Microsoft®

Build Windows® 8 Apps with Microsoft® Visual C#® and Visual Basic®

Luca Regnicoli
Paolo Pialorsi
Roberto Brunetti

ebook+exercises

Step by Step

Build Windows 8 Apps with Microsoft Visual C# and Visual Basic Step by Step

Your hands-on, step-by-step guide to building Windows 8 apps using the Microsoft .NET Framework.

Teach yourself how to create apps for Windows 8 and the Windows Store—one step at a time. Ideal for those with intermediate to advanced .NET development skills, this tutorial provides practical, learn-by-doing exercises to help you master the fundamental techniques for building Windows 8 apps using the .NET Framework 4.5 with Visual C# 2012 or Visual Basic 2012.

Discover how to:

- Employ Windows 8 design and usability principles
- Apply tools and libraries from Visual Studio® and the Windows 8 SDK
- Use Windows Runtime APIs and create a library of custom APIs
- Develop flexible layouts that work in landscape and portrait views
- Enhance the user experience using live tiles, badges, and toasts
- Define robust and security-enhanced architectures for your apps
- Use async techniques to build modern applications
- Prepare your app for the Windows Store

Your *Step by Step* digital content includes:

- Downloadable practice files
  See http://go.microsoft.com/FWLink/?Linkid=275453
- Fully searchable eBook. *See the instruction page at the back of the book.*

About the Authors

Luca Regnicoli is a cofounder of DevLeap, a company that provides content and consulting to professional developers. He has more than 10 years of experience developing the presentation tier of enterprise applications.

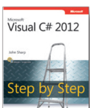
Paolo Pialorsi, a cofounder of DevLeap, is a trainer and author whose books include *Programming Microsoft LINQ in Microsoft .NET Framework 4* and *Microsoft SharePoint 2010 Developer Reference*.

Roberto Brunetti, a cofounder of DevLeap, is an experienced consultant, trainer, and author. He is a regular speaker at major technology conferences and the author of *Windows Azure™ Step by Step*.

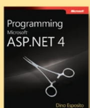
DEVELOPER ROADMAP

Start Here!
- Beginner-level instruction
- Easy-to-follow explanations and examples
- Exercises to build your first projects

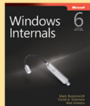
Step by Step
- For experienced developers learning a new topic
- Focus on fundamental techniques and tools
- Hands-on tutorial with practice files plus eBook



Developer Reference
- Professional developers; intermediate to advanced
- Expertly covers essential topics and techniques
- Features extensive, adaptable code examples



Focused Topics
- For programmers who develop complex or advanced solutions
- Specialized topics; narrow focus; deep coverage
- Features extensive, adaptable code examples



microsoft.com/mspress

ISBN: 978-0-7356-6695-5

U.S.A. $44.99
Canada $47.99
[*Recommended*]

Programming/Windows

Microsoft®

Microsoft

Build Windows® 8 Apps with Microsoft® Visual C#® and Visual Basic® Step by Step

Luca Regnicoli
Paolo Pialorsi
Roberto Brunetti

Copyright © 2013 by Luca Regnicoli, Paolo Pialorsi, Roberto Brunetti.
All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-6695-5

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

Printed and bound in the United States of America.

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

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

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

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

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Melanie Yarbrough

Editorial Production: S4Carlisle Publishing Services

Technical Reviewer: John Mueller

Indexer: WordCo Indexing Services

Cover Design: Twist Creative • Seattle

Cover Composition: Zyg Group, LLC

Illustrator: Rebecca Demarest

This book is dedicated to Barbara.

—Roberto Brunetti

This book is dedicated to my parents. Thanks!

—Paolo Pialorsi

This book is dedicated to my mother, Vanna, the strongest woman I have ever known.

—Luca Regnicoli

Contents at a Glance

	Introduction	*xi*
CHAPTER 1	**Introduction to Windows Store apps**	**1**
CHAPTER 2	**Windows 8 UI style**	**31**
CHAPTER 3	**My first Windows 8 app**	**65**
CHAPTER 4	**Application lifecycle management**	**99**
CHAPTER 5	**Introduction to the Windows Runtime**	**133**
CHAPTER 6	**Windows Runtime APIs**	**155**
CHAPTER 7	**Enhance the user experience**	**185**
CHAPTER 8	**Asynchronous patterns**	**231**
CHAPTER 9	**Rethinking the UI for Windows 8 apps**	**259**
CHAPTER 10	**Architecting a Windows 8 app**	**295**
	Index	*329*
	About the Authors	*341*

Contents

Introduction . . . . *xi*

Chapter 1 Introduction to Windows Store apps 1

The Windows 8 experience 1

Charms and App Bars 8

The Windows Runtime 14

Badges, Live Tiles, Toasts, and Lock Screen 15

Background tasks 20

Contracts and extensions 23

Visual Studio 2012 and Windows 8 Simulator 25

Summary 28

Quick reference 29

Chapter 2 Windows 8 UI style 31

Influences 31

Seeing the Bauhaus style in the Windows 8 UI 38

Characteristics of a Windows 8 app 41

Silhouette 41

Full screen 47

Edges 49

Comfort and touch 51

Semantic Zoom 56

Animations 58

Different form factors 58

Snapped and fill view 60

Summary 63

Quick reference 64

Chapter 3 My first Windows 8 app **65**

Software installation. 65

Windows Store project templates 66

Adding UI elements 75

Adding search functionality 86

Summary. 98

Quick reference 98

Chapter 4 Application lifecycle management **99**

Application manifest 100

Application package. 103

The Windows Store. 107

Launching. 111

Activation 118

Suspension 121

Resume 126

Summary. 132

Quick reference 132

Chapter 5 Introduction to the Windows Runtime **133**

Overview of the Windows Runtime 133

Windows Runtime under the covers 138

Windows Runtime design requirements. 142

Creating a WinMD library 143

Windows Runtime app registration 150

Summary. 154

Quick reference 154

Chapter 6 Windows Runtime APIs 155

Pickers .155
Webcam .163
Sharing contracts .171
Summary .183
Quick reference .184

Chapter 7 Enhance the user experience 185

Draw an application using Visual Studio 2012 .185
Create the layout of a Windows 8 application .189
Customize the appearance of controls .214
Summary .228
Quick reference .229

Chapter 8 Asynchronous patterns 231

await and async keywords for asynchronous patterns .231
Writing asynchronous methods .237
Wait for an event asynchronously .243
Handling exceptions in asynchronous code .244
Cancel asynchronous operations .246
Track operation progress .249
Synchronization with multiple asynchronous calls .253
Choose SynchronizationContext in libraries .257
Summary .258
Quick reference .258

Chapter 9 Rethinking the UI for Windows 8 apps **259**
Use Windows 8 UI-specific controls. 259
Designing flexible layouts 278
Using tiles and toasts 285
Summary. 294
Quick reference 294

Chapter 10 Architecting a Windows 8 app **295**
Application architecture in general 295
Architectures for Windows 8 apps. 298
Implementing the data layer. 299
Implementing the communication layer using a SOAP service 302
Implementing the communication layer using an OData service 306
Consuming data from a Windows 8 app 310
Implementing an app storage/cache 316
SOAP security infrastructure. 320
OData security infrastructure 324
Summary. 328
Quick reference 328

Index *329*

About the Authors *341*

Introduction

Windows 8 is Microsoft's newest operating system, intended to let developers fluent in various programming languages—such as C#, VB, C++, or JavaScript—leverage its powerful infrastructure with a brand new library, called the Windows Runtime API, to build successful applications.

This book provides an organized walkthrough of the Windows 8 features, APIs, and user experience. The text is definitely introductory; it discusses each component from a theoretical viewpoint interspersed with basic but effective code samples, which you can follow to get a jump start in developing for the Windows 8 platform.

The book provides coverage of almost all the main Windows 8 aspects and features, and it offers essential guidance for learning them using the classic Step-by-Step approach.

In addition to its coverage of core Windows 8 features using C# and XAML, the book discusses some related topics such as WCF Data Services, OData, ADO.NET Entity Framework, and applications architecture. Beyond the explanatory content, each chapter includes a rich set of step-by-step examples, as well as downloadable sample projects that you can explore by yourself.

Who should read this book

This book's goal is to provide developers conversant with .NET programming the experience they need to begin working with the main components of the Windows 8 operating system and Windows Runtime. Starting with the Windows Runtime APIs, the book drives the reader into a comprehensive discussion on the new user experience—including how to design for keyboard, mouse, and touch screen interfaces. A solid knowledge of the .NET Framework is helpful to understand the code presented in the book fully, and to follow along, perform the exercises using Microsoft Visual Studio 2012. This book is also useful for software architects who need an overview of the components they would plan to include in the overall architecture of a real-world Windows 8 solution.

Who should not read this book

If you have worked with Windows 8 already, this book is probably not for you; this is an introductory guide to developing applications that leverage the platform.

Assumptions

To get the most out of this book, you should have at least a minimal understanding of .NET development and object-oriented programming concepts. Although you can develop for Windows 8 using all .NET languages—as well as C++ and JavaScript—this book includes examples in C# only in the text, but includes Visual Basic samples in the downloadable companion code.

If you have not yet picked up C# or Visual Basic, you might consider reading John Sharp's *Microsoft Visual C# 2012 Step by Step* (Microsoft Press, 2012).

In addition to a .NET language, the examples on application architecture chapter assume you have a basic understanding of ASP.NET and Windows Communication Foundation (WCF), although the presented code doesn't use any advanced features of either of those two technologies.

Organization of this book

This book is divided into 10 chapters, each of which focuses on a different aspect or technology within the Windows 8 operating system and the Windows Runtime APIs.

Finding your best starting point in this book

We suggest that you start reading the book from the beginning. By following this path, you will discover all the aspects of the new look and feel, the new user experience, and new user interface for touch-based devices that are required for building successful Windows 8 applications. Chapter 2 is particularly important because you need to understand the design concepts underlying the Windows 8 UI style. Chapter 3 is the fundamental starting point for building your first Windows 8 application. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to Windows 8 development	Start with Chapter 1
New to Windows 8 UI style	Start with Chapter 2
Not new to Windows 8 development using the provided templates	Start with Chapter 4
XAML developer	Start with Chapter 3 and then skip to Chapter 9 to gain a solid understanding of the controls that are specific to Windows 8 apps and how to design flexible layouts.

Most of the book's chapters include hands-on procedures and examples that let you try out the concepts discussed in each chapter. No matter which sections you choose to focus on, be sure to download the companion code from the publisher's site (see the "Code samples" section of this Introduction), and install them on your system.

Conventions and features in this book

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

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

System requirements

You will need the following hardware and software to complete the practice exercises in this book:

- Windows 8, installed
- Visual Studio 2012—any edition tailored for Windows 8 (the Express edition for Windows 8 is free)
- A computer with a 1.6 GHz or faster processor
- 1 GB of RAM (1.5 GB if running on a virtual machine)
- 10 GB (NTFS) of available hard disk space
- 5400 RPM (or faster) hard disk drive
- DirectX 9-capable video card running at 1024 x 768 or higher display resolution

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2012.

Code samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All the sample projects are available for download from the book's page online:

http://aka.ms/666955/files

Note In addition to the code samples, your system must have Microsoft Visual Studio 2012 installed.

Installing the code samples

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

1. Unzip the 9780735666955_files.zip file that you downloaded from the book's website (name a specific directory along with directions to create it, if necessary).
2. If prompted, review the displayed end user license agreement. If you accept the terms, select the accept option, and then click Next.

Note If the license agreement doesn't appear, you can access it from the same webpage from which you downloaded the 9780735666955_files.zip file.

Acknowledgments

We'd like to thank all the people who have supported us in writing this book.

Marco Russo has shared with all of us in the most important phases of writing this book and its twin, *Building Windows 8 Apps with Microsoft Visual C++ Step by Step.*

Vanni Boncinelli tested all the code samples we wrote in C# and adapted each sample to Visual Basic.

Errata and book support

We've made every effort to ensure the accuracy of this book and its companion content. If you do find an error, please report it on our Microsoft Press site:

1. Go to *www.microsoftpressstore.com*.
2. In the Search box, enter the book's ISBN or title.
3. Select your book from the search results.
4. On your book's catalog page, find the Errata & Updates tab
5. Click View/Submit Errata.

You'll find additional information and services for your book on its catalog page. If you need additional support, please e-mail Microsoft Press Book Support at *mspinput@microsoft.com*.

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

We want to hear from you

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

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

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

Stay in touch

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

CHAPTER 1

Introduction to Windows Store apps

After completing this chapter, you will be able to

- Understand the main features of a Windows Store app.
- Evaluate the key benefits of creating an app for Microsoft Windows 8.
- Recognize the main capabilities and features of the new Windows 8 operating system.

This chapter provides an overall introduction to Windows 8 and the new world of the Windows Store apps from a developer perspective. In this chapter you will learn the basics of the Windows 8 user interface (UI), as well as gain an overview of the new features and capabilities that this new platform provides. The chapter targets any developer—even those who have not yet seen Windows 8. You will also learn how to set up a development environment for building your own Windows 8 apps.

The Windows 8 experience

Windows 8 is one of the most innovative and revolutionary operating systems investments made by Microsoft in the last decade. Before Windows 8, the operating systems market was divided into at least three main families: server operating systems, client/desktop operating systems, and mobile/tablet-oriented operating systems.

Windows 8, together with its sibling on the server side, Windows Server 2012, introduces a new paradigm where the client/desktop OS and the mobile/tablet-oriented OS can be exactly the same, sharing features, capabilities, user interfaces, and behaviors. In the last few years, there has been an explosion of tablet devices, and the number of people working at home and in their offices using the same small tablet devices is increasing. Nevertheless, until the release of Windows 8, it was not so simple to combine the preferences and needs of users with the infrastructure constraints of corporate networks. For example, employees would like to be able to install software on their own tablets, taken from a more-or-less checked and trustable marketplace available on the Internet, regardless of the corporate policies of their companies. Moreover, these employees would like the ability to check their corporate email accounts, as well as any private email accounts, using a unique device and unique email client software. Furthermore, the emerging social-oriented consumption of devices leads to the sharing of private contacts, agendas, tasks, pictures, and instant messages through business contacts, meetings, and corporate network instant communication and video-conferencing.

However, technology without governance could become a nightmare both for users and IT professionals. With Windows 8, employees can leverage a corporate-provided tablet device that allows them to install their choice of software from a safe and secure marketplace, either publicly or corporately constrained. Using this single device, they can check multiple email accounts or socialize with friends, colleagues, and business contacts—all while remaining compliant with their employer's security policies within a safe and sandboxed environment.

Moreover, for the sake of backward compatibility, most of the software targeting Windows 7 desktops will still continue to work on Windows 8, using the old-style desktop-oriented approach.

So, let's explore the new Windows 8 UI and the key features of this new operating system. Figure 1-1 shows the new Start screen, which is one of the most apparent changes introduced with Windows 8.

FIGURE 1-1 The new Windows Start screen.

As shown in Figure 1-1, the new Start screen is composed of a set of squares and rectangles, called *Tiles*, each of which represents a link to a software application, and can provide animated feedback to users. Tiles can be either square or wide tiles. Many apps provide both sizes so users can choose the one that best suits their needs. For example, in the upper-left corner of Figure 1-1, just under the Main title, there's a wide tile for the Mail App, which indicates that there are 15 email messages in the inbox. The tile also provides a brief preview of the messages.

