applications that use WebSockets. If you start using WebSockets in any real-world app, you should definitely learn about SignalR.

EXAM TIP

Whenever you see a requirement for real-time communication in your exam, think about WebSockets.

Thought experiment

Retrieving remote data

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 building a multiplayer puzzle game. Users can look through their open games, start new games, and play games in real time against other users.

With this in mind, answer the following questions:

1. **Would you use HttpClient or WinJS.xhr for certain parts of your app?**
2. **How can you create the real-time portion of your communication?**

Objective summary

- Although WinJS.xhr can be used to access remote web services, using HttpClient is the preferred option.
- When working with REST services, you use HTTP verbs such as Get, Delete, Post, and Put. Those verbs are all supported by HttpClient.
- You can use HttpClient to listen for progress updates for your request.
- SOAP services follow a design different from REST services. SOAP is based on calling an actual method by name on the server and passing all required data in an XML envelope.
- Using WebSockets is a perfect solution whenever your app requires bidirectional communication with the server for real-time updates.

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 using a REST service and you want to add data to your service. Which HTTP verb should you use?
   - **A.** Get
   - **B.** Delete
   - **C.** Post
   - **D.** Put

2. You want to get only the headers of a web request. Which method should you use?
   - **A.** GetAsync(Uri)
   - **B.** GetAsync(Uri, HttpCompletionOption)
   - **C.** GetBufferAsync
   - **D.** GetInputStreamAsync
3. You want to set up bidirectional communication. Which technique should you use?
   - **A.** WinJS.xhr
   - **B.** HttpClient
   - **C.** WebSockets
   - **D.** jQuery

Objective 5.3: Implement data binding

The last two objectives looked at loading data from a variety of sources. In earlier chapters, you saw some examples of working with that data. This objective brings all the content together and shows you how to work with data in your apps.

Some of the content is duplicated throughout this book, but because this topic is so important for the real world and for your exam, it's an important objective to study.

This objective covers how to:

- Bind data to controls by using data-win-control and data-win-bind
- Choose and implement data-bound controls, including WinJS.UI.ListView, to meet requirements
- Bind data to item templates such as WinJS.Binding.Template
- Configure an iterator with data-win-options
- Enable filtering, sorting, and grouping data in the user interface

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

WinJS enables you to use special attributes to configure data binding, and one of them is the data-win-bind attribute. This attribute is used to bind a property of an element to a property of a data source. It uses the following syntax:

```
<element data-win-bind="elementProperty1 : dataSourceProperty1; elementProperty2:
dataSourceProperty2" />
```

The following HTML defines a div that has its innerText and style property bound:

```
<div id="boundDiv" data-win-bind="innerText: index; style.background: color"></div>
```

In your JavaScript code, you can now define an object with the properties index and then color and bind that object to your HTML:

```
var object = { index: 0, color: "orange" };
var div = document.getElementById("boundDiv");
WinJS.Binding.processAll(div, object);
```

That's all there is to the basics of data binding. This code takes your div and binds it to your newly created object.

By default, WinJS creates one-way binding, so you can update your data and the screen automatically updates. However, plain JavaScript objects don't support any notification mechanism to make this possible. To add notification support, you use the WinJS.Binding.as method.

The following code uses a timer to regularly update the color property of your object:

```
var colorArray = ["red", "green", "blue"];
var object = { index: 0, color: colorArray[0] };
var div = document.getElementById("boundDiv");

WinJS.Binding.processAll(div, object);

var bindingObject = WinJS.Binding.as(object);

 setInterval(function () {
     changeColor(bindingObject);
 }, 500);

 var index = 0;

 function changeColor(p) {
     if (index > 2) {
         index = 0;
     }
     p.index = index++;
     p.color = colorArray[index];
 };
```

Here you see how the WinJS.Binding.as method is used to take your regular object and transform it into an object that supports notifications.

Data-win-control takes things a step farther. With the data-win-control attribute, you take an existing div in your markup and convert into the host of a WinJS control. You can use all kinds of data controls and bind data to those prebuild controls by using this attribute.

Choosing and implementing data-bound controls

There are many controls available when Windows Store apps are built with JavaScript, HTML and CSS. Of course, you can use standard HTML controls such as hyperlinks, input fields, check boxes, and so on, but you can also take advantage of WinJS controls that are ready to use in your app.

Some controls are specifically designed to be used with collections of data: FlipView, Repeater, SemanticZoom, and ListView. These controls can be bound to a set of data and displayed on-screen. Pay attention to the ListView control because it is explicitly mentioned in the exam requirements.

Each control has its own specific uses. SemanticZoom (discussed in Chapter 4) is used when you want different views on the same data set. FlipView (discussed in Chapter 3) displays a collection of items one at a time and allows the user to flip through the collection. The Repeater (also discussed in Chapter 3) can generate HTML from a set of items.
As mentioned in the previous section, the data-win-control attribute is used to convert a regular div into a WinJS control. For example, the following markup defines a div for the ItemContainer control:

```
<div id="itemContainerControlHost" data-win-control="WinJS.UI.ItemContainer"></div>
```

Using data-bound controls follows the same pattern. This code creates a ListView:

```
<div id="basicListView"
    data-win-control="WinJS.UI.ListView">
</div>
```

Now you can define the data you want to bind to in your code-behind:

```
var dataArray = [
       { title: "A", text: "AAAA" },
       { title: "B", text: "BBBB" },
       { title: "C", text: "CCCC" },
       { title: "D", text: "DDDD" },
];

var dataList = new WinJS.Binding.List(dataArray);

var publicMembers =
{
    itemList: dataList
};
WinJS.Namespace.define("DataExample", publicMembers);
```

You expose the newly created binding list by adding to it to a namespace. Now that the data is available, you can add a data-win-option attribute to your div to bind to it:

```
<div id="basicListView"
     data-win-control="WinJS.UI.ListView"
     data-win-options="{ itemDataSource : DataExample.itemList.dataSource }">
</div>
```

The following sections show you how to use templates to customize data–bound item rendering and how to control the items that are bound to your control.

Binding data to item templates

The previous section showed some code to bind items with a title and text property to a ListView. Because the ListView doesn't know how to render the items, it displays them as strings.

If you don't want your items to display as strings, you can use templates instead. Templates define how an individual item renders when it is used in a data-bound control such as ListView.