To reduce the size of the tile you can right-click it, or swipe down on the tile, which selects it and activates a command bar, called the App Bar, which will be discussed later. Figure 1-2 shows how the Mail App tile looks after it has been selected.

FIGURE 1-2 The Mail App tile is selected, and the App Bar is visible.

Several commands are available in the App Bar. For example, you can select the Smaller command to reduce the tile's size from wide to square. You can also turn off the dynamic update feature of the tile, by clicking Turn Live Tile Off, or you can click Uninstall to remove the app from your device. If you click the Smaller command, the tile becomes square and the preview of unread email disappears (see Figure 1-3).

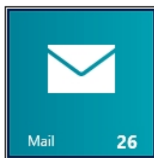


FIGURE 1-3 The Mail App tile after clicking the Smaller command on the App Bar.

A user with a tablet device can tap (that is, touch using a single finger) a tile to start an application instance or to resume an already running instance. A user with a desktop PC and a mouse can click the tile to get the same result. The Start screen is based on the idea of the panorama view, which has been available in the Windows Phone since version 7.0. You can scroll horizontally, using either touch gestures on a touch-enabled device or the mouse wheel, or if you are working on a desktop, the keyboard. You can also use the traditional scrollbar that appears at the bottom of the screen.

As soon as you tap an app tile, that app will become the foreground application. If you are starting that app for the first time in a given session, Windows will create and load the app instance in memory. Subsequently, when the app is already running, tapping the app tile switches that app to the foreground application. In both cases, the previous application is sent into the background and may eventually be suspended by the operating system. Suspension means freezing; a suspended app uses no CPU threads and no I/O functionality is provided to the application, leaving all the computer resources to the main (foreground) application. When you return to a suspended application, the operating system resumes it in its previous state. In Chapter 4, "Application lifecycle management," you will learn more about the application lifecycle for Windows Store apps. Figure 1-4 shows the Bing Weather App running in the foreground.

FIGURE 1-4 The Bing Weather App running in the foreground.

By default, an app uses the entire screen, in order to satisfy one of the main concepts of the user experience design of Windows Store apps: “content, not chrome.” In Chapter 2, “Windows 8 UI style,” you will discover more about exactly what user experience design means.

Not all apps that run under Windows 8 are Windows Store apps. If you start an old-style desktop application, you will see the classic and familiar Windows Desktop UI, just as if you were running a previous version of Windows. Figure 1-5 shows an old-style application, in this case SQL Server Management Studio, running in desktop mode. Notice the absence of the classic “Start” button.

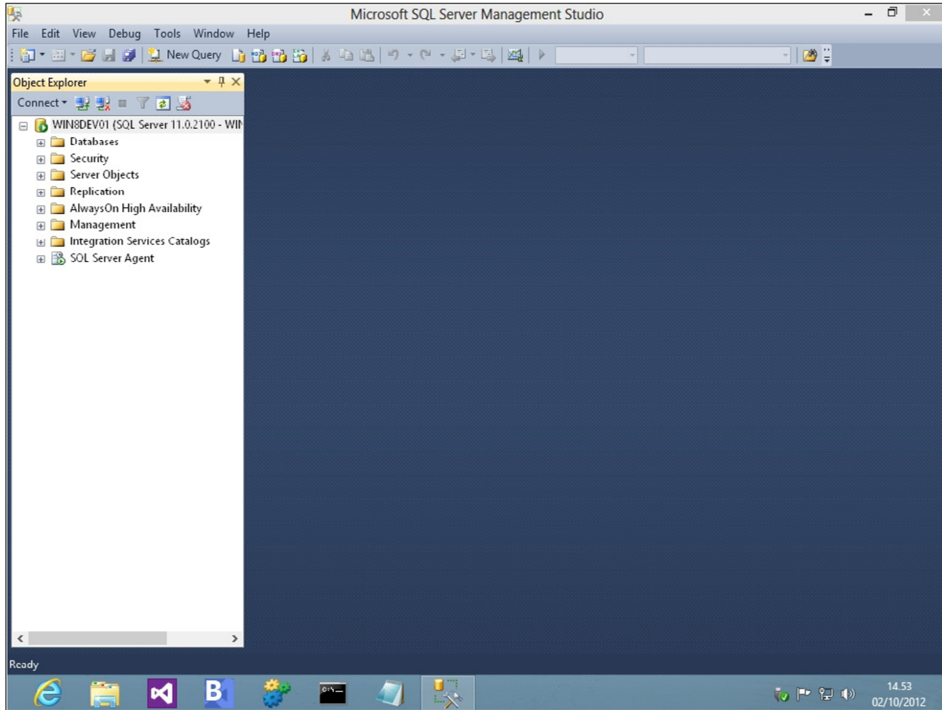


FIGURE 1-5 SQL Server Management Studio running in the classic desktop mode.

You aren’t always limited to a single full-screen application, however. If you have a device with a 1366 × 768 or higher resolution, you can leverage the Windows 8 capability to “snap” two applications into the display area. Figure 1-6 shows the Bing Weather App snapped together with the new Microsoft Internet Explorer 10 for Windows 8.

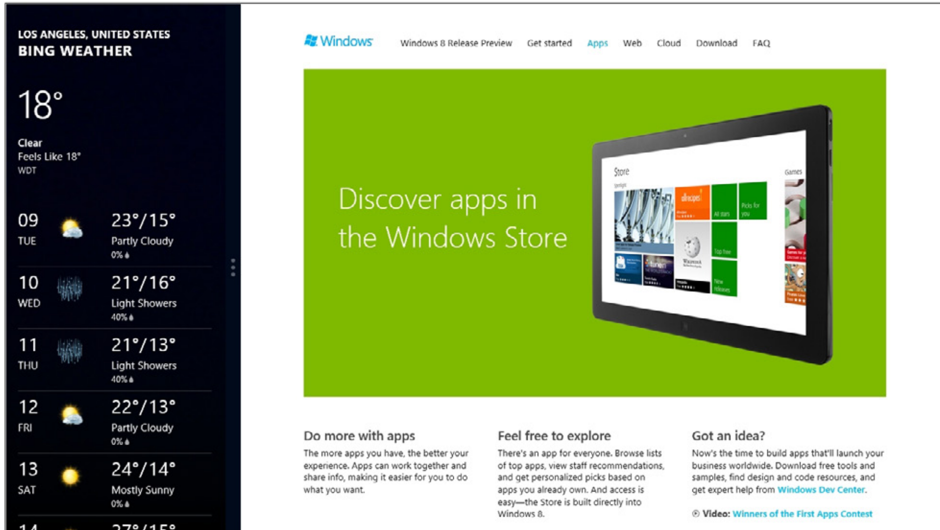


FIGURE 1-6 The Bing Weather App snapped together with Internet Explorer 10.

Of course you can also switch the relative sizes of two snapped apps, as shown in Figure 1-7.

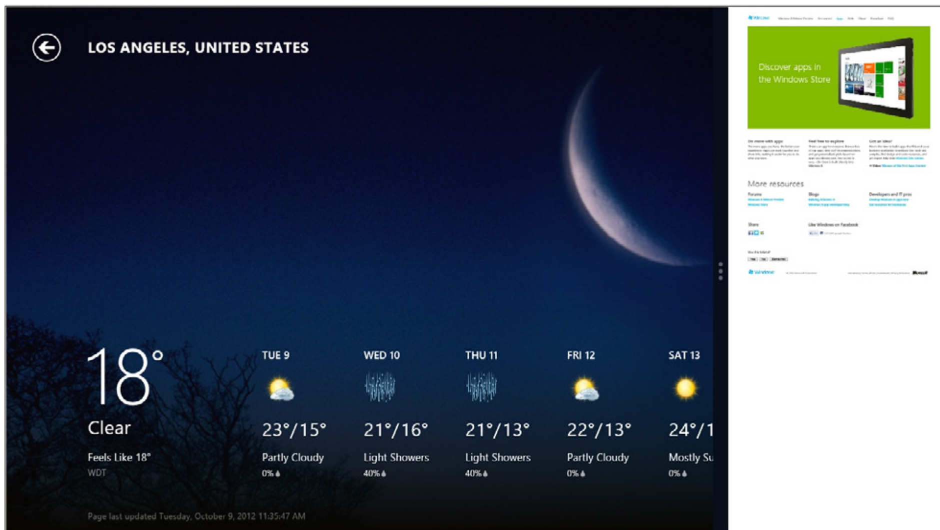


FIGURE 1-7 An example of switching two app panes, with the Weather App in the larger pane and Internet Explorer 10 in the smaller pane.

From a developer perspective, the important thing to understand and master is that every Windows Store app must support snapping; otherwise, it won't be certified by the Windows Store. The Bing Weather App, as shown in the previous figures, supports the snapped view. When snapped, it adapts its page layout to present information in a small horizontal portion of the screen. If your apps are unable to present information in a snapped view, you must fill the snapped pane where your application would be with a clear message for the user. You should never use the "full-screen" view for a snapped view because the user would not be able to interact properly with the application.

In fact, whenever you want to develop and publish a Windows Store app you have to submit it to the Windows Store, or eventually to a corporate catalog. From the official and public Windows Store viewpoint, an app must adhere to a clear set of requirements to be certified. Any application that does not adhere to these requirements will be rejected. You can find more details about the requirements in the Windows 8 developer section of MSDN (Microsoft Developer Network): *http://msdn.microsoft.com/library/windows/apps/hh694083.aspx*. For example, one rule states that you have to provide a privacy information page if your app connects to the Internet for any purpose. Thus, if your app invokes a remote web service, which is a common situation, you must provide a privacy page illustrating how you manage users' data. In Chapter 4, you will learn how to submit an app to the Windows Store.

Turning the focus back to the Start screen, another useful feature is that you can collect tiles into groups to organize them better in the menu. To move a tile from one group to another you just drag it, using touch gestures or the mouse. To create a new group you need to move a tile into the middle region between two existing groups. A gray bar will appear that represents the frame of the new group, and dragging the tile onto this gray bar will create the new group. By using a specific gesture (pinch) that will be explained in Chapter 2, or rolling the mouse wheel backward while pressing the Ctrl key, the Start screen zooms out so you can see more groups. By clicking a group, or swiping your finger down on a group to select it, you can give that group a name using a command in the App Bar. In Figure 1-8, you can see the UI of the Start screen while zoomed out, with a group of tiles selected and the App Bar showing the available commands.

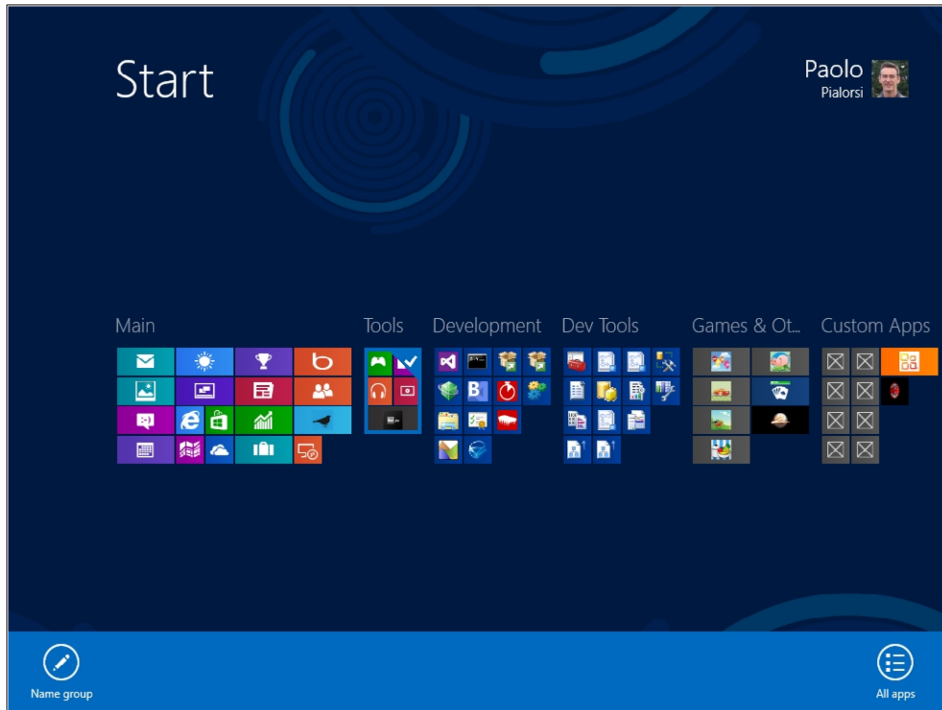


FIGURE 1-8 The Start screen zoomed out with the App Bar available.

Charms and App Bars

Other new and key features of Windows 8 are the App Bars and Charms. In Chapter 2, you will see more information about these features and the philosophy behind them. For now, simply consider that the need to support new devices, such as tablet and mobile devices, and the need to make the apps usable with just your hands, introduced new tools through the new touch-oriented perspective and solutions. In Windows 8 you have two kinds of App Bars: the bottom App Bar and the top App Bar. As their names indicate, these two kinds of App Bars are shown, respectively, in the lower and upper regions of the screen. Using the bottom App Bar, you manage tasks and actions related to the current context or item. Figure 1-9 shows an example with Internet Explorer 10, where you use the bottom App Bar to edit the current URL or enter a new URL, refresh the page, pin the page to the new Start screen, or change browser settings.

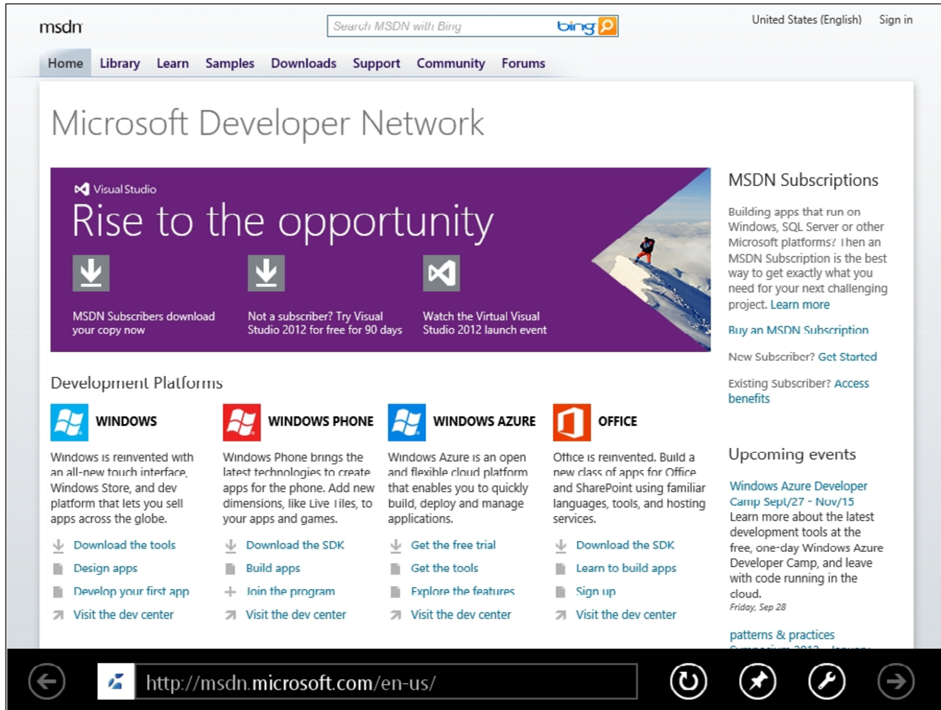


FIGURE 1-9 Internet Explorer with the bottom App Bar showing the URL.