A template is defined as a container div with a data-win-control of WinJS.Binding.Template:

```
<div id="myTemplate" data-win-control="WinJS.Binding.Template"></div>
```

Inside the container div, you can define the HTML elements that you want to make up one item. You can use data-win-bind to bind specific elements to properties on your data source. A template for the items from the previous section might look like this:

```
<div id="myTemplate" data-win-control="WinJS.Binding.Template">
    <div>
        <h1 data-win-bind="innerText: title"></h1>
        <span data-win-bind="innerText: text"></span>
    </div>
</div>
```

The item template defines two elements that are bound to the title and text property of the data source. Now that you have the template, you have to configure ListView to use it:

```
<div id="basicListView"
     data-win-control="WinJS.UI.ListView"
     data-win-options="{ itemDataSource : DataExample.itemList.dataSource,
      itemTemplate: myTemplate }">
</div>
```

It's that simple. Now ListView uses the template when rendering the individual items.

Configuring an iterator with data-win-options

Iterators in languages such as C# are objects that allow you to loop through a collection without having to worry about the actual implementation. WinRT application programming interfaces (APIs) define two iterator interfaces: IIterable<T> and IIterator<T>. You never work with those two interfaces directly; if you see an argument of IIterable when working with Windows Store apps, just think "array."

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

When you bind collection data, you can use the WinJS.Binding.List class to create your collection. This class has a couple of interesting projection methods that you can use:

- createFiltered
- createGrouped
- createSorted

For example, start with the following data set:

```
var dataArray = [
       { title: "A", text: "AAAA" },
       { title: "A", text: "AAAA" },
       { title: "B", text: "BBBB" },
       { title: "B", text: "BBBB" },
       { title: "B", text: "BBBB" },
       { title: "C", text: "CCCC" },
       { title: "C", text: "CCCC" },
       { title: "C", text: "CCCC" },
       { title: "C", text: "CCCC" },
       { title: "D", text: "DDDD" },
       { title: "D", text: "DDDD" },
];

var dataList = new WinJS.Binding.List(dataArray);
```

You can now use dataList to filter, sort, or group the data. Filtering requires a method that takes one item and returns true to signal that you want the item to be in the filtered collection (or false otherwise). So this line returns all objects that don't have a title of A:

```
var filteredList = dataList.createFiltered(function (x) { return x.title != "A"; });
```

Sorting takes a function with two parameters. It is your task to compare those two items and return a negative value if the first argument is less than the second, zero if both are equivalent, and positive if the first argument is greater than the second. This code sorts items by title in descending order:

```
var sortedList = dataList.createSorted(function (x, y) { return x.title < y.title; });
```

Grouping data is harder; you have to create two collections of data: one that contains the different groups with their headers and one with the data that belongs in each group. The following code shows how to group the data on the title character and count the number of items in each group:

```
var groupedList = dataList.createGrouped(function (x) { return x.title; },
                                         titleGroupData.bind(dataList));
function titleGroupData(j) {
    var titleArray = this.filter(function (v) { return (v.title == j.title); })
    return {
        title: j.title,
        count: titleArray.length
    }
}
```

When you display grouped data, you need a template for the individual items and both a template and the data for the group headers. The following markup shows this:

```
<div id="myTemplate" data-win-control="WinJS.Binding.Template">
    <div>
        <h1 data-win-bind="innerText: title"></h1>
        <span data-win-bind="innerText: text"></span>
    </div>
</div>
<div id="headerTemplate" data-win-control="WinJS.Binding.Template">
    <div>
        <h1 data-win-bind="innerText: count"></h1>
    </div>
</div>

<h1>Grouped</h1>
<div data-win-control="WinJS.UI.ListView"
     data-win-options="{
        itemDataSource : DataExample.groupedList.dataSource,
        itemTemplate: myTemplate,
        groupDataSource: DataExample.groupedList.groups.dataSource,
        groupHeaderTemplate: headerTemplate
     }"
    >
</div>
```

Remember that filtering, grouping, and sorting are projections over your data, not copies of your data; whenever the original data changes, all projections also update.

Thought experiment

Why use data binding?

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 the lead developer of a team that has built a lot of web applications and used jQuery to retrieve elements and populate them with data. You are building your first Windows Store app and you are discussing the benefits of using data binding with your team.

What are the advantages of using data binding over manual binding through a library such as jQuery?

Objective summary

- By using the data-win-bind attribute, you can bind data to elements in your UI.
- The data-win-control attribute lets you use WinJS controls in your app.
- WinJS offers various data–bound collection controls such as FlipView, Repeater, SemanticZoom, and ListView.
- Item templates such as WinJS.Binding.Template define the layout for items rendered in collection controls.
- Iterators in languages such as C# are objects that allow you to loop through a collection without having to worry about the actual implementation. WinRT application programming interfaces (APIs) define two iterator interfaces: IIterable<T> and IIterator<T>.
- WinJS.Binding.List is a collection class that can be used for data binding. It supports methods such as createFiltered, createGrouped, and createSorted to create projections of your data.

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 have set up data binding in your app. On the first run, the data is correctly shown in the UI. However, when you update the data in the back end, the UI doesn't reflect those changes. What have you done wrong?
   - **A.** You did not call WinJS.Binding.processAll after each update.
   - **B.** You forgot that WinJS doesn't support automatic updates of data-bound controls.
   - **C.** You have to use a WinJS.Binding.Template so WinJS knows how to render your newly updated item.
   - **D.** You have to use WinJS.Binding.as to set up change notification.
2. You have created a ListView and configured the itemDataSource to point to your data. When you launch the app, all your items are rendered as simple string representations. What should you do next? (Choose all that apply.)
   - **A.** Create an item template with WinJS.Binding.Template in the ListView element.
   - **B.** Use the data-win-option attribute to set the itemTemplate.
   - **C.** Create an item template with WinJS.Binding.Template outside of the ListView element.
   - **D.** Use the data-win-control attribute to set the itemTemplate.
3. You have a person object with a lastName property. You want a list with people starting with an A, sorted alphabetically. Which line do you use?
   - **A.** dataList.createFiltered(function (x) { return x. lastName.charAt(0) == "A"; }).createSorted(function (x, y) { return x. lastName > y. lastName; });
   - **B.** dataList.createSorted(function (x, y) { return x. lastName < y. lastName; });
   - **C.** dataList.createFiltered(function (x) { return x. lastName == "A"; }).createSorted(function (x, y) { return x. lastName > y. lastName; });
   - **D.** dataList.createFiltered(function (x) { return x. lastName.charAt(0) == "A"; })

Objective 5.4: Manage Windows authentication and authorization

The next two objectives look at security. Authorizing users, knowing what they are allowed to do and making sure this can't be compromised is becoming more and more important.

This objective discusses security-related features for storing passwords and credentials in a safe way. You also learn how to ask users for their credentials.

This objective covers how to:

- Retrieve a user's roles or claims
- Store and retrieve credentials by using the PasswordVault class
- Implement the CredentialPicker class
- Verify credential existence by using Credential Locker
- Store account credentials in app settings

NOTE **RETRIEVING A USER'S ROLES OR CLAIMS**

While the exam OD mentions retrieving a user's roles or claims as a possible exam topic, it seems unlikely. There is currently no information available on this topic in the context of developing Windows Store apps. This might be the result of the OD being written well before the product details are final.

Storing and retrieving credentials by using the PasswordVault class

Storing sensitive data in a secure location is important. Maybe you want to store users' login credentials so you can automatically authenticate them whenever they come back to your app. Or perhaps you are using some kind of security token to integrate with back-end systems. This token also needs to be stored in a secure location.

PasswordVault can help. Although the name suggests that you can store only passwords in it, it isn't the case. You can store any type of data that you want to be securely stored.

EXAM TIP

Never store security sensitive data in plain text on the users device. When the exam requires you to store sensitive data use the PasswordVault.

The PasswordVault class is found in the Windows.Security.Credentials namespace. Constructing a new instance can take some time, so it's wise to load PasswordVault asynchronously before you will use it, as the samples in the Microsoft software development kit (SDK) show:

```
function asyncVaultLoad()
{
    return new WinJS.Promise(function (complete, error, progress) {
        var vault = new Windows.Security.Credentials.PasswordVault();

        // any call to the password vault will load the vault
        var creds = vault.retrieveAll();
        complete();
    });
}
```

After constructing PasswordVault, you can add, read and remove credentials.

Adding a credential takes three properties:

- Resource name
- UserName
- Password

By using a unique resource name, you can later retrieve the credentials from PasswordVault.

You can use the following code to construct a new PasswordCredential and add it to PasswordVault:

```
var vault = new Windows.Security.Credentials.PasswordVault();
var cred = new Windows.Security.Credentials.PasswordCredential(resource,
                                                               username,
                                                               password);
vault.add(cred);
```

By default, the content of PasswordVault is roamed to other user devices. For example, if you get a new device and start configuring it, apps you already installed on your other devices can automatically get any required credentials; you don't have to reenter them on your new device.

Reading credentials from PasswordVault can be done in a couple of different ways. PasswordVault offers you access only to the values stored for your app. You can't see any credentials saved by other apps. The same is true for multiple users installing the same app on a shared device. The PasswordVault will keep the credentials of the users separated.

To list all the credentials that your app has saved, you can call retrieveAll on PasswordVault:

```
var creds = vault.retrieveAll();
```

You can also search for specific credentials by using findAllByResource or findAllByUserName:

```
creds = vault.findAllByResource(resource);
creds = vault.findAllByUserName(username);
```

If you have both the resource and user name value, you can use retrieve to search for a specific credential:

```
cred = vault.retrieve(resource,username);
```

One important difference between findAllByResource, findAllByUserName, and retrieve is that retrieve returns a fully populated object that includes the password. The findAll methods return objects with only the user name and resource set. If you want the password, you can use the user name and resource and then pass them to the retrieve method to avoid having to decrypt passwords for all credentials in the vault for the current user and app.

You can also remove a credential from PasswordVault. To do so, you need both the user name and resource name. First, retrieve the correct credential and then call vault.remove to delete it:

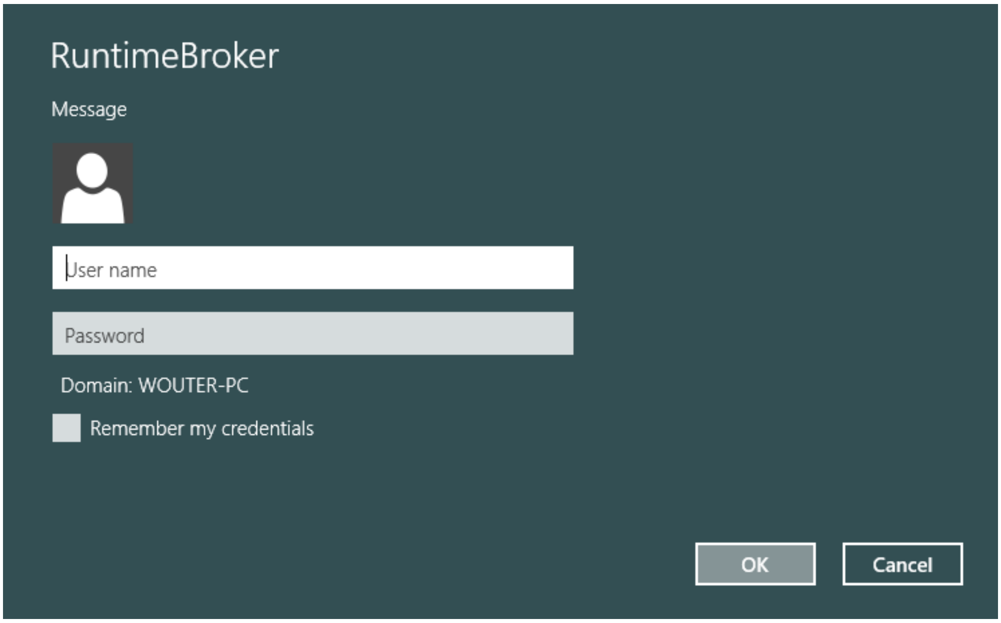
```
var vault = new Windows.Security.Credentials.PasswordVault();
var cred = vault.retrieve(resource, username);
vault.remove(cred);
```

MORE INFO **PASSWORDVAULT SAMPLE**

You can find a complete sample of how to use PasswordVault at *http://code.msdn.microsoft.com/windowsapps/PasswordVault-f01be74a/*.

Implementing the CredentialPicker class

To enable users to enter credentials in enterprise scenarios, Windows Store apps can use CredentialPicker. This object shows a UI to users, they enter credentials, and the entered values are returned to your app. An example of what CredentialPicker looks like is shown in Figure 5-1. Note that depending on if you are connecting remotely or working on a physical machine, you might see another option to authenticate with a smart card.