In contrast, the top App Bar is used to provide navigation aids. For example, you can use it to show such things as a top-level menu or a list of main sections available in the current app. Figure 1-10 shows the top App Bar of the Windows Store app, which is an app you can use to search, download, buy, and install other apps.

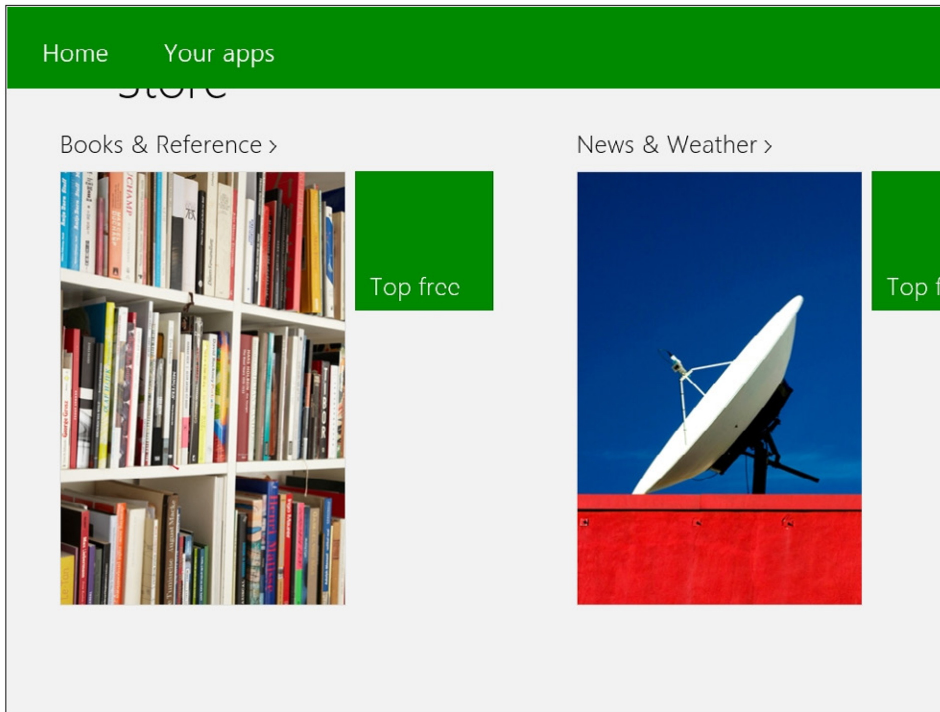


FIGURE 1-10 The Windows Store app with the top App Bar showing.

To show the App Bars, you can swipe your finger from the top or bottom border of the screen to the center of the screen. Alternatively, you can press Windows+Z on the keyboard, or right-click.

Charms allow you to access the most useful features and actions provided by the operating system. For example, you can use Charms to access system settings, the local search engine, the sharing features, and so on. Figure 1-11 shows Charms in action.

FIGURE 1-11 Charms are on the right side of the screen.

To display Charms, you can swipe your finger from the right border of the screen to the center of the screen, or you can press Windows+C. You can also move the cursor to either of two invisible "hot spots" in the lower-right or upper-right corner of the screen. Finally, you can directly activate specific Charms using keyboard shortcuts. For example, pressing Windows+Q activates a search for the installed applications (Q stands for query), whereas pressing Windows+F (F stands for Find Files) activates the search for files function. To activate the sharing feature, press Windows+H.

Through Charms you can also activate specific panels, such as the Settings Panel, which can be activated by pressing Windows+I. In Figure 1-12, you can see the Settings Panel in action.

FIGURE 1-12 The Settings Panel is visible on the right side of the screen.

One key feature of Charms is that you can also host custom commands and custom panels in it. For example, if you are developing a Windows Store app and you want to provide some custom settings for users, you can add a custom Charm. By selecting the custom command while the app is in the foreground, you can activate a *fly-out panel*, which is a custom control that renders within the Charms. Figure 1-13 shows the fly-out panel.

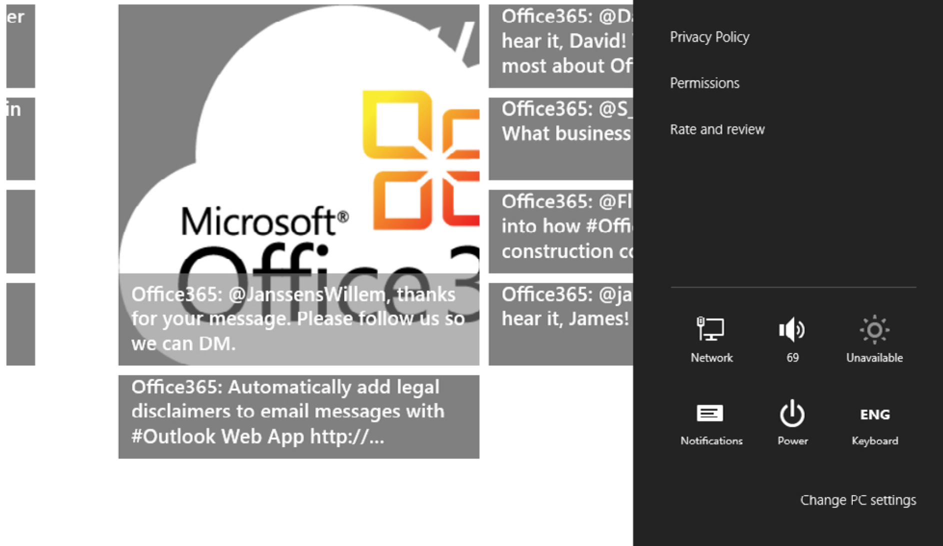


FIGURE 1-13 A custom fly-out panel rendered within Charms.

The Charms shown in Figure 1-13 provides Support Request and Privacy Policy commands, which are custom and specific to the app currently in the foreground. The latter command leads users to the privacy page required for any Windows Store app that consumes a remote service over the Internet, as you learned earlier in this chapter.

The Windows Runtime

A Windows Store app is a software solution that adheres to the UI and technical specifications of the Windows Store. You can create a Windows Store app using any language that supports the Windows Runtime (WinRT). The WinRT is a rich set of application programming interfaces (APIs) built upon the Windows 8 operating system, providing direct and easy access to all the main primitives, devices, and capabilities for any language available to develop Windows 8 apps. The WinRT is available only for Windows 8 apps and its main goal is to unify the development experience of building a Windows 8 app, regardless of the programming language you use.

Saying that you can use any language supporting the Windows Runtime means that, currently, you can choose from C++, .NET (C# or VB), and JavaScript. Nevertheless, there are no technical limitations to support the Windows Runtime from any other language, as long as it adheres to the Windows Runtime specifications.

In Chapter 5, "Introduction to the Windows Runtime," you will learn more about this topic and the architecture of the Windows Runtime. For now, you can imagine the Windows Runtime as an infrastructural framework of libraries that allows easy development of Windows Store apps, hiding all the inner details of the operating system from the common and everyday developer perspective. For instance, you can ask the Windows Runtime to open the webcam standard UI to capture photos or videos without knowing anything about the underlying driver or Win32 API.

For example, in the following code excerpt you can see how simple it is to capture a picture from the camera of your PC, using the C# language.

```
private async void TakePhoto_Click(object sender, RoutedEventArgs e) {

    var camera = new CameraCaptureUI();
    var img = await camera.CaptureFileAsync(CameraCaptureUIMode.Photo);
    if (img != null) {
        var stream = await img.OpenAsync(FileAccessMode.Read);
        var bitmap = new BitmapImage();
        bitmap.SetSource(stream);
        image.Source = bitmap;
    }
}
```

You can perform the same action using JavaScript, with the following code excerpt:

```
var dialog = new Windows.Media.Capture.CameraCaptureUI();
dialog.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.photo).done(function (file) {
          if (file) {
              var photoBlobUrl = URL.createObjectURL(file, { oneTimeOnly: true });
              document.getElementById("capturedPhoto").src = photoBlobUrl;
          }
      };
```

Moreover, you can achieve the same result using C++, as shown in the following code excerpt:

```
void CaptureWin8::MainPage::TakePhoto_Click(Platform::Object^ sender,
          Windows::UI::Xaml::RoutedEventArgs^ e) {


        CameraCaptureUI^ dialog = ref new CameraCaptureUI();
        concurrency::task<StorageFile^> (
            dialog->CaptureFileAsync(CameraCaptureUIMode::Photo)).then([this]
            (StorageFile^ file) {
            if (nullptr != file) {
                concurrency::task<Streams::IRandomAccessStream^> (
                file->OpenAsync(FileAccessMode::Read)).then([this] (
                    Streams::IRandomAccessStream^ stream) {
                        BitmapImage^ bitmapImage = ref new BitmapImage();
                        bitmapImage->SetSource(stream);
                        image->Source = bitmapImage;
                });
            }
        });
    }
```

Badges, Live Tiles, Toasts, and Lock Screen

Another set of new features found in Windows Store apps includes Badges, Live Tiles, Toasts, and the Lock Screen. Badges and live tiles show dynamic information to users even while they are not directly using the app providing the information—the tiles display such information directly on the Start screen. You can use a badge and/or a live tile to provide information about news, new items to check, new tasks to execute, or whatever else is meaningful for the user to best experience your app from the Start screen, without opening the application. For example, the out-of-the-box Mail App uses the badge to show the number of unread emails in the inbox, and a live tile to show a rotating list of excerpts from all the unread messages. Moreover, the Windows Store App notifies you through a badge about the number of updates available for apps you have installed. In Figure 1-14, you can see these badges and live tiles in action.

FIGURE 1-14 The Start screen with badges and live tiles in action.

Notice the number 4 in the bottom-right corner of the Windows Store app; this badge indicates that there are four pending updates. You can also see the badge with 15 in the bottom-right corner of the Mail app, indicating 15 new emails in the inbox. Furthermore, the Mail app uses a live tile to show an excerpt of the most recent unread emails, but a live tile can do even more. For example, a live tile can completely change its content in order to be dynamic and fresh and to trigger curiosity in the mind of the user. Figure 1-15 shows four different states that the tile of a single app can assume (the Bing Travel app that ships with Windows 8).

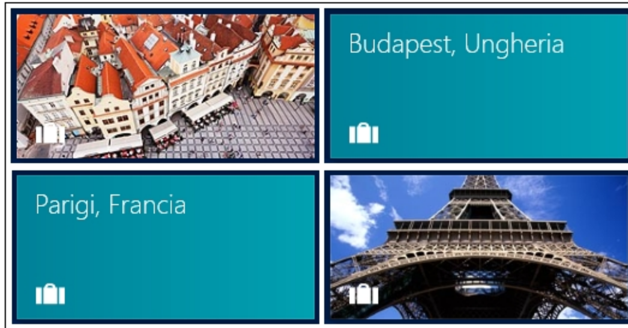


FIGURE 1-15 The Bing Travel App tile assuming four different states.

The official guidelines for Windows Store apps (see *http://msdn.microsoft.com/en-us/library/windows/apps/hh465403.aspx*) suggest using a wide tile only when your app has live tiles to show. Otherwise, you should use square tiles when your tiles contain static content, and simply use a badge for small and lightweight notifications. In Chapter 9, "Rethinking the UI for Windows 8 apps," you will learn how to create a live tile.

Toasts are another technique for providing asynchronous alerts to the user. For example, an alert or alarm application can ask the operating system to send to the user a toast at a preset time; the Windows Runtime will send the toast even if the application is not running at that time. Moreover, when users are working on an app in the foreground, background apps will not be able to interact with them unless the app uses a toast.

In fact, as you will see in Chapter 4, due to the architecture of Windows 8 and because of the application lifecycle management of Windows Store apps, only the foreground app has the focus and is running; all the other background apps can be suspended (or even terminated) by the Windows Runtime. A suspended app cannot execute or consume any CPU cycle. However, you can define a background task (more on this topic later in this chapter) that will work in the background, even in a separate process from the owner app, and you can define background actions. When these actions need to alert users about their outcomes, they can use a toast.

A *toast* can be plain text, an image, or any combination of the two. In the upper-right corner of Figure 1-16, you can see a toast provided by the Windows Store app informing the user that an app installation task has completed in the background.

FIGURE 1-16 An example toast message shown in the upper-right corner of the screen.

In Chapter 9, you will learn how to create a toast for your own Windows 8 apps.

One other option you possess while developing a Windows Store app is to provide lightweight information to the user through the Lock screen. The Lock screen is the screen that is shown when a Windows 8 user session is locked out, for example after a period of inactivity or when a user presses Windows+L to lock the session.

Figure 1-17 shows the Lock screen providing some information about the current date and time, the next appointment in the user's agenda, and a set of small icons, in the lower part of the screen.

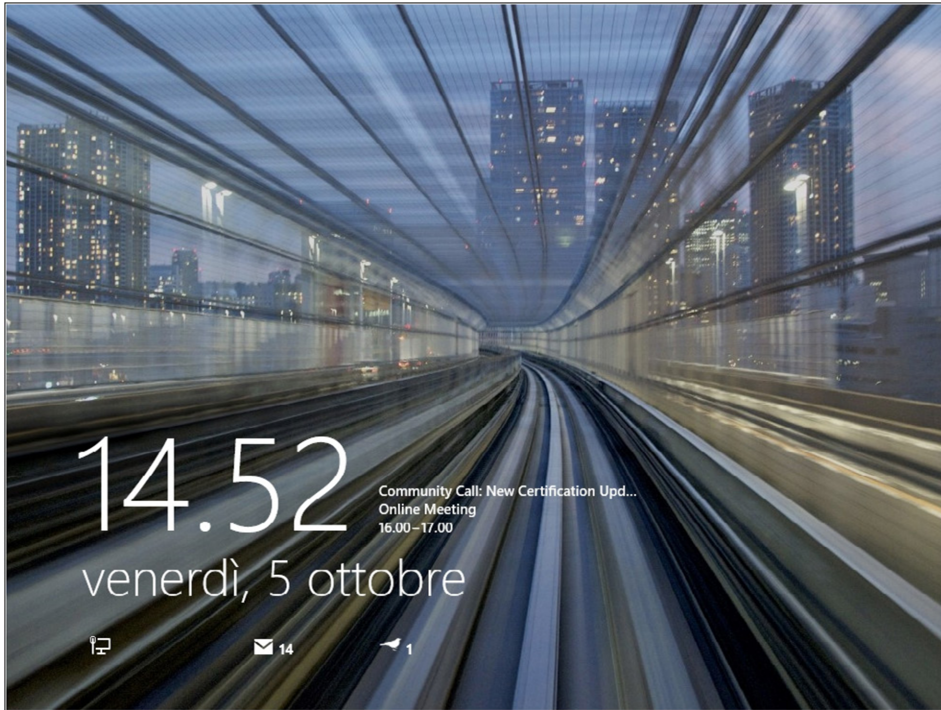


FIGURE 1-17 Lock screen showing status information.

Those icons provide information about the network connection status, battery status (for a device running on battery power), number of unread emails in the inbox, and some other lightweight information. A user can choose what information appears in the Lock screen by using the proper panel in the system configuration. However, you are limited to no more than seven Lock screen items simultaneously providing detailed information. All seven apps will be able to show badges and toasts in the Start screen, but only one of those apps will be allowed to show the text of its latest tile notification in the Lock screen. Figure 1-18 shows the configuration panel for the Lock screen. To reach it, you need to display the Charms; for example, press Windows+C, and then select the Settings command. Finally, click the Change PC Settings command. Under the Personalize section in the Lock screen tab, you will find the Lock screen configuration.