RuntimeBroker

Message

Domain: WOUTER-PC

Remember my credentials

FIGURE 5-1 The CredentialPicker UI

CredentialPicker provides the UI for domain logins complete with support for smart cards. You can find the CredentialPicker class in the Windows.Security.Credentials.UI namespace. CredentialPicker takes a couple of options to construct:

- **Target Name** This is ignored
- **Message** This is shown in Figure 5-1
- **Caption** This is shown at the top of the image in Figure 5-1 as RuntimeBroker

When activating CredentialPicker, you have three overloads. The simplest one is this:

```
Windows.Security.Credentials.UI.CredentialPicker.pickAsync(target, message)
.done(function (results) { }
```

The pickAsync method takes only a target and message. You can also pass the caption by using another overload:

```
Windows.Security.Credentials.UI.CredentialPicker.pickAsync(target, message, caption)
.done(function (results) { }
```

The third overload of pickAsync allows you to pass a CredentialPickerOptions object that allows for more configuration options:

- **AlwaysDisplayDialog** Gets or sets the option of whether the dialog box is displayed
- **AuthenticationProtocol** Gets or sets the authentication protocol
- **CallerSavesCredential** Gets or sets whether the caller wants to save the credentials

- **Caption** Gets or sets caption text displayed to the user
- **CredentialSaveOption** Gets or sets the option of saving credentials
- **CustomAuthenticationProtocol** Gets or sets whether the authentication protocol is custom rather than a standard authentication protocol
- **ErrorCode** Gets or sets the error code
- **Message** Gets or sets the body of text that displays to the user
- **PreviousCredential** Gets or sets whether to fill dialog box fields with previous credentials
- **TargetName** Gets or sets the name of the target computer (not used at the moment)

The ErrorCode property lets you show the CredentialPicker UI with an error message such as "The Username Or Password Is Incorrect". You can use this message to tell users what went wrong during logon. Error codes include the following:

- **1326** Logon failure
- **1330** Password expired
- **2202** Bad user name
- **1907** or **1938** Password must change/password change required
- **1351** Can't access domain info
- **1355** No such domain

Authentication protocol is one of the values found in the AuthenticationProtocol enumeration:

- **Basic** The authentication protocol is basic. Credentials are returned to the caller as plaintext.
- **Digest** The authentication protocol is digest. Credentials are returned to the caller as plaintext.
- **Ntlm** The authentication protocol is NTLM. Credentials are transformed before being returned to the caller.
- **Kerberos** The authentication protocol is Kerberos. Credentials are transformed before being returned to the caller.
- **Negotiate** The authentication protocol is negotiate, including negotiate extensions. Credentials are transformed before being returned to the caller.
- **CredSsp** The authentication protocol is for remote access using the Credential Security Support Provider (CredSSP) protocol.
- **Custom** The authentication protocol is anything other than the previous ones. Credentials are returned to the caller as plaintext.

The back end you use determines which authentication protocol you need.

The following code sample uses CredentialPickerOptions to show CredentialPicker to the user with an error message:

```
var options = new Windows.Security.Credentials.UI.CredentialPickerOptions();
options.message = "Let's login!";
options.caption = "CredentialPickerSample";
options.targetName = "Target";
options.alwaysDisplayDialog = true;
options.errorCode = 1326; // Shows "The username or password is incorrect."
options.callerSavesCredential = true;
options.authenticationProtocol = Windows.Security.Credentials.UI.AuthenticationProtocol.
negotiate;
options.credentialSaveOption = Windows.Security.Credentials.UI.CredentialSaveOption.
selected;

Windows.Security.Credentials.UI.CredentialPicker.pickAsync(options).done(function
(results) {
});
```

Verifying credential existence by using Credential Locker

One popular use of PasswordVault is to store user credentials so users don't have to reenter their credentials every time they access your app.

When you need to authenticate the user, you should check whether the correct credentials are already stored in PasswordVault. If not, you can use the CredentialPicker UI to let users enter their credentials and then store them in PasswordVault for later use.

The following sample code shows how to differentiate between the users credentials being available or having to ask the user for credentials. You check to see whether the credentials are available from PasswordVault. If they are available, you can use them; if not, you have to show the UI to the user, get the credentials, and store them for later use.

```
function showCredentialPickerIfNotStored() {
    var loginCredential = getCredentialFromLocker();
    if (loginCredential != null) {
        loginCredential.retrievePassword();
        document.getElementById("output").innerText = loginCredential.userName + ": " +
                                                      loginCredential.password;
    } else {
        Windows.Security.Credentials.UI.CredentialPicker.pickAsync("target",
                      "please enter credentials").done(function (results) {
            loginCredential = results;
            storeCredentialsInLocker(loginCredential);
            document.getElementById("output").innerText =
                               loginCredential.credentialUserName + ": " +
                               loginCredential.credentialPassword;
        });
    }
}
```

The following code shows the basic structure of checking to see whether credentials are available. If they aren't, show CredentialPicker. The getCredentialFromLocker function looks like this:

```
function getCredentialFromLocker() {
    try {
        var vault = new Windows.Security.Credentials.PasswordVault();
        var credentialList = vault.findAllByResource(resourceName);
        if (credentialList.length > 0) {
            return credentialList[0];
        }
        else {
            return null;
        }
    }
    catch (ex) {
        return null;
    }
}
```

The try/catch block is important because the findAllByResource method throws an exception when nothing is found for the specified resource name.

Storing the credentials in PasswordVault is done like this:

```
function storeCredentialsInLocker(loginCredential) {
    var vault = new Windows.Security.Credentials.PasswordVault();
    var cred = new Windows.Security.Credentials.PasswordCredential(resourceName,
                                                           loginCredential.
credentialUserName,
                                                           loginCredential.
credentialPassword);
    vault.add(cred);
}
```

By using this code, you can now load the credentials from PasswordVault or get them from the user if PasswordVault is empty.

EXAM TIP

The Credential Locker is the object in Windows that stores credentials. PasswordVault is a class that you can use from your Windows Store apps that uses the Credential Locker behind the scenes. So whenever you see something about the Credential Locker on your exam, remember that this can be used through the PasswordVault class.

Storing account credentials in app settings

In addition to storing data in the PasswordVault, you can also store settings in the ApplicationData.localSettings and ApplicationData.roamingSettings properties. However, storing secure data in ApplicationData is not recommended because ApplicationData is stored on the users device at *C:\Users\<user_name>\AppData\Local\Packages\<package>\Settings\settings.dat*.

This file can be edited by using the registry editor making it easy for a malicious user to change the data in your settings file outside of your app.

If you do want to store sensitive data in the app settings, you should at least encrypt the data. The problem with encrypting, however, is that the key used for the encryption process needs to be stored in the application code that can also be accessed by the user.

Hashing data is another option. Hashing is a one-way mathematical operation where you take some data and generate a fixed length value. This resulting value can't be transformed into the original value. This means that storing a hashed value is more secure then storing the original value.