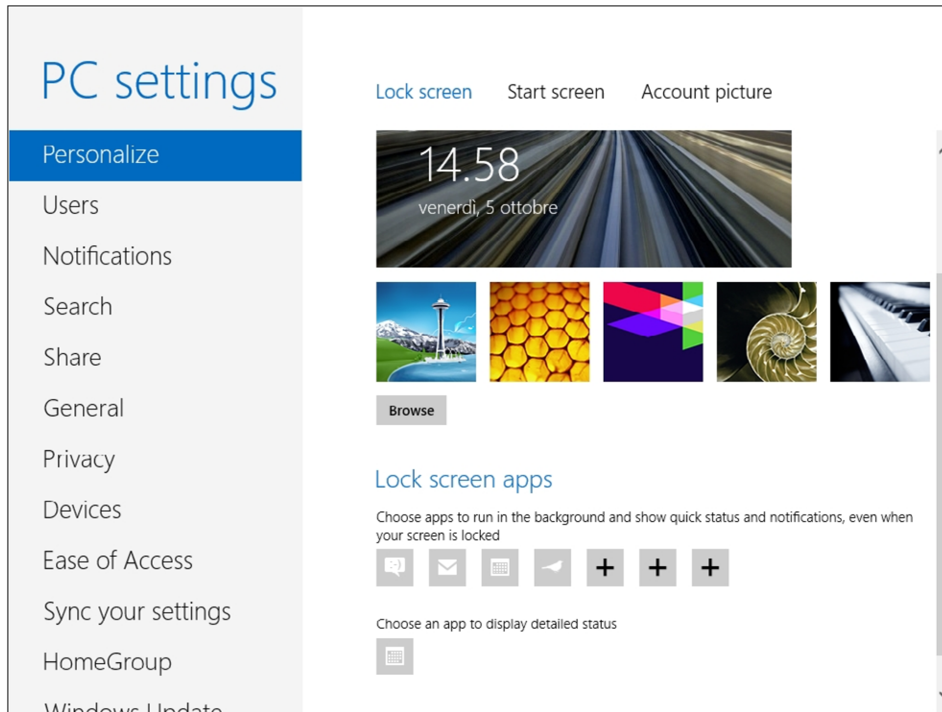


FIGURE 1-18 The Lock screen configuration panel in the PC Settings.

The Lock screen configuration allows you to choose a background image, select which seven apps will execute in the background to provide information through the Lock screen icons, and—last but not least—choose the app that will be allowed to display detailed text status. The last one, by default, is configured to be the Calendar app. For your apps to be available as Lock screen apps, your software must declare that capability within an *app manifest file*, which will be explained later in this book, starting with Chapter 3, "My first Windows 8 app."

The information shown by a Lock screen–enabled app is the same as the information provided by the app's tile on the Start screen. In fact, the text shown beside the Lock screen icon is taken from the badge of the app, whereas the detailed text status is taken from the tile text of the app.

Background tasks

As stated earlier in this chapter (and as will be explored more in Chapter 4), a Windows Store app executes code only when it is the foreground app. However, there are situations where you want to execute some code when your app is in the background. A background task can execute code even when the corresponding app is suspended, but it runs in an environment that is restricted and resource-managed. Moreover, background tasks receive only a limited amount of system resources. You should use a background task to execute small pieces of code that require no user interaction. You should *not* use a background task to execute complex business logic or calculations because

the amount of system resources available to background apps is both tight and limited. In addition, complex background workloads consume battery power, reducing the overall efficiency and responsiveness of the system.

To create a background task, you have to define a class and register it with the operating system. A background task is just a class that implements a specific interface (*IBackgroundTask* in C#, for example) defined by WinRT and that is registered by using a *BackgroundTaskBuilder* class instance. There are many types of background tasks available, and these respond to different kind of *triggers*, such as the following:

- ***ControlChannelTrigger*** Raised when there are incoming messages on the control channel.
- ***MaintenanceTrigger*** Raised when it is time to execute system maintenance tasks.
- ***PushNotificationTrigger*** Raised when a notification arrives on the Windows Notifications Service channel.
- ***SystemEventTrigger*** Raised when a specific system event occurs.
- ***TimeTrigger*** Raised when a time event occurs.

In particular, a *SystemTrigger* can occur in response to any of the following system events:

- ***InternetAvailable*** An Internet connection becomes available.
- ***LockScreenapplicationAdded*** An app tile is added to the Lock screen.
- ***LockScreenapplicationRemoved*** An app tile is removed from the Lock screen.
- ***ControlChannelReset*** A network channel is reset.
- ***NetworkStateChange*** A network change, such as a change in cost or connectivity, occurs.
- ***OnlineIdConnectedStateChange*** An online ID associated with the account changes.
- ***ServicingComplete*** The system has finished updating an application.
- ***SessionConnected*** The session is connected.
- ***SessionDisconnected*** The session is disconnected.
- ***SmsReceived*** A new SMS message is received by an installed mobile broadband device.
- ***TimeZoneChange*** The time zone changes on the device (for example, when the system adjusts the clock for daylight saving time).
- ***UserAway*** The user becomes absent.
- ***UserPresent*** The user becomes present.

Whenever such an event occurs, you can check a set of conditions to determine whether your background task should execute. The conditions you can check include the following:

- ***InternetAvailable*** An Internet connection must be available.
- ***InternetNotAvailable*** An Internet connection must be unavailable.
- ***SessionConnected*** The session must be connected.
- ***SessionDisconnected*** The session must be disconnected.
- ***UserNotPresent*** The user must be away.
- ***UserPresent*** The user must be present.

To optimize resource consumption, some trigger notifications are provided only to apps that have been included in the Lock screen. For example, a *TimeTrigger* can be leveraged only by an app in the Lock screen. The same requirement holds true for *PushNotificationTrigger* and *ControlChannelTrigger*. Even some of the *SystemTrigger* events are reserved for apps in the Lock screen, including events such as *SessionConnected*, *UserPresent*, *UserAway*, or *ControlChannelReset*. Because you should register for these events and triggers only if your application is in the Lock screen, you use the *SystemTrigger* events *LockScreenApplicationAdded* and *LockScreenApplicationRemoved* so that your app can register and unregister such triggers accordingly.

Generally speaking, in common language runtime (CLR) and C++ apps, you can execute a background task in the app itself or in a system-provided host (*BackgroundTaskHost.exe*). Additionally, you can also execute tasks for triggers of the type *PushNotificationTrigger* or *ControlChannelTrigger* in the app process.

One last topic to properly complete the introduction of background tasks is *resource management*. Every background task must execute its code using a constrained amount of CPU and network bandwidth. For example, each app on the Lock screen receives two seconds of CPU time every 15 minutes, plus two more seconds allotted to background task execution just after the previous two seconds. In contrast, apps that are not on the Lock screen receive one second of CPU time every two hours.

From a network bandwidth perspective, these constraints are a function of the amount of energy consumed by the network interface. For example, with a throughput of 10 Mbps, an app on the Lock screen can consume about 450 MB per day, whereas an app that is not on the Lock screen can consume about 75 MB per day.

These constraints are defined to reduce battery and resource consumption. It's worth noting that these rules do not apply to apps that rely on critical background tasks, such as *ControlChannelTrigger* and *PushNotificationTrigger*. Instead, these kinds of tasks receive guaranteed resources. Finally, there is a global pool of resources (CPU and network) that is shared across apps and can be used to provide some extra resources to those apps that need them. Of course, an app should not rely on such resources being available because they are shared between *all* background tasks for any app—in other words, another app could have already consumed all the global pool resources. The global pool is refilled every 15 minutes, with a refill quota related to the power source of the device (AC adapter or battery).

Contracts and extensions

Another powerful set of features available for developing Windows Store apps are *WinRT Contracts*. The Windows Runtime and Windows Store apps can share data, information, features, and behaviors through shared *communication contracts*. A contract is an agreement between an app and the Windows 8 operating system that allows an app to talk to and exchange data with any other app, without directly knowing anything about the other app, using the operating system and WinRT as a proxy.

For example, launch the Bing Travel app from the Start screen and navigate to a target travel location, such as Rome in Italy. Then show the Charms (Windows+C) and select the Share command. You will see a fly-out panel within the Charms that lets you select how you want to share that location: by email, to friends using the People app, or via any other Windows Store app configured as a sharing target for the current content. Figure 1-19 shows an example of this process.

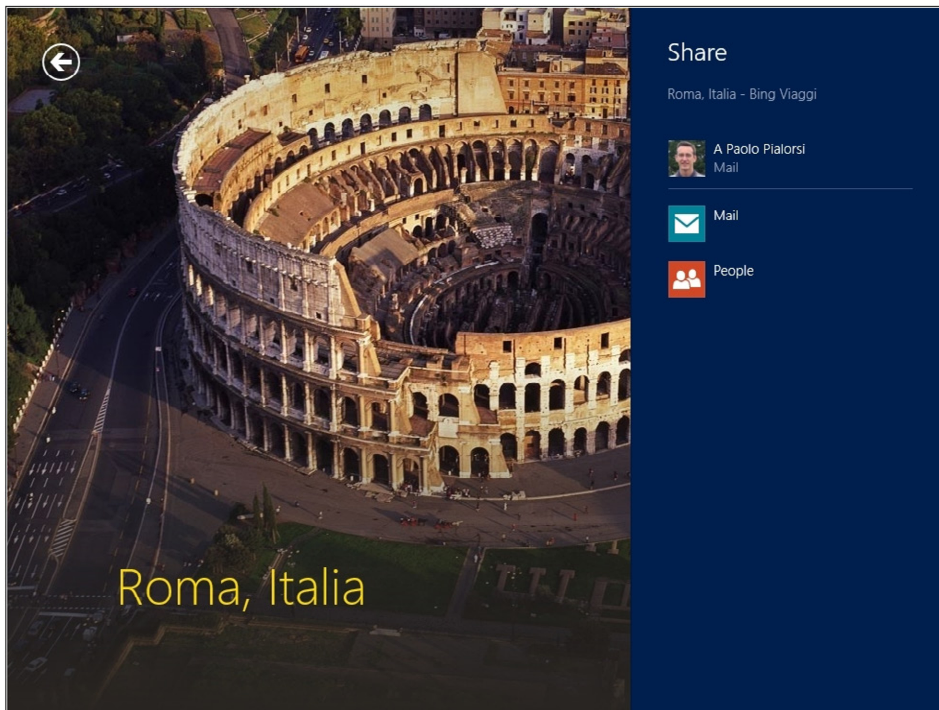


FIGURE 1-19 Sharing a location by taking advantage of a communications contract baked into the Bing Travel app.

As soon as you have made a choice, for example by selecting Mail, Windows will take you into the sharing target app, and you can handle the shared content there. For example, Figure 1-20 shows how you can send the Rome information to someone via email in Windows Mail.

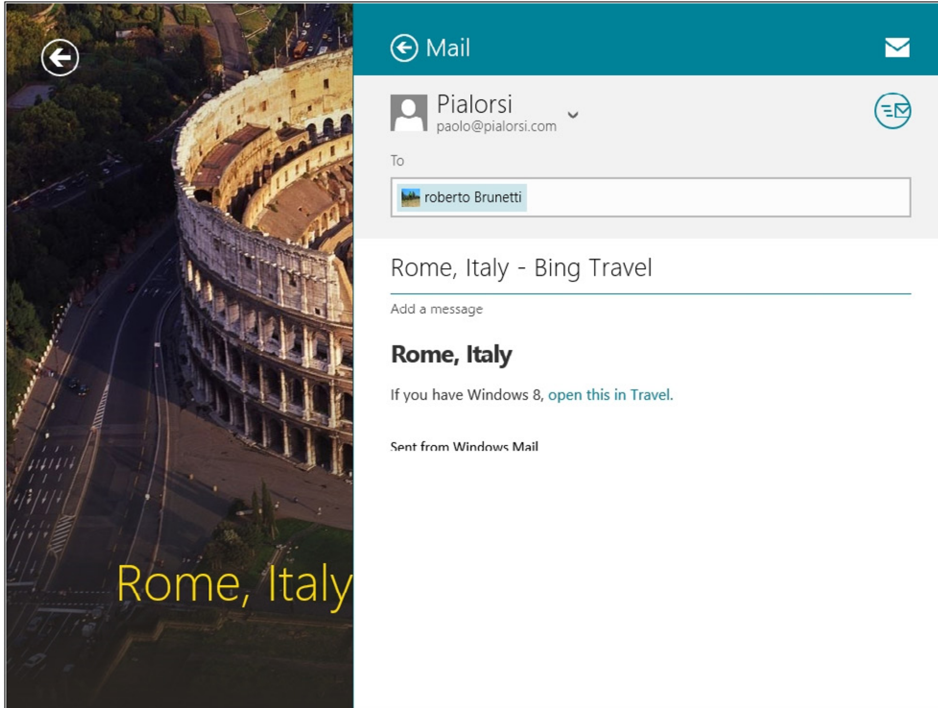


FIGURE 1-20 Sharing Rome information by email via the Windows Mail app.

It's worth reiterating that neither of the apps involved in this sharing transaction (Bing Travel or Windows Mail) is aware of the other. The Windows Runtime, sitting in the middle, joins them through a contract called a Share contract.

Similarly, when you are using an app such as the Windows Store app, and you activate the search feature (Windows+Q), the operating system uses a Search contract to query the Windows Store app for apps that satisfy the search criteria provided.

The Windows Runtime exposes a rich set of contracts, as shown in the following list:

- **Cached File Updater contract** You can leverage this contract to keep track of file changes and cache them. For example, an app like SkyDrive uses this contract to monitor file changes.
- **File Picker contract** You can register your app as a target for the File Picker UI.
- **Play To contract** This allows your app to be enlisted in the list of apps available in the Play To section of the Connect command in the Charms.
- **Search contract** This provides search capabilities to your app.
- **Settings contract** This contract provides a panel for custom settings of your app.
- **Share contract** This contract shares content between apps.

There are also extensions that allow an app to adhere to an agreement with the operating system instead of with a third-party app. You can use these extensions to extend standard Windows features. For the sake of simplicity, consider what happens when you connect a new device or insert a disk into the CD/DVD reader. An operating system message appears that informs users that they can play the new device or media, providing a list of available actions and players. For example, you can register your app as supporting the *AutoPlay* extension, and subsequently your app will be listed in the list of available autoplay targets.

You can see an enumeration in the following list:

- **Account picture provider** When a user changes his or her account picture, you can register an app as an account picture provider.
- **AutoPlay** The app will be listed as an autoplay target.
- **Background tasks** The app can run background tasks.
- **Camera settings** The app provides custom UI for camera settings.
- **Contact picker** The app is registered as a contact picker provider.
- **File activation** The app is registered as being associated with a specific file type based on the file extension.
- **Game Explorer** You can register the app as a game, providing a Game Definition File (GDF), and your app will be available as a game only if compliant with the target machine's family safety rules.
- **Print task settings** This declares that your app has a custom printer UI and can print by talking directly to a printer device.
- **Protocol activation** You can register a protocol moniker associated with your app. For example, Windows Mail can be activated with a *mailto:* protocol moniker. Internet Explorer 10 can be activated with an *http:* protocol moniker. You can register your own moniker and use it to activate your app.
- **SSL/certificates** Enable your app to install a digital certificate onto the target device.

As you will see in Chapter 3, registering or consuming a contract through WinRT is very straightforward.

Visual Studio 2012 and Windows 8 Simulator

To develop a Windows Store app, you will need to install a development environment such as Microsoft Visual Studio 2012. To accomplish this task, you can buy and install a regular license of Microsoft Visual Studio 2012 directly from Microsoft or from an authorized reseller. However, you can also get started by downloading and installing a free edition of Visual Studio 2012, called Visual Studio 2012 Express edition. In particular, the Express family contains one product named Visual Studio 2012 Express for

Windows 8. Using this development tool, you can create Windows Store apps by starting from scratch or starting with a set of prebuilt application templates and models. You can download Visual Studio 2012 Express for Windows 8 from the Microsoft website at *http://www.microsoft.com/visualstudio/*, or you can find it in the Windows Store app, under the "Tools" app category. Figure 1-21 shows the page dedicated to Visual Studio 2012 Express for Windows 8 in the Windows Store app.

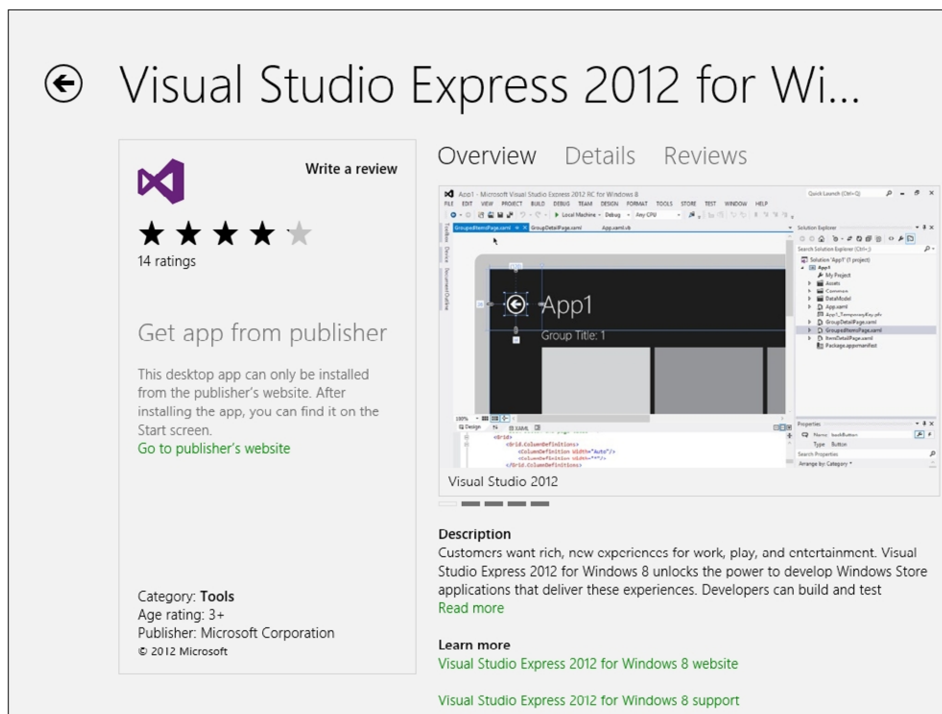


FIGURE 1-21 The Visual Studio Express 2012 for Windows 8 page in the Windows Store app.

After installing Visual Studio Express 2012 for Windows 8, you will be able to create custom apps and publish them to the Windows Store—a process discussed in much more detail in Chapters 3 and 4.

Note that you can download and install a retail version of Microsoft Visual Studio 2012 (that is, Professional, Premium, or Ultimate) even on previous editions of Windows. For example, suppose you don't have a Windows 8 PC; instead, you have a machine running Windows 7. You can still install Visual Studio 2012 and develop your software solutions on Windows 7, but you will not be able to develop Windows Store apps on it.

Note You cannot download and install the free Microsoft Visual Studio 2012 Express for Windows 8 edition on a computer without Windows 8; that edition *requires* you running Windows 8 or later.

One useful option for testing and executing your apps is to use the Windows 8 Simulator, which is part of the Windows 8 SDK included in Visual Studio 2012.

Figure 1-22 shows the Windows 8 Simulator in action.

FIGURE 1-22 The Windows 8 Simulator.

As you can see, the simulator looks like a small tablet PC running Windows 8. On the right side there is a set of commands to simulate various scenarios. These commands are, from top to bottom:

- **Always on top** Puts the simulator always on top.
- **Mouse mode** When you move and click your mouse, the simulator will react to mouse interactions as well.
- **Basic touch mode** Your mouse pointer will become like a finger and when you click the simulator it will be handled as a finger touch.
- **Pinch/zoom touch mode** Similar to the previous option, but used to simulate zoom in and zoom out via touch gestures.
- **Rotation touch mode** Similar to the previous option, but used to simulate touch rotation gestures.
- **Rotate clockwise (90 degrees)** Rotates the device clockwise 90 degrees.
- **Rotate counterclockwise (90 degrees)** Rotates the device counterclockwise 90 degrees.

- **Change resolution** Changes the screen resolution of the simulator device. The available resolutions are:
  - 10.6″ 1024 × 768
  - 10.6″ 1366 × 768
  - 10.6″ 1920 × 1080
  - 10.6″ 2560 × 1440
  - 12″ 1280 × 800
  - 23″ 1920 × 1080
  - 27″ 2560 × 1440
- **Set location** Allows simulating a GPS location for testing location-based apps.
- **Copy screenshot** Creates a screenshot of the simulator screen. This is useful for creating promotional pictures of your apps and the required images to publish an app on the Windows Store.
- **Screenshot settings** Configures copy screenshot behavior, such as the destination directory of the image files.
- **Help** Provides a link to the simulator's Help.

Using the Windows 8 Simulator, you can test your apps fully, even without a real tablet device or a Windows 8 environment.

One of the most important features of the simulator is the ability to change the resolution, orientation, and form factor of the screen to test the application behavior for many different "devices" without the need to buy real ones.

Also, remember that you cannot develop a Windows Store app using Microsoft Visual Studio 2010 or any other earlier edition of the product. The only edition of Microsoft Visual Studio suitable for developing Windows Store apps is Visual Studio 2012 or later.

Summary

In this chapter, you have been introduced to some basic information about Windows 8 and Windows Store apps. You learned the key new features of the Windows 8 UI, as well as the main goals behind the development of a Windows Store app. You saw several apps and features, including the Windows Store, badges, live tiles, toasts, background tasks, the new Lock screen, the new Start screen, and more. You also learned about the development environment required to develop Windows Store apps.

Quick reference

To	Do this
Notify a user of an action happening in the background	Use a toast, a badge, or a live tile. You can also use the Lock screen, in case it is suitable for your context.
Execute some code while the app is suspended	Use a background task.
Make the contents managed by your app searchable by the user	Support the Search contract.
Develop a Windows Store app	Install Microsoft Visual Studio 2012 Express edition for Windows 8 or Microsoft Visual Studio 2012 on a Windows 8 device.
Simulate the execution of a Windows 8 app in different resolutions, orientations, and form factors	Run the Windows 8 Simulator available within Visual Studio 2012.

CHAPTER 3

My first Windows 8 app

After completing this chapter, you will be able to

- Install and use the Microsoft Visual Studio 2012 tools to develop a Windows 8 app.
- Understand and use the Project template.
- Create a simple application using C# and Visual Basic (VB).
- Test the application.
- Use the Windows 8 Runtime (WinRT) APIs from a Windows 8 application.

The preceding chapters showed you how Microsoft Windows 8 provides a new user interface, a completely new user experience, and exposes a new set of application programming interfaces (APIs) called Windows Runtime APIs (WinRT). The new user interface and experience is based around the Windows 8 UI style you just learned about in Chapter 2, "Windows 8 UI style."

This chapter translates what you saw into practice. You will start by creating a simple Windows 8 app from scratch using one of the templates provided by Visual Studio 2012. Then you will deploy it to the local machine. Finally, you will implement a simple call to some WinRT APIs.

Software installation

To start developing Windows 8 applications, you need Visual Studio 2012. This new version of Visual Studio can be installed to run side by side with an existing Visual Studio 2010 installation and contains the .NET Framework version 4.5. The .NET Framework 4.5 is not a major release but it does contain some important features that enable the use of WinRT APIs. Even though you can develop applications using other versions of Windows and deploy them to a Windows 8 box or test it in the provided emulator, we suggest you install the development environment directly on a machine with Windows 8. This will speed up the development and testing processes on hardware-related components: for instance, if your apps use the accelerometer, the inclinometer, the camera, or any other sensor, the testing and debugging phase will be more accurate and quicker.

To download Windows 8, go to *http://msdn.microsoft.com/windows/apps*—the home page for the Windows 8 app development. From this page, it is easy to reach all the downloads for Windows 8. In the Getting Started section, you can find useful information for the download and installation process.

Note Because URLs and component packaging may vary over time, start looking for Windows 8 and Visual Studio 2012 on the Windows 8 home page (*http://msdn.microsoft.com/windows/apps*) or search for it on Bing (*http://www.bing.com*).

As you saw in Chapter 1, "Introduction to Windows Store apps," Visual Studio 2012 Express for Windows 8 is a free version of Visual Studio tailored to contain just what you need to develop a Windows 8 app. You can also use the full version of Visual Studio 2012 by installing it on top of the Express edition or you can keep it as a separate installation.

To summarize, the components you need to start developing a Windows 8 app are the following:

- **Visual Studio 2012 Express edition for Windows 8** On top of this version, you can install a more advanced edition of Visual Studio 2012 (for instance the Ultimate edition).
- **The Windows 8 SDK** To obtain the templates and the integration with the Windows 8 environment, this component is packaged together with Visual Studio 2012 Express for Windows 8.
- **Windows 8** You'll need this to test the application in the real environment.
- **A developer license** The integrated development environment (IDE) handles this requirement automatically and all you need to do is select Yes when the dialog box pops up.

Windows Store project templates

The easiest way to start developing a Windows 8 application is to use one of the out-of-the-box project templates that are available. Visual Studio 2012 provides a group of templates called "Windows Store" templates to develop an application for the Windows Store. These templates create all the files you need in the project to start developing, testing, and deploying the application on your local machine and the emulator, and they include a procedure to create an application package suitable for the Windows Store.

Each supplied template provides a solid starting point so that you can begin developing different kinds of Windows Store applications. The following list summarizes the characteristics of the various templates:

- **Blank App (XAML)** This template provides a minimal skeleton using Windows Store frameworks.
- **Grid App (XAML)** This template provides a multipage project for navigating multiple layers of content. The item details can be reached by tapping or clicking on the item itself and are displayed on a dedicated page.
- **Split App (XAML)** This template is a good starting point to create a master details list of items using a list on the left of the page and the details directly shown in the right of the same page.

- **Class Library (Windows Store apps)** The resulting project is the classic class library that can be used to centralize the code for Windows Store applications. This template can also be used to create a Windows Runtime component.
- **Windows Runtime Component** This allows the development of a component that can be used by Windows Store applications, regardless of the programming languages in which the app is written.
- **Unit Test Library (Windows Store apps)** The goal for this template is to create a project that contains unit tests to be used with Windows Store apps, Windows Runtime components, or class libraries for Windows Store apps.

In the following procedure, you'll create a project.

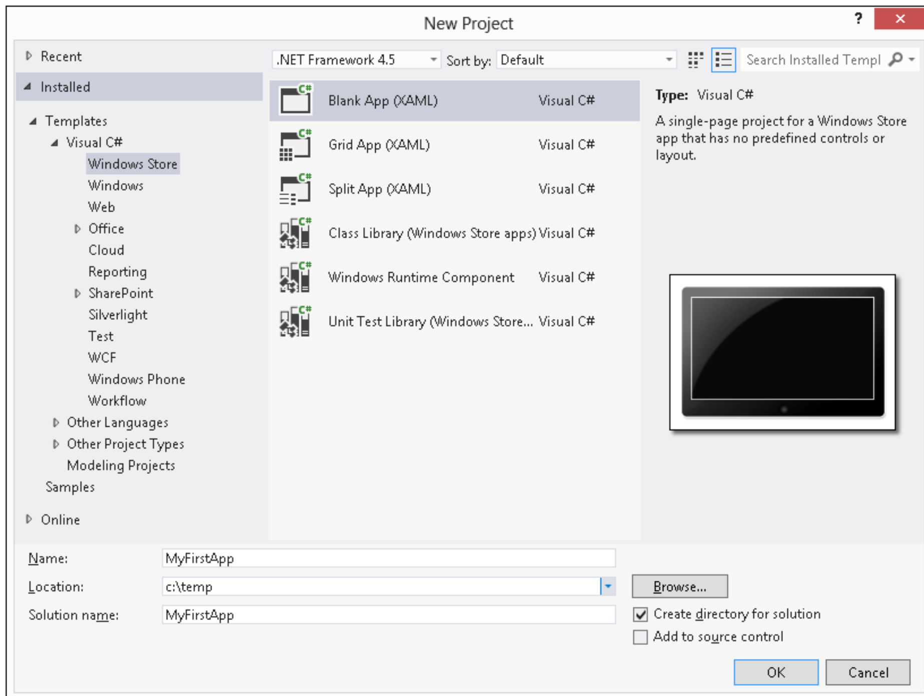
Create the project

As you may remember from Chapter 1, the SDK setup process installed some new templates and wizards to facilitate the creation of a Windows Store project. In the graphic that follows step 3, under the C# or VB project types, you can see a new section, named Windows Store, which represents the entry point for this new kind of project. This section exposes all the templates that are tailored to Windows 8.

1. Create a new Application project. To do that, open Visual Studio 2012, and from the File menu, select New Project (the sequence can be File | New | Project for full-featured versions of Visual Studio). Choose Visual C# in the Templates tree and Windows Store from the list of installed templates. Then choose Blank App (XAML) from the list of available projects.
2. Select version 4.5 as the target .NET Framework version for your new project (this step is not necessary in Visual Studio Express edition).
3. Name the new project **MyFirstApp**. Then choose a location on your file system as well as a solution name. When you're finished, click OK.

   If you use a source control system, you can select the Add To Source Control check box.

   The following graphic shows the first step of the New Project wizard: both the project and the solution will be assigned the name MyFirstApp.



At this stage, Visual Studio 2012 normally creates the solution folder, the project folder, and a project related to the chosen template.

Because you selected the Blank App project template, Visual Studio uses the simplest project structure to create your new application. Figure 3-1 shows the result of the procedure you just completed.