For example, the following code is a simple example of a hash function:

```
var hashCode = function (s) {
    return s.split("").reduce(function (a, b) { a = ((a << 5) - a) + b.charCodeAt(0);
return a & a }, 0);
}
```

You can use this function like this:

```
var username = "Baz";
var password = "bar";

var hash = hashCode(username + password);
var outputDiv = document.getElementById("outputDiv");
outputDiv.innerHTML += "username: " + username + " password: " + password;
outputDiv.innerHTML += "<br />";
outputDiv.innerHTML += "hash: " + hash;
outputDiv.innerHTML += "<br />";
outputDiv.innerHTML += "password: xxx " + (hashCode(username + "xxx") == hash).
toString();
outputDiv.innerHTML += "<br />";
outputDiv.innerHTML += "username: xxx " + (hashCode("xxx" + password) == hash).
toString();
outputDiv.innerHTML += "<br />";
outputDiv.innerHTML += "correct credentials: " + (hashCode(username + password) ==
hash).toString();
```

By comparing the credentials to the hash, you can verify if the user entered the same username and password combination. Nowadays, hash functions can be cracked by using brute-force approaches that use video card GPUs. But as always is the case with security, adding layers makes it more difficult for malicious users to attack your app.

MORE INFO **HASHING ALGORITHMS**

The quality of your hash depends on the hashing algorithm you use. An algorithm should create values that are as unique as possible to avoid collisions in the hashed values. One implementation of a hashing algorithm that is also used on servers, namely SHA256, can be found at *https://bitwiseshiftleft.github.io/sjcl/*.

Now that you know something about hashing, the remaining part of this objective looks at using the localSettings and roamingSettings properties to store data.

The following code sample shows how to use the localSettings property:

```
var applicationData = Windows.Storage.ApplicationData.current;
var settings = applicationData.localSettings;

// Create a simple setting
settings.values["exampleSetting"] = "Hello Windows";

// Read data from a simple setting
var value = settings.values["exampleSetting"];

if (!value)
{
    // No data
}
else
{
    // Access data in value
}

// Delete a simple setting
settings.values.remove("exampleSetting");
```

To work with roaming instead of local data, change the following line:

```
var settings = applicationData.localSettings;
```

to this line:

```
var settings = applicationData.roamingSettings;
```

All other code can stay the same.

If you want more structure in your settings, you can use a container. Creating a container is easy:

```
var applicationData = Windows.Storage.ApplicationData.current;
var localSettings = applicationData.localSettings;

var container = localSettings.createContainer("exampleContainer",
    Windows.Storage.ApplicationDataCreateDisposition.Always);
```

The ApplicationDataCreateDisposition enumeration can have a value of always to make sure that the container is created if it doesn't exist. A value of existing opens the container only if it already exists.

You can see whether a container exists by calling this:

```
var hasContainer = localSettings.containers.hasKey(containerName);
```

You can get to the values stored in a container by using the lookup method:

```
var hasSetting = localSettings.containers.lookup(containerName).values;
```

Use the following code to write a value to your container:

```
localSettings.containers.lookup(containerName).values[settingName] = "Hello World";
```

Retrieving a value is also easy:

```
var value = localSettings.containers.lookup(containerName).values[settingName];
```

And, finally, you can delete a container:

```
localSettings.deleteContainer(containerName);
```

Thought experiment

Authenticating 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 building an intranet app that connects to various systems in your organization. Some systems don't integrate well, so users have to enter their credentials.

List the features discussed in this objective that you will use.

Objective summary

- Use PasswordVault to securely store credentials on a user's device.
- CredentialPicker offers a standard UI for users to enter credentials.
- To help users use your app, make sure to store their credentials in PasswordVault so they don't have to reenter credentials when coming back to your app.
- You can use ApplicationData to store settings locally or roaming.

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 want to securely store user credentials. Which class do you use?
   - **A.** CredentialLocker
   - **B.** PasswordVault
   - **C.** CredentialPicker
   - **D.** ApplicationData
2. A user entered incorrect credentials in CredentialPicker. You want to ask the user to reenter credentials. Which method do you use?
   - **A.** CredentialPicker.pickAsync(target, message, caption)
   - **B.** CredentialPicker.pickAsync(target, message)
   - **C.** CredentialPicker.pickAsync(CredentialPickerOptions) with CredentialSaveOption set to false
   - **D.** CredentialPicker.pickAsync(CredentialPickerOptions) with ErrorCode set to 1326
3. You want to see whether a user has already entered credentials for your app. Which elements do you need? (Choose all that apply.)
   - **A.** PasswordVault findAllByResource method
   - **B.** A try/catch block
   - **C.** CredentialPicker.pickAsync
   - **D.** PasswordCredential.retrievePassword();

Objective 5.5: Manage web authentication

The previous objective discussed the security features Windows Store apps can use. This objective discusses securing your app by integrating it with external services such as Facebook or Twitter, setting up single sign-on, and using other important security features.

This objective covers how to:

- Use the Windows.Security.Authentication.Web namespace
- Set up OAuth2 for authentication
- Set up single sign-on (SSO)
- Implement the CredentialPicker class
- Implement credential roaming
- Implement the WebAuthenticationBroker class
- Support proxy authentication for enterprises

Using the Windows.Security.Authentication.Web namespace

The rest of this objective shows you how to use various members of the Windows.Security.Authentication.Web namespace. For now, it's important to have a conceptual foundation when it comes to authentication and the web.

Regulating complete security around user authentication at the client is hard, which is why most apps use a server back end that handles authentication. Your app sends a password and user name to the back-end service and gets special *token* in return. This token represents the user and allows your app to send the token when issuing new requests to the back-end service. You can now store the token in PasswordVault and use it in the future.

Another way to authenticate users is to completely outsource the whole process to another service. Maybe you came across websites that let you log on with your Facebook, Twitter, Windows Live, and Gmail accounts. What happens is that the user is redirected to another page that is under control of Microsoft or Google, for example. The user enters the user name and password on that page and is authenticated. This authentication happens with the OAuth2 protocol, which describes how the communication should take place.

When working with external web services for communication, you also have to think about the Internet lines that are between the user's app and the server. Protecting the data while it is traveling from one end to the other is also an important part of authentication and is where SSL is used. Websites that use SSL start their addresses with "https". Whenever you see a website that starts with "https" instead of "http", you know that the communication between your browser and the website is being encrypted and ensures that potential listeners can't read the content you are sending. Building a secure authentication mechanism in your app starts with using the correct https protocol for sending data.

The remaining part of this objective discusses how to implement authentication features in your app.

Setting up OAuth2 for authentication

OAuth2 authentication is the process of allowing an external website to handle your authentication. This website can be a well-known party such as Facebook or Twitter.

This process starts with your app providing the URI of the authentication page of an external site. This URI needs to start with "https" to make sure the communication is encrypted and is then loaded into an overlay in your app. This page is completely defined by the OAuth2 provider; you can't change its content or appearance.

It could be that authenticating yourself with an OAuth2 provider takes you through multiple pages. Maybe the first page lets you enter your credentials while other pages explicitly ask you to allow certain data to be exchanged. To make sure that your app knows when the authentication is finished, you specify not only the starting URI but also the URI that signals authentication is done. Your app makes sure that when this page is hit, the logon overlay is closed, and the user returns to your app.

The broker that orchestrates this process, WebAuthenticationBroker, is found in the Windows.Security.Authentication.Web namespace.

For example, suppose that you want to set up OAuth2 authentication with Twitter. The first thing you should do is set up a new Twitter app at *https://apps.twitter.com/app/new*. Here you specify the callback URL that is used by your app to verify that authentication is finished. After creating your app, you receive two important values: your API key and your API secret. Those values are required for authenticating your app with Twitter.

You can now use those values in the WebAuthentication sample that Microsoft created.

MORE INFO **WEBAUTHENTICATION SAMPLE**

You can find the WebAuthentication sample at *http://code.msdn.microsoft.com/windowsapps/Web-Authentication-d0485122*.

The sample contains a lot of code that you can inspect. However, the most important step is this:

```
Windows.Security.Authentication.Web.WebAuthenticationBroker.authenticateAsync(
            Windows.Security.Authentication.Web.WebAuthenticationOptions.none,
            startURI,
            endURI).done(function (result) {
});
```

The call to WebAuthenticationBroker.authenticateAsync starts the OAuth2 connection. By passing it the start and end URI, the broker knows which page to load and when the authentication is finished. The start URI is composed of your secret keys and some other data that Twitter requires.

EXAM TIP

Remember that OAuth2 depends on both a start URI and an end URI for configuration the WebAuthenticationBroker.

After the authentication finishes, the returned result has a responseData property that has the data passed to you by the OAuth2 provider. Depending on your OAuth2 provider, you know the schema of the data and you can parse it for the token and other data you require.

Offering an OAuth2 sign-in option in your app makes it easier for users to use your app. Instead of having to create a new account, they can use an existing account and just give your app the correct permissions.

Setting up Single Sign-On (SSO)

If you use PasswordVault and CredentialPicker correctly, your app asks for the user's credentials only once. However, multiple apps using the same back-end services still ask the users to provide their credentials.

From the users' point of view, nothing has improved because they still enter credentials multiple times. SSO can be used to authenticate the user in one app for a given OAuth2 provider, effectively logging users on for all apps that use that OAuth2 provider.

SSO requires cookies to be stored in an application. Although the default behavior of the WebAuthenticationBroker class doesn't allow cookie storage, when the WebAuthenticationBroker class is set up for SSO authentication, cookies received during the authentication steps can remain in a special SSO application container. To support SSO, the online identity provider must allow the registration of URLs with the format ms-app://[Application SID]. The application's package ID (also called the security identifier [SID]) can be obtained from the application's page in the Windows Store.

The following steps are required to set up SSO with WebAuthenticationBroker in Windows Store apps:

1. Register your application with the identity provider and obtain a client ID and secret key. Request the provider to register a redirect URL of the form ms-app://[Application SID].
2. Use the SilentMode and UseHttpPost members of the WebAuthenticationOptions enumeration in the AuthenticateAsync method; do not specify the redirect URL.
3. If authentication was successful, the online identity provider redirects the app to the redirect URL configured with an access token as a query parameter.

4. If the redirect URL matches with the application's SID, the WebAuthenticationBroker class returns the token to the app. The whole URL is kept in a cookie stored in a special app container so that future authentication requests do not require the user to log on with the credentials.

The cookies used for SSO in a Windows Store app are not shared with Internet Explorer, other browsers, or other apps. If an app attempts to misuse a cookie for an app in which the user is already logged on, it fails authentication because the redirect URL of the two apps never match.

MORE INFO **FACEBOOK SSO SAMPLE**

A sample of implementing SSO with Facebook can be found at *http://facebooksdk.net/docs/windows/sso/*.

Implementing the CredentialPicker class

The CredentialPicker class is used in Windows Store apps to show users a familiar interface in which they can enter their credentials. You can find the CredentialPicker class in the Windows.Security.Credentials.UI namespace.

MORE INFO **CREDENTIALPICKER CLASS**

The CredentialPicker class is discussed in more detail in Objective 5.4.

Implementing credential roaming

Credential roaming is the process in which a user logs on in your app or device, after which their credentials are automatically roamed to other devices. This way, they have to log on only once for each app.

The PasswordVault class provides built-in support for roaming credentials.

MORE INFO **CREDENTIAL ROAMING CLASS**

Credential roaming and PasswordVault are discussed in more detail in Objective 5.4.

Implementing the WebAuthenticationBroker class

In the "Setting up OAuth2 for authentication", section, you saw how to use the WebAuthenticationBroker class. The idea is that you launch WebAuthenticationBroker with a start and end URI. This start URI points to a specific provider and often contains a unique ID that helps the provider identify your app.

When the authentication provider navigates to the end URI, WebAuthenticationBroker knows the authentication is finished and closes the UI.

When the authentication finishes, it's up to you to see whether everything went well and to parse the result.

The following code shows how to log on with a Facebook provider:

```
var facebookURL = "https://www.facebook.com/dialog/oauth?client_id=XXXXXXXXXXXXXXX";

var callbackURL = "https://www.facebook.com/connect/login_success.html";

facebookURL += "&redirect_uri=" + encodeURIComponent(callbackURL) + "&scope=read_
stream&display=popup&response_type=token";

var startURI = new Windows.Foundation.Uri(facebookURL);
var endURI = new Windows.Foundation.Uri(callbackURL);

Windows.Security.Authentication.Web.WebAuthenticationBroker.authenticateAsync(
    Windows.Security.Authentication.Web.WebAuthenticationOptions.none, startURI, endURI)
    .done(function (result) {
    }, function (err) {
    });
```