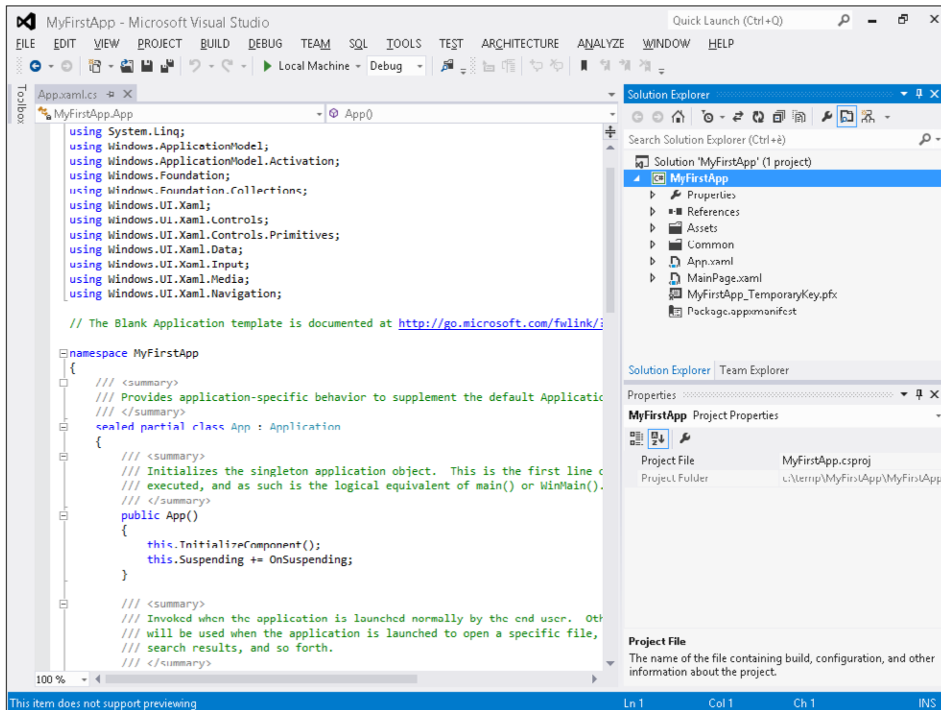


FIGURE 3-1 A blank Windows Store app in Solution Explorer.

In fact, you can easily find a file called App.xaml and one named MainPage.xaml, as well as a folder named Properties, that contains the classic AssemblyInfo.cs file. The file list is similar to the one you would get if you had created a Windows Presentation Foundation Browser application (or even a Windows Presentation Foundation application); however, there are some differences.

The first difference from a Windows Presentation Foundation (WPF) application is the absence of the app.config file. This means that, as in a Microsoft Silverlight or Windows Presentation Foundation Browser application, you cannot use the classic .NET configuration mechanism. In fact, the runtime system is somewhat sandboxed as in a Silverlight or WPF Browser application: specifically, the users cannot navigate to the file system where the application will be installed and change application files because Windows Store apps are usually downloaded and installed from the Windows Store. The exception to this rule is when you're working in the development environment, where Visual Studio 2012 (or you using a command-line tool) can install the application for testing purposes.

The second difference from Silverlight and WPF Browser applications is the presence of the Package.appxmanifest file. This file contains a description of the application (its icon and synergy with the operating system) and the operating system features that the application uses, called "application capabilities and declarations." From this perspective, the project is similar to one targeting Windows Phone 7.x, where the WMAppManifest.xml file informs the operating system of the capabilities the application requires to run.

Figure 3-2 shows the Package.appxmanifest designer that Visual Studio provides to simplify the application definition. As you can see, the Application UI tab lets you choose the Display Name of the application—that is, the name for the Start screen—the application description, three logos for the application, and so on.

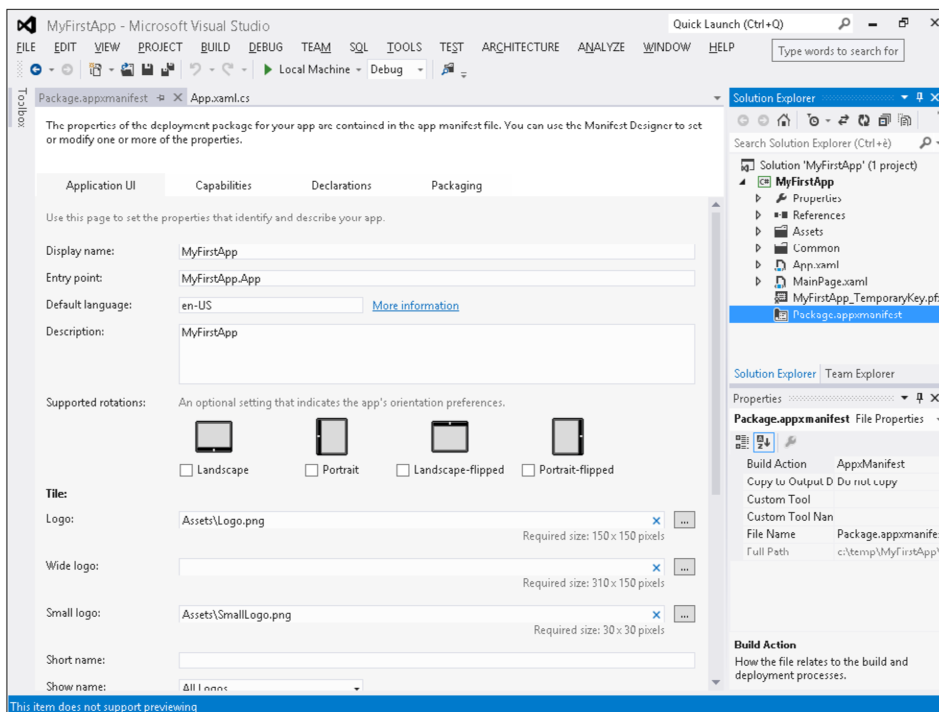


FIGURE 3-2 Visual Studio application manifest designer.

Another similarity with a Windows Phone project is the presence of some default images in the project. These images are available in the Assets folder and are referenced from the Package.appxmanifest file. The default template uses an application logo image for the default application tile (Logo.png), an image for the initial splash screen (SplashScreen.png), a small logo image displayed in the application's tile—used if the application changes the tile size from code (SmallLogo.png)—and last but not least, the image used by the Windows Store to represent the application (StoreLogo.png). As you can see from Figure 3-2, there is no default wide logo, nor is this image referenced by the Package.appxmanifest.

If you run the application now, leaving all the default files and manifest settings intact, you will experience a short delay while Visual Studio deploys the application to the developer system, and then you will see the splash screen, followed by a completely blank screen that represents the application. This may seem strange—because Visual Studio has traditionally added some sample text to all its templates—but as you will discover in the following procedure, many things happened during application deployment.

Explore the deployed app on the system

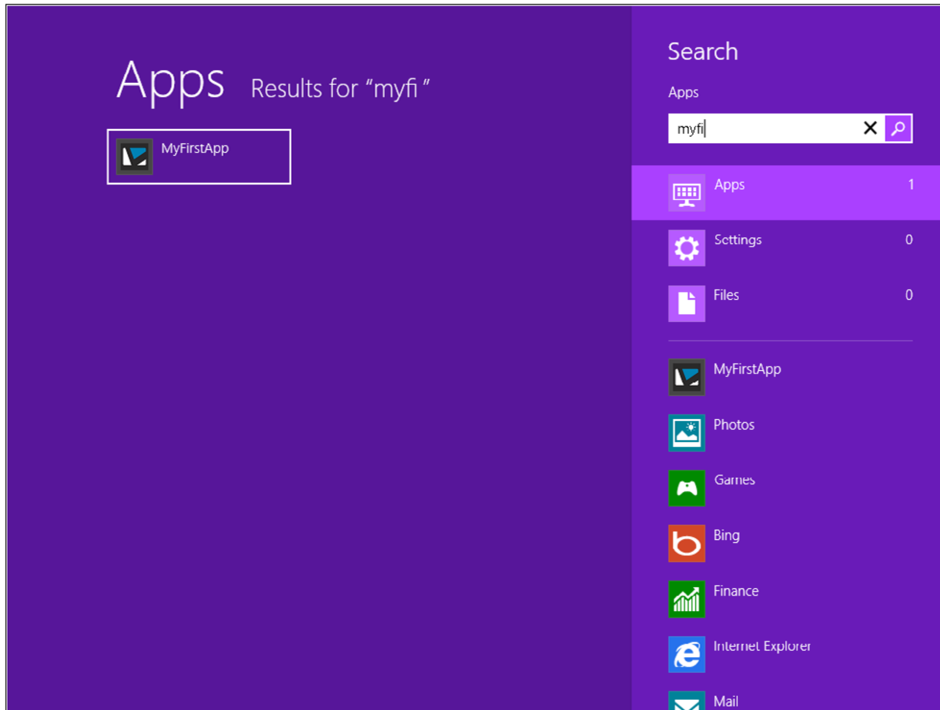
First, note the absence of the classic window frame with the "X," minimize, and maximize buttons. In fact, this is the first version of Windows without windows.

Follow these steps to explore what Visual Studio has asked Windows 8 to do during the deployment of the application.

1. Click the Start button of your tablet or keyboard, or go to the left-bottom corner of the screen using your mouse and click Start to return to the Start screen.
2. Scroll to the right using your finger, the mouse wheel, or the bottom scroll bar until you reach the rightmost end of the application Tiles on the Start screen.

   At the very end of the application tiles, you'll see your first deployed Windows 8 app, which has a tile with the name "MyFirstApp."
3. Click on the app's tile to reopen the application.
4. Return to Visual Studio and stop the debugging session by clicking Stop Debugging or pressing Shift+F5.
5. Repeat steps 1 and 2 and now tap and hold your finger (or right-click). The command bar will ask if you want to uninstall or simply unpin the application; "unpin" means deleting the application tile from the start menu, while leaving the application on the system.
6. Unpin the application by clicking Unpin.
7. Move your mouse to the bottom-right corner of the screen to view the Charms, and then choose Search, or press Windows+Q on the keyboard. The Search pane will appear at the right of the screen.

8. Type the first few letters of the name of the application and choose Apps from the list of places where Windows should search. Your application will appear in the left pane.



9. You can launch the application by either tapping or clicking the application's name—but don't do that now. Instead, tap and hold (or right-click) the application to open the command bar.

10. Pin the application using the Pin button. The application is now listed in the Start screen using the default tile. You can verify the tile's presence by repeating steps 1 and 2 of this procedure.

Note that you can search for files or settings within the same Search pane, as well as perform a search inside the listed applications. These applications have declared the Search capability in their Package.appxmanifest. Next, you'll add the Search capability declaration to the simple application you are developing in this chapter.

Before proceeding, if you launched the application from the Search pane or the Start screen—that is, if you launched the application from outside of Visual Studio—you need to close it before you can deploy it again. If you use Visual Studio to launch an application, the first operation that the IDE requests from the operating system is package deployment. When deployment is complete, Visual Studio starts the application and attaches the debugger to it. If you stop the debugging session from Visual Studio, the Windows process is terminated; the same termination occurs if the application crashes. If the application is launched outside of Visual Studio using the default template, you do not have any close button—as you saw in the previous examples. The application occupies the entire screen and you will need to manually stop (*kill* is a better word) the process from running indefinitely. You can do this through the Windows Task Manager, or by pressing Alt-F4, or using the application close gesture to close the

application in a more graceful way. (The application close gesture closes an application when you quickly swipe your finger from the top-center of the screen to the bottom-center.)

You will learn the details of the application lifecycle in Chapter 4, "Application lifecycle management," but for now it is important to understand that Windows 8 has a completely new way of managing the lifecycle of applications. An application is in the running state when the user uses it (the user has chosen the application as the foreground application); when the user leaves the application in any manner—by clicking Start, going back to the previous application, or starting a new search, and so on—the system may suspend the application or terminate it if the system needs more memory. This behavior is in some ways similar to the application lifecycle management in Windows Phone 7.x, as well as other modern operating systems.

As mentioned, Task Manager provides another way to stop a running application. Task Manager has been modified in Windows 8 so you can also see an application's status under the advanced options of the View menu. If you cannot see the View menu, click More Details in Task Manager. Figure 3-3 shows MyFirstApp in the suspended state within Task Manager. Save the Planet, a real application ported from Windows Phone 7 to Windows 8, is not in the suspended state—meaning that it is still running.

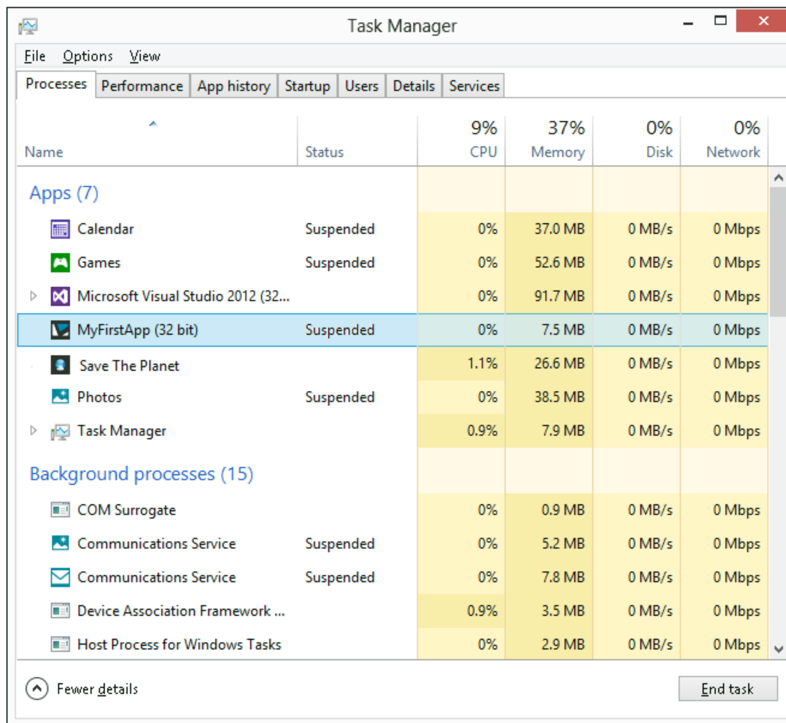


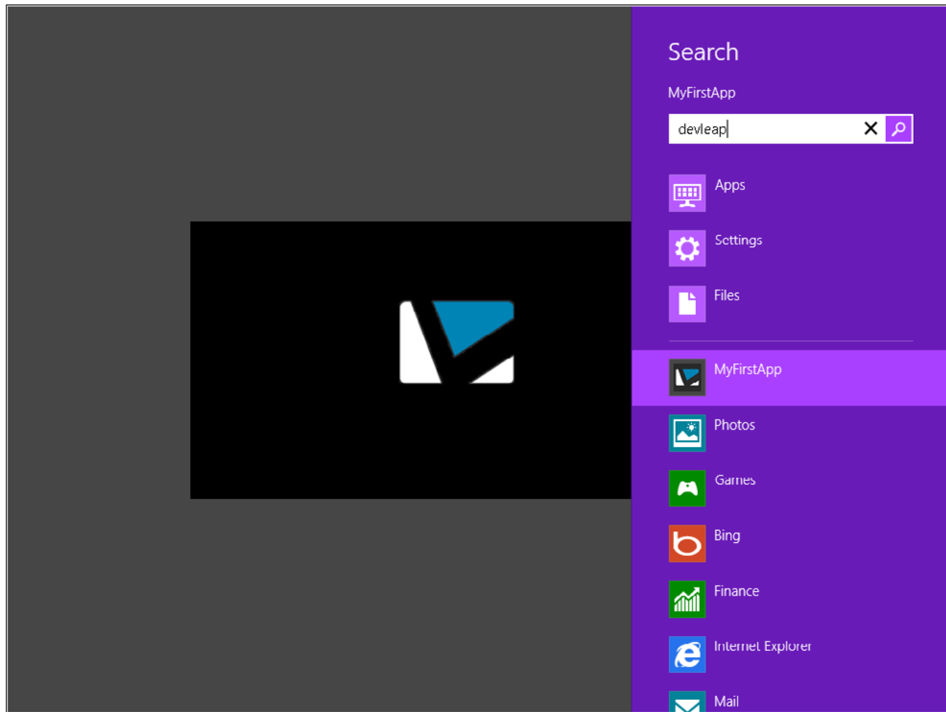
FIGURE 3-3 Task Manager showing the suspended/running state for a Windows Store app.