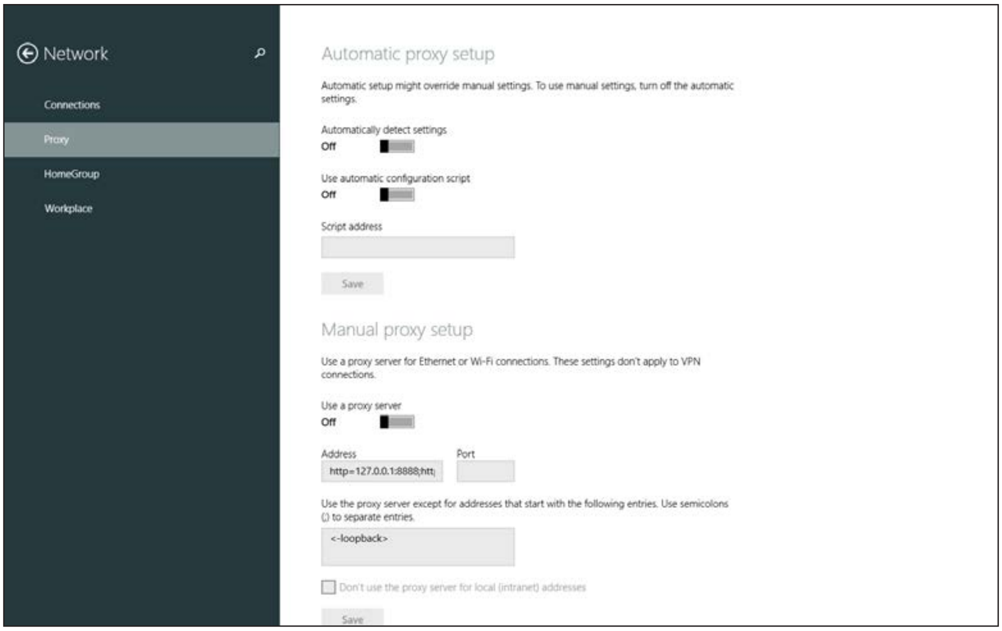
You should decide what to do when an error happens. It's also up to you to parse the incoming result and extract the token.

Supporting proxy authentication for enterprises

When an app runs in an enterprise environment, Internet traffic often runs through a proxy.

With the release of Windows 8.1, proxy authentication works out of the box. When your network uses Web Proxy Auto-Discovery (WPAD) and Privilege Attribute Certificate (PAC), Windows prompts the user for authentication to connect to the network.

If you want to configure proxy settings on your device, press **Windows key+W** and search for Change Proxy Settings. You see the screen shown in Figure 5-2.



Automatic proxy setup

Automatic setup might override manual settings. To use manual settings, turn off the automatic settings.

Automatically detect settings
Off

Use automatic configuration script
Off

Script address

Save

Manual proxy setup

Use a proxy server for Ethernet or Wi-Fi connections. These settings don't apply to VPN connections.

Use a proxy server
Off

Address: http=127.0.0.1:8888;htt|
Port:

Use the proxy server except for addresses that start with the following entries. Use semicolons (;) to separate entries.

<-loopback>

Don't use the proxy server for local (intranet) addresses

Save

FIGURE 5-2 Configuring proxy settings in Windows 8.1

If your organization has set up a Group Policy that configures the proxy settings for Internet Explorer, those settings are automatically used in Windows Store apps.

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 have enabled logging in your app with logging so you can see what your users are doing, and you notice that your app is becoming more and more popular. However, you also notice that a lot of users stop using your app when they are asked to accounts.

How can you enhance your app to address this issue?

Objective summary

- The Windows.Security.Authentication.Web namespace contains the WebAuthenticationBroker class that you can use to implement web authentication scenarios.
- OAuth2 allows you to use an external provider to authenticate users for your app. This external provider could be Twitter, Facebook, or a Microsoft account.
- SSL is used to authenticate a user once for a back-end service in multiple apps. It is supported by putting a special access token in the URI, after which WebAuthenticationBroker handles SSO.

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. By popular demand, you have decided to implement photo sharing using your photo mosaic Windows Store app with Facebook. You have to allow users to post photos to their Facebook timelines. What are the steps required to implement this feature? (Choose all that apply.)
   - **A.** Authenticate users by requesting Facebook credentials in a custom dialog box.
   - **B.** Register your app with Facebook to obtain a client ID; configure a redirect URL to obtain an access token.
   - **C.** Set up the WebAuthenticationBroker with a URL that contains the client ID, redirect URL, and the permissions requested.
   - **D.** Use the AuthenticateAsync method of the WebAuthenticationBroker class to authenticate users and use the access token from the result for posting photos to Facebook using the recommended APIs.
   - **E.** After users are logged on, post photos to Facebook using the methods available for desktop apps.
2. To improve employee productivity in an organization that provides a Windows Store to its employees, you recommend that the organization uses SSO in the app. Which of the following is true of SSO?
   - **A.** Implementation of SSO requires the application's SID to be present in a URL that is used as the redirect URL.
   - **B.** Each time an update for the app is released, users need to log on again because the cookie is no longer available as the SID changes with every update.
   - **C.** SSO works with both http:// and https://. Therefore, if employees are logged on to the corporate network through a virtual private network (VPN), they do not require http:// for the authentication.
   - **D.** A user can use SSO to log on to any app along with the organization's app using the same set of credentials; this is one of the major benefits of using SSO.

3. You are investigating the feasibility of implementing OAuth2 authentication using an online identity provider. To provide a consistent user experience, you decide to use the WebAuthenticationBroker class in your app. Which of the following is a correct statement?
   - **A.** The WebAuthenticationBroker class supports OAuth2 authentication with only those identity providers that are located in the domain in which the user has joined with the app.
   - **B.** The WebAuthenticationBroker class supports OAuth2 authentication with online identity providers that implement the OAuth2 standard and it also supports SSO.
   - **C.** The WebAuthenticationBroker class must be used every time the user wants to log on because it does not support SSO.
   - **D.** The WebAuthenticationBroker class is suitable for use with online identity providers that support only the http:// prefix.

Answers

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

Objective 5.1: Thought experiment

The Weather app stores your preferred cities and current location. Storing these preferences can be done in the roaming settings folder so that each device knows your settings. The actual weather data is retrieved remotely. Although the data can be cached locally, it should be updated from a remote back end.

Objective 5.1: Review

1. **Correct answer:** D
   - **A. Incorrect:** The Roaming folder is used only for small files. Although the roaming option would be perfect, storing large video files is not an option.
   - **B. Incorrect:** Tables in Azure Mobile Services shouldn't be used to store binary data.
   - **C. Incorrect:** Storing the video files in the local folder can be used as a cache, but they don't sync to other devices.
   - **D. Correct:** You should store them remotely in Azure blob storage because of the file size and need for availability on multiple devices.
2. **Correct answers:** A, B
   - **A. Correct:** Azure Mobile Services can be used to store details about the image such as the rating.
   - **B. Correct:** Blob storage can be used to store the actual image.
   - **C. Incorrect:** The roaming folder is use-specific. It is not meant to be used to share data with other users.
   - **D. Incorrect:** The local folder is on only one device. It is not meant to be used to share data with other users.
3. **Correct answer:** B
   - **A. Incorrect:** Executing a web service request asynchronously doesn't make it finish faster.
   - **B. Correct:** By executing the request asynchronously, your app can handle user input while it waits for the request to finish.
   - **C. Incorrect:** The fact that the client app invokes the request asynchronously doesn't influence the web server.
   - **D. Incorrect:** Users can always fire multiple requests at the same time from multiple applications. It is handled by Windows and doesn't have anything to do with sending requests asynchronously from your app.