This mechanism applies only to Windows Store applications and not to classic .NET or Win32 applications. In fact, the two instances of Visual Studio, Paint (used to take the screenshots for this book) and many other Win32 applications are in the running state.

Adding the Search Declaration to the application manifest

In this procedure, you will add the Search Declaration to the application manifest to let the user search for text "inside" this sample application. Follow these simple steps inside the Visual Studio 2012 project you are building.

1. Double-click the Package.appxmanifest file inside the MyFirstApp application to open the designer.
2. Click the Declarations tab to manage the declarations for this application.
3. Choose Search from the Available Declaration listbox, and then click Add. As stated in the Description section, the Search declaration "...registers the application as providing search functionality. Users will be able to search the application from anywhere in the system." The phrase "search the application" means passing the search text entered by the user to the application so it can search inside the application.
4. Before testing the application, click the Application UI tab and make sure that All Logos is selected in the Show Name drop-down list.
5. To change the default logo, copy the .png files you can find in the Chapter 03 Demo Files in the Logos folder to the Assets folder of the project. The files have the default names so you do not need to modify the Package.appxmanifest.
6. Right-click the project item in the solution (MyFirstApp) and choose Deploy. This operation deploys the application to Windows 8 without launching a debugging session.
7. Open the Start screen by using the Start button and scroll to the right to verify that the name and the new logo appear on the application tile.
8. Press Windows+F or Windows+Q to activate one of the Windows Search interfaces (the first opens the search page to search for files; the second to search for applications), and type some text in the textbox. Scroll the resulting list of applications to verify that your application is shown in the list. You can click an application to open it (the sample application does nothing right now); you will add the code to implement the search in the last part of this chapter.



Adding UI elements

In this section, you will analyze the remaining project items that the template created and add some code to build a list of people and bind it to the user interface.

Note It is beyond the scope of this chapter to analyze the various binding techniques, as well as the user interface patterns such as MVVM (Model View ViewModel) or MVC (Model View Controller).

Let's start by analyzing the code proposed by the Visual Studio 2012 template. You have explored the meaning and functionality of the application manifest and the image folder. Listing 3-1 shows the XAML source code for the main page, which has been modified to contain a *ListView* standard user control that will display the *FullName* property of a list of bound elements.

LISTING 3-1 Modified MainPage.xaml page

```
<Page
    x:Class="MyFirstApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyFirstApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <ListView x:Name="list" DisplayMemberPath="FullName" />
    </Grid>
</Page>
```

The page includes the classic XAML definition for a page control represented by the *MyFirstApp.MainPage* class. The user control references four XML namespaces—just like a Silverlight project, a WPF app, or a Windows Phone 7.x application.

By default, the template uses a *Grid* for the layout, but you will change this in a later procedure, where you will add some styling to change the look and feel of this simple application.

You will also modify the code behind for the MainPage.xaml page, as shown in Listing 3-2, so that it calls a fake "business layer" that returns a list of people represented by the *Person* class you will also implement shortly.

LISTING 3-2 Modified MainPage.xaml.cs code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=234238

namespace MyFirstApp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();

            // Fill the ListView
            var biz = new Biz();
            list.ItemsSource = biz.GetPeople();

        }
```

```
        /// <summary>
        /// Invoked when this page is about to be displayed in a Frame.
        /// </summary>
        /// <param name="e">Event data that describes how this page was reached.  The Parameter
        /// property is typically used to configure the page.</param>
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
        }
    }
}
```

Modify and test the application

1. Modify the MainPage.xaml file so that its contents are identical to Listing 3-1.
2. Open the code-behind file (MainPage.xaml.cs) and insert the bold lines in Listing 3-2.
3. Add a new class file to the project to implement the *Biz* class by right-clicking the term *Biz* in the code behind. Then choose Generate | Class.
4. Generate a method stub for the *GetPeople* method by using the same technique: right-click the *GetPeople* method, choose Generate | Method Stub. Use the following code to replace the code of the Biz.cs file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstApp
{
    public class Biz
    {
        public List<Person> GetPeople()
        {
            return new List<Person>()
            {
                new Person() { FullName = "Roberto Brunetti" },
                new Person() { FullName = "Paolo Pialorsi" },
                new Person() { FullName = "Marco Russo" },
                new Person() { FullName = "Luca Regnicoli" },
                new Person() { FullName = "Vanni Boncinelli" },
                new Person() { FullName = "Guido Zambarda" },
                new Person() { FullName = "Jessica Faustinelli" },
                new Person() { FullName = "Katia Egiziano" }
            };
        }
    }

    public class Person
    {
        public string FullName { get; set; }
    }
}
```

5. Run the application.

The code in the *Biz* class simply returns a list of people represented by the *Person* class. For the sake of simplicity, this class has just one property, *FullName*.

When you run the app, the result will look similar to Figure 3-4. You should be able to select a person from the list.

FIGURE 3-4 Main page of the application presenting the listbox of names.

It is time to forget the developer inside you and put on your designer hat to transform the plain vanilla list into something more appealing. Stop the debugging session and return to Visual Studio 2012.

Before refining the appearance of the list, you need to add some more user interface elements to the page—such as a TextBlock control to display the application's title—and make your first app appear more integrated with the Windows 8 environment.

To add a title, you need to modify the XAML source in the MainPage.xaml file, as shown in Listing 3-3:

LISTING 3-3 MainPage.xaml with a *GridView* control

```
<Page
    x:Class="MyFirstApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyFirstApp"
    xmlns:d="http://schemas.m icrosoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- page title -->
        <Grid Grid.Row="0" Grid.Column="0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="120"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <TextBlock x:Name="pageTitle" Grid.Column="1" Text="My First Windows 8 App"
                Style="{StaticResource PageHeaderTextStyle}"/>
        </Grid>

        <ListView x:Name="list" DisplayMemberPath="FullName" Grid.Row="1" Grid.Column="0"
                Margin="116,0,0,46"/>
    </Grid>
</Page>
```

Now, if you press F5 in Visual Studio, your page should look similar to the one shown in Figure 3-5.

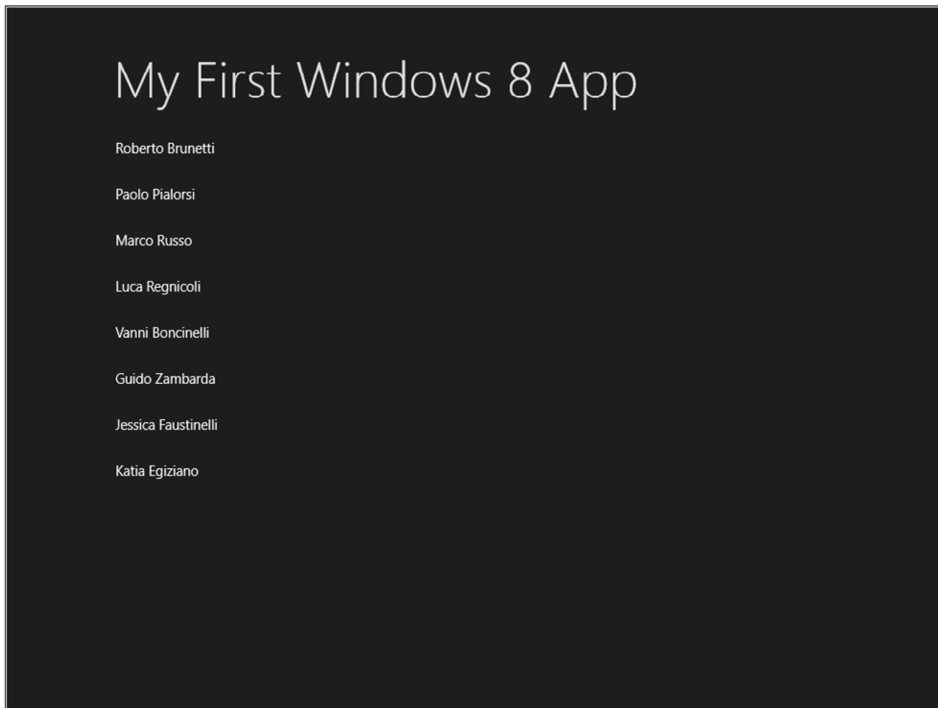


FIGURE 3-5 The main page with the title.

Listing 3-3 used a Grid element as the root element of the page. In XAML, the *Grid* panel allows you to place child elements in rows and columns, as well as define in advance the number and the properties of each row and column by leveraging the *RowDefinitions* and *ColumnDefinitions* properties of the Grid control.

In the example, the main grid was split into two rows. But now it is time to return to the code for a deeper explanation. The first four lines of the *Grid* control definition are as follows.

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="140"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
```

To define rows and columns of the main Grid control, we used the *Grid.RowDefinitions* property. This syntax (in the form *classtype.propertyname*, also known as extended property syntax) represents a standard way to set complex properties using the XAML markup language. Within the *RowDefinitions* property you'll find two instances of *RowDefinition*: the first sets the height equal to 140 pixels, whereas the second uses the "*" (star) character to define an unknown-at-design-time value that can fill the remaining space on the screen. Keep in mind that it is very important to design a user interface that can adapt to the user's screen resolution; tablets and devices are available with widely varying screen resolutions and orientations. Using relative rather than absolute sizing helps a great deal in achieving the goal of an adaptive interface.

Assigning each graphic element to a cell of the grid suffices to set the *Grid.Row* and *Grid.Column* properties of the element itself. These properties are also called *attached properties* because they don't belong to the object model of the target element, but are instead "attached" to the control itself. This scenario includes two child elements in the main grid.

- First, a secondary *Grid* control that will contain the title page elements. This *Grid* control has two attached properties: *Grid.Row*, with a value of *0*, and *Grid.Column*, also with a value of *0*. This will place it in the first row and first column of the main grid.
- Next, there is a *ListView* control, with the properties *Grid.Row = "1"* and *Grid.Column = "0,"* that place it in the second row of the first column.

Here are some other useful tidbits of information about how to use the *Grid* control.

- You can omit the *Grid.Row* and/or *Grid.Column* properties if their value is 0.
- If a *Grid* control does not explicitly set the *RowDefinitions* property, it is treated as having a single *RowDefinition* definition whose *Height* property is set to "*".
- If a *Grid* control does not explicitly set the *ColumnDefinitions* property, it is treated as having a single *ColumnDefinition* definition whose *Width* property is set to "*".

- You can set the RowDefinition's *Height* property to "Auto," in which case its size is defined at runtime by the height of the controls it contains.
- You can set the ColumnDefinition's *Width* property to "Auto," in which case its size is defined at runtime by the width of the controls it contains.

Continuing the analysis of the XAML code, you'll find a secondary *Grid* control, further divided into two columns, whose only child is a *TextBlock* control.

```
<TextBlock x:Name="pageTitle" Grid.Column="1" Text="My First Windows 8 App"
    Style="{StaticResource PageHeaderTextStyle}"/>
```

The property setting *Grid.Column = "1"* means that the *TextBlock* control will be positioned in the second column of the parent *Grid* control, whereas the *Style* property references a style called PageHeaderTextStyle using the special *{StaticResource}* syntax (you will explore the basic concepts underlying such styles in later chapters). For now, just remember that a style is simply a container for property settings—a shared object that can be reused in different scenarios.

The property *Grid.Row = "1"* has been added to the *ListView* control so that it will occupy the entire second row of the main grid, and the property *Margin = "116,0,0,46"* places the *ListView* control a few pixels away from the edges of the cell. The *Margin* property is set using four numbers separated by commas. The first number identifies the distance from the left edge and then continuing clockwise; in our example, the *ListView* control is placed 116 pixels away from the left edge, 0 from the top and right edges, and 46 pixels from the bottom edge.

Now try to add some photos to the project. To do that, simply drag the folder called Photos (included in the Demo Files for this chapter) into Visual Studio, and drop it when your cursor is on the project root called MyFirstApp. As a result of this operation, Visual Studio will create a directory called Photos in the project's root (at the same level as the Assets and Common folders) containing some .jpg files.

The next step is to modify the *Person* class to add a custom property called *Photo*, and define the business component to set that property.

Listing 3-4 shows the code for the modified Biz.cs file. Copy Listing 3-4 into the Biz.cs file.

LISTING 3-4 Modified Biz.cs code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstApp
{
    public class Biz
    {
        public List<Person> GetPeople()
        {
            return new List<Person>()
        {
```

```
            new Person() { FullName = "Roberto Brunetti", Photo = "Photos/01.jpg" },
            new Person() { FullName = "Paolo Pialorsi", Photo = "Photos/02.jpg" },
            new Person() { FullName = "Marco Russo", Photo = "Photos/03.jpg" },
            new Person() { FullName = "Luca Regnicoli", Photo = "Photos/04.jpg" },
            new Person() { FullName = "Vanni Boncinelli", Photo = "Photos/05.jpg" },
            new Person() { FullName = "Guido Zambarda", Photo = "Photos/06.jpg" },
            new Person() { FullName = "Jessica Faustinelli", Photo = "Photos/07.jpg" },
            new Person() { FullName = "Katia Egiziano", Photo = "Photos/08.jpg" }
        };

        }
    }

    public class Person
    {
        public string FullName { get; set; }
        public string Photo { get; set; }
    }
}
```

To make the view of the people contained in the *ListView* control more appealing, you must modify the control's *ItemTemplate* property. It is important to understand that in XAML, a template object is equivalent to the concept of "structure," and the *ItemTemplate* property represents the structure of the individual items in the *ListView* control.

You start by editing the XAML source code of the MainPage.xaml page to make some tweaks to the *ListView* control.

Replace the ListView definition in the MainPage.xaml:

```
<ListView x:Name="list" DisplayMemberPath="FullName" Grid.Row="1"
    Grid.Column="0" Margin="116,0,0,46"/>
```

with this markup code:

```
<ListView Grid.Row="1" Grid.Column="0" x:Name="list" Margin="116,0,0,46">
    <ListView.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding FullName}" FontSize="10" />
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

The second example removes the *DisplayMemberPath* property, which displayed only simple strings connected to the *FullName* property of the bound objects, and replaces it with the *ItemTemplate* property that accepts objects of type *DataTemplate*. In this scenario, the *DataTemplate* consists of a simple label (a TextBlock) with its *Text* property connected to the *FullName* property of the bound object; if you now run the application, you will see the list of people displayed in a smaller font. This is not a huge graphical improvement over the previous version, but these steps function as the basis for subsequent activities you will perform.

In the next step, you will try to change the *DataTemplate* of each item to display both the name and the photo. Replace the DataTemplate definition of the *ListView*.