Objective 5.2: Thought experiment

1. HttpClient is the preferred solution over WinJS.xhr. You can use HttpClient to load open games and start a new game.
2. The real-time game play can be done with WebSockets.

Objective 5.2: Review

1. **Correct answer:** C
   - **A. Incorrect:** The Get verb is used to read or select data.
   - **B. Incorrect:** Delete is used to remove data.
   - **C. Correct:** Post is used to add data.
   - **D. Incorrect:** Put is used to update data.
2. **Correct answer:** B
   - **A. Incorrect:** GetAsync(Uri) downloads the complete request.
   - **B. Correct:** You can specify that you want only the request headers, not the content.
   - **C. Incorrect:** GetBufferAsync loads the data as a buffer.
   - **D. Incorrect:** GetInputStreamAsync streams the incoming data.
3. **Correct answer:** C
   - **A. Incorrect:** WinJS.xhr is used for regular AJAX requests. HttpClient should be favored when doing those types of requests.
   - **B. Incorrect:** HttpClient isn't used for real-time communication; it fires only one single request.
   - **C. Correct:** Using WebSockets is the perfect solution for real-time communication.
   - **D. Incorrect:** jQuery is a JavaScript framework that can be used to send AJAX requests. It doesn't feature real-time communication.

Objective 5.3: Thought experiment

When you use your JavaScript or a library such as jQuery to select elements in your HTML and set their values from your code, you create a hard dependency between your code and the UI. Something simple, such as changing the element ID or moving it around in your HTML, can break your code.

Data binding creates a loose coupling between your data and the UI, which makes for a more maintainable application in which it is easier to change the layout without having to change the JavaScript.

Objective 5.3: Review

1. **Correct answer:** D
   - **A. Incorrect:** WinJS.Binding.processAll has to be called only once to initiate the binding.
   - **B. Incorrect:** WinJS supports automatic updates of data-bound controls. Although JavaScript objects don't have intrinsic change notifications, you can easily add them to an existing object.
   - **C. Incorrect:** WinJS templates define the layout of data-bound items; they don't control data updates.
   - **D. Correct:** WinJS.Binding.as adds change notification features to an existing object. WinJS checks for updates and automatically refreshes the UI.
2. **Correct answers:** B, C
   - **A. Incorrect:** The item template should be a separate element outside of the ListView element.
   - **B. Correct:** The data-win-option attribute on ListView is used to connect ListView to the template.
   - **C. Correct:** The item template is a separate element located outside of ListView.
   - **D. Incorrect:** You should use the data-win-option attribute, not data-win-control.
3. **Correct answer:** A
   - **A. Correct:** Here you compare only the first character of the last name and create a collection that is both filtered and sorted.
   - **B. Incorrect:** The resulting collection is not filtered.
   - **C. Incorrect:** The filter function checks to see whether the complete name is equal to A, not only the first character.
   - **D. Incorrect:** The resulting collection is not sorted.

Objective 5.4: Thought experiment

CredentialPicker can be used whenever the user needs to enter credentials for a back-end system. PasswordVault can then be used to store those credentials so the user has to enter them only once.

Objective 5.4: Review

1. **Correct answer:** B
   - **A. Incorrect:** The CredentialLocker class doesn't exist; PasswordVault is sometimes called a Credential Locker.
   - **B. Correct:** PasswordVault is recommended for storing sensitive data such as credentials.
   - **C. Incorrect:** CredentialPicker shows the UI for users to enter their credentials.
   - **D. Incorrect:** ApplicationData is not safe for storing security sensitive data.
2. **Correct answer:** D
   - **A. Incorrect:** This shows the CredentialPicker UI as if it were the first time. No error message is displayed.
   - **B. Incorrect:** This shows the CredentialPicker UI as if it were the first time. No error message is displayed.
   - **C. Incorrect:** CredentialSaveOption doesn't show an error message.
   - **D. Correct:** Using CredentialPickerOptions with the correct error code shows an error message to the user.
3. **Correct answers:** A, B, D
   - **A. Correct:** The findAllByResource method searches PasswordVault for the credential you are looking for.
   - **B. Correct:** If no credentials are saved for the particular resource, an exception is thrown that you need to handle.
   - **C. Incorrect:** This shows the CredentialPicker UI so users can enter their credentials.
   - **D. Correct:** After loading PasswordCredential by using findAllByResource, you have to explicitly load the password for the retrieved credential.

Objective 5.5: Thought experiment

Users often dislike having to create a new account for each app they want to use. Instead, you can offer OAuth2 integration so they can log on with an existing account, such as Windows Live ID, Twitter, or Facebook.

Objective 5.5: Review

1. **Correct answers:** B, C, D
   - **A. Incorrect:** Users might never enter user names and passwords from third-party sites into an app's custom dialog box because of lack of trust and fear of having their credentials compromised.
   - **B. Correct:** Registration of a new app with a remote service provides your app with a client ID, secret key, and option to set up the redirect URL.
   - **C. Correct:** The URL used by the WebAuthenticationBroker class needs to contain the client ID; redirect URI; and, in most cases, the scope for the permissions requested.
   - **D. Correct:** If the authenticateAsync is successful in authenticating the user, the response contains the access token. The app then uses the access token with all future requests.
   - **E. Incorrect:** An app is expected to use well-defined APIs for accessing resources for users and carrying out actions such as posting photos. These processes require an access token that is available only through authentication using OAuth2.
2. **Correct answer:** A
   - **A. Correct:** The WebAuthenticationBroker class uses a custom URL that contains the SID of the app to ensure that the access token contained in a cookie in the SSO app container belongs to the app.
   - **B. Incorrect:** The SID of an application does not change with updates, so the user is automatically signed in after the app is updated.
   - **C. Incorrect:** The WebAuthenticationBroker class can be used only with URLs that support the https:// prefix.
   - **D. Incorrect:** Other apps can't log on with the user's organizational domain account. SSO can help authenticate back-end services.
3. **Correct answer:** B
   - **A. Incorrect:** The WebAuthenticationBroker class supports identity providers that implement the OAuth2 standard and are located on an intranet or the Internet.
   - **B. Correct:** The WebAuthenticationBroker class can be configured for SSO with OAuth2 providers. SSO requires a redirect URL containing the app's SID to be set up with the identity provider.
   - **C. Incorrect:** The WebAuthenticationBroker class supports SSO for Windows Store apps.
   - **D. Incorrect:** The WebAuthenticationBroker class can be used only with URLs that support the https:// prefix.

Index

This page intentionally left blank

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.

This page intentionally left blank