```
<DataTemplate>
       <TextBlock Text="{Binding FullName}" FontSize="10" />
</DataTemplate>
```

with this code:

```
<DataTemplate>
       <StackPanel Width="200" Height="200">
                <TextBlock Text="{Binding FullName}" />
                <Image Source="{Binding Photo}" />
       </StackPanel>
</DataTemplate>
```

Compared to the previous step, this uses a new panel called *StackPanel*, which places child items arranged vertically, one under the other, or—if the *Orientation* property is set to *Horizontal*—side by side. In this scenario, each item in the *ListView* will be displayed using a *StackPanel* that will render the person's name and photo by binding, respectively, the *FullName* property with the *Text* property of a *TextBlock* and the *Photo* property with the *Source* property of an *Image* control.

Until now we have used the *ListView* control, which can display a series of vertical elements; now, let's try to replace the previous *ListView* definition:

```
<ListView Grid.Row="1" Grid.Column="0" x:Name="list" Margin="116,0,0,46">
       <ListView.ItemTemplate>
                <DataTemplate>
                    <StackPanel Width="200" Height="200">
                        <TextBlock Text="{Binding FullName}" />
                        <Image Source="{Binding Photo}" />
                    </StackPanel>
                </DataTemplate>
       </ListView.ItemTemplate>
</ListView>
```

with this new markup code that uses a *GridView* control:

```
<GridView Grid.Row="1" Grid.Column="0" x:Name="list" Margin="116,0,0,46">
       <GridView.ItemTemplate>
                <DataTemplate>
                    <StackPanel Width="200" Height="200">
                        <TextBlock Text="{Binding FullName}" />
                        <Image Source="{Binding Photo}" />
                    </StackPanel>
                </DataTemplate>
       </GridView.ItemTemplate>
</GridView>
```

The *GridView* control, as the name suggests, is able to display its items in a tabular form, or grid.

If you press F5 in Visual Studio, you will see the result shown in Figure 3-6.

FIGURE 3-6 Element selected in the customized *GridView* control.

This outcome is acceptable, but you can do even better using just a bit of creativity and a few lines of XAML code within the *DataTemplate*. The next listing shows the entire MainPage.xaml page with the code changed in the previous step highlighted in bold.

Replace the entire code of the MainPage.xaml with the following.

```
<Page
    x:Class="MyFirstApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyFirstApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
```

```
        <!-- Back button and page title -->
        <Grid Grid.Row="0" Grid.Column="0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="120"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <TextBlock x:Name="pageTitle"  Grid.Column="1"
               Text="My First Windows 8 App" Style="{StaticResource PageHeaderTextStyle}"/>
        </Grid>

        <GridView Grid.Row="1" Grid.Column="0" x:Name="list" Margin="116,0,0,46">
            <GridView.ItemTemplate>
                <DataTemplate>
                    <Grid>
                        <Image Source="{Binding Photo}" Width="200" Height="130"
                            Stretch="UniformToFill" />
                        <Border Background="#A5000000" Height="45" VerticalAlignment="Bottom">
                            <StackPanel Margin="10,-2,-2,-2">
                                <TextBlock Text="{Binding FullName}" Margin="0,20,0,0"
                                    Foreground="#7CFFFFFF" HorizontalAlignment="Left"  />
                            </StackPanel>
                        </Border>
                    </Grid>
                </DataTemplate>
            </GridView.ItemTemplate>
        </GridView>
    </Grid>
</Page>
```

The new *DataTemplate* uses a *Grid* as the root element, with two elements nested within it: an *Image* and a *Border*. Because the *Grid* has neither *RowDefinitions* nor *ColumnDefinitions*, it will render as a single cell containing the two child elements, following the order defined in the markup—that is, the first child element rendered by the runtime will be the *Image* control, then the *Border* control (with all its children) will be rendered in overlay. Beyond those changes, the XAML markup adds only one new thing: the *Background* property of the *Border* control that contains the following string "#A5000000." It is worth noting the first two characters after the #: they represent the alpha channel, or transparency, of the color defined by the subsequent six characters (black, in this case). In fact, in this example, the *Border* does not have a full and "opaque" color as background, but rather uses a semi-transparent black for graphical purposes.

The result is quite in line with the Windows 8 ecosystem and visually pleasing, as you can see in Figure 3-7.

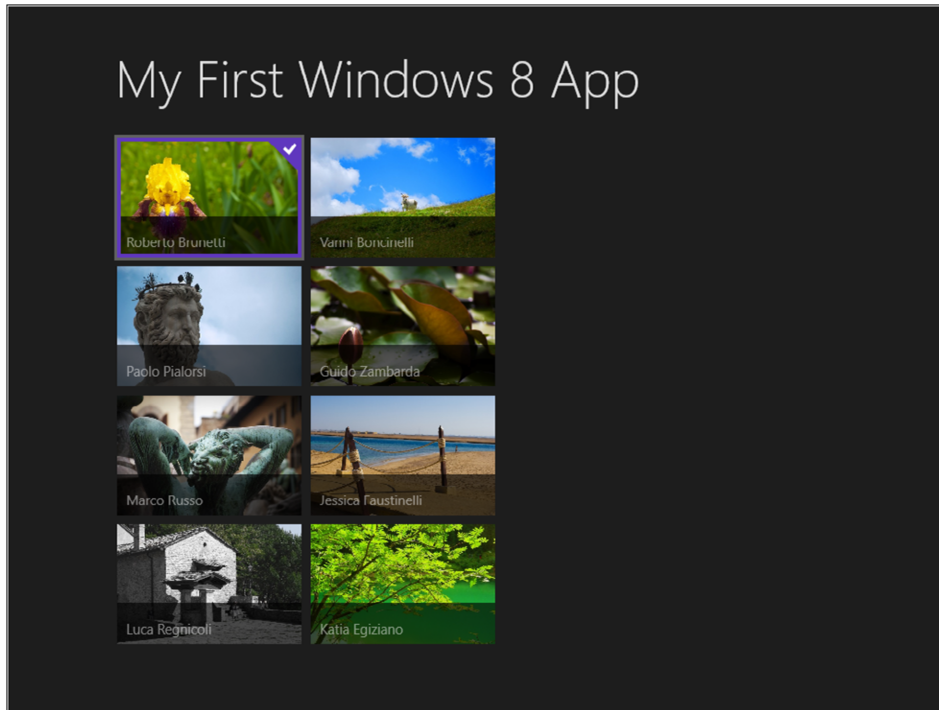


FIGURE 3-7 A different customization of the *GridView* control.

It is worth noting that the controls provided by the framework support all types of input, such as mouse, keyboard, touch screen, and pen for free—in other words, you don't have to write code to make the controls respond to normal input.

Adding search functionality

In this section, you will add the code that enables the searching capability inside the application.

One thing you may notice in a Windows Store application project is the absence of direct references; if you open the *References* element in the project tree you will not find the classic *System.Something* assembly. Instead, there is just a .NET for Windows Store apps reference and a Windows reference. These contain all the Windows Runtime classes you need to develop Windows Store apps.

You can add the complete implementation of the search feature inside the application without adding any references; you need only to add a reference if you create your own class library, for which you would need to add a reference to the corresponding assembly. You can find more information about developing custom class libraries in Chapter 5, "Introduction to the Windows Runtime."

In a previous procedure, you added the Search Declaration to the application, letting the operating system include the application in the Search pane. The declaration in the manifest tells the Windows 8 runtime: "I'm a searchable application." In other words, the system will present the

application as a possible target for a search inside the application itself. A search target is the scope for the user's search, which may be a file in the file system, an installed application, a setting in the control panel, or some text inside a searchable application.

When the user selects the application as the target for his or her search, the application is activated for the search and the search string typed by the user is passed to the application. The idea is simple: the application is the only component that can correctly show the search result; no other component, nor the operating system itself, knows about the data inside the application. The way the application presents the data is tailored to the specific application data. In Chapter 6, "Windows Runtime APIs," you will learn more about search integration as well as about other WinRT APIs, such as Share, Webcam, FilePicker, and so on.

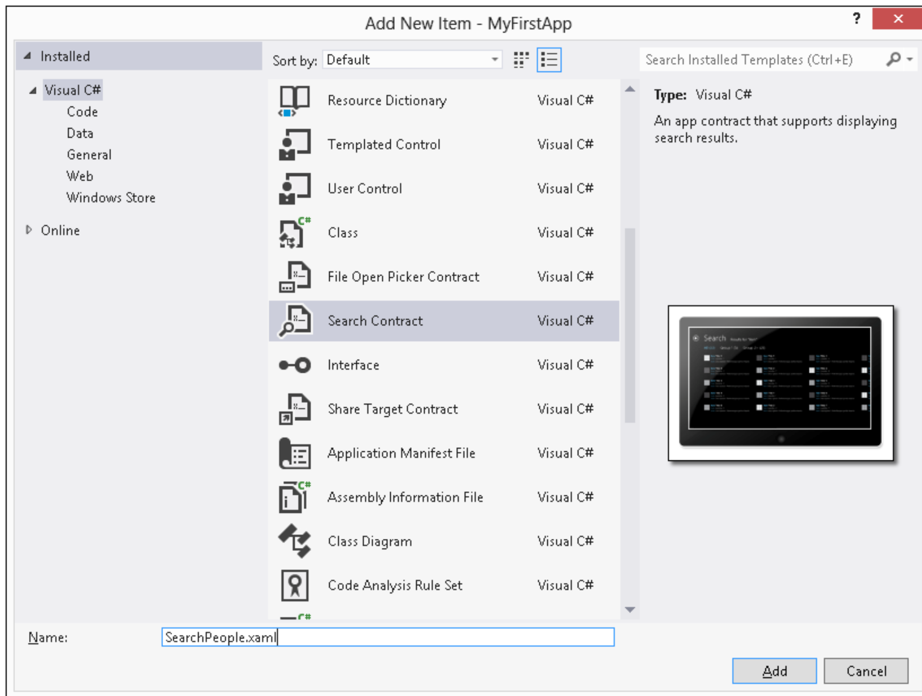
The search feature is implemented by a contract, called a *search contract,* that regulates the search interaction between an application and the operating system. The search contract states the following:

- The application needs a registration. This registration is based on the manifest declaration.
- The declaration can include the executable name, that is, the application .exe file name—the entry point for the application that the system will call when the user chooses the application as the search target.
- The application will present the data in the appropriate format using a page.
- The application will receive the search text entered by the user in the entry point. It is the responsibility of the application to present the page with some feedback to the user; the feedback can be the list of items found or a message (in case of search failure). The failure can be a "Not Found" text or graphics, or "Data not available, try again later." Be as specific as you can with the message.
- Windows manages the Search History for the user.
- The application can provide suggestions for the text entered by the user.

Add the search contract

There is a Visual Studio template that provides a simple implementation of a contract that covers all the search points in the preceding list—except for the last one. The first step you will perform in this procedure is to remove the Search Declaration you added in a preceding procedure to explore the default implementation. Then follow the remaining steps to add the search functionality.

1. Remove the Search Declaration from the manifest opening the Package.appxmanifest. Go to the Declarations tab, look for "Search" in the Supported Declaration list, select it, and click Remove. Save the manifest.
2. Add a new Search Contract item by right-clicking the project in the Solution Explorer and choosing Add | New Item.
3. In the Add New Item dialog, select Search Contract and name it **SearchPeople.xaml**.



Click OK.

4. In the dialog that asks you to add all the files you need to implement the contract, click Yes.

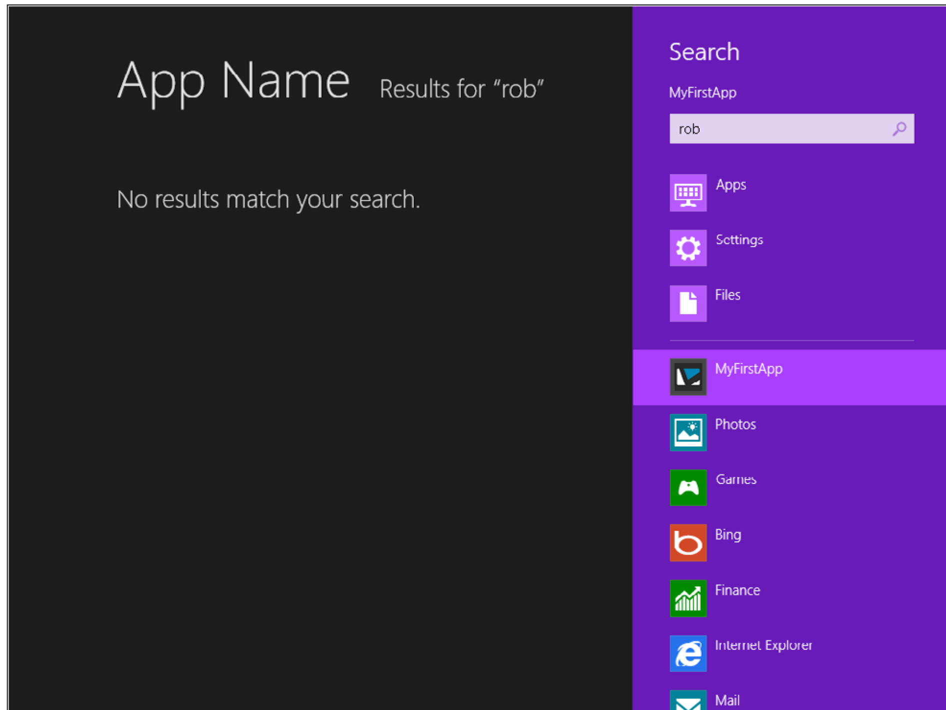
Test the default search component

Before doing anything else, you can test the application immediately to fully understand the complete flow. You will implement the people search in the procedure after this one.

1. Deploy the application from Visual Studio by right-clicking the project element in the Solution Explorer and choosing Deploy.

2. Press Windows+Q to activate the Search pane.

3. Type the text you want in the search box and choose MyFirstApp from the application list. The operating system will launch the application (which was not running yet because you just deployed it), and activate the search inside the application using a call to the search contract entry point. The application shows the SearchPeople.xaml page that, obviously, presents no results yet.



4. Close the application using Alt+F4 or Task Manager.
5. Start the application from the Start screen.
6. Press Windows+Q again to start a new search.
7. Type some text in the search box and choose MyFirstApp in the application list. The result page is identical to the previous one, but the Back button is now enabled because the search target (your application) was already running when you activated the search.
8. Click the Back button and note that the application is in the same state.
9. Go to the Start screen and open another application (Mail works fine). Repeat steps 6 through 8. The result will be always a blank page. However, if you click the Back button, you can see the page that shows the previous search; this demonstrates that the application was put into the suspended state and resumed when the search target was activated.
10. Press Alt+Tab (yes, that key combination still works in Windows 8) to select another application for the foreground.
11. Go to the Start screen and launch your application. The application presents the search result because Windows 8 suspends the application and restores it if the user comes back.

Now that you have explored the search flow, it's time to implement the Search Contract template. The template adds the Search Declaration to the Package.appxmanifest, as you can verify by double-clicking the file and selecting the Declarations tab.

This template also modifies the project—among other things, it adds a new page to display the search results (SearchPeople.xaml or whatever name you used in the Add New Item dialog) that you saw in the previous procedure when you chose MyFirstApp as the search target.

This new page is shown when a search is activated. The contract defines the entry point for the "search call" that, by default, is the *App* class.

The Search Contract Visual Studio Template also modified the App.xaml.cs file to override the *OnSearchActivated* method of the base class so that it shows the search result page. Listing 3-5 shows the complete code for the App.xaml.cs file.

LISTING 3-5 Code-behind file for the *App* class: App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Application template is documented at http://go.microsoft.com/fwlink/?LinkId=234227

namespace MyFirstApp
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.
        /// This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Invoked when the application is launched normally by the end user.
        /// Other entry points will be used when the application is launched to open
        /// a specific file, to display, search results, and so forth.
        /// </summary>
        /// <param name="args">Details about the launch request and process.</param>
```

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
    Frame rootFrame = Window.Current.Content as Frame;

    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // Create a Frame to act as the navigation context and navigate
        // to the first page
        rootFrame = new Frame();

        if (args.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: Load state from previously suspended application
        }

        // Place the frame in the current Window
        Window.Current.Content = rootFrame;
    }

    if (rootFrame.Content == null)
    {
        // When the navigation stack isn't restored navigate to the first page,
        // configuring the new page by passing required information as a navigation
        // parameter
        if (!rootFrame.Navigate(typeof(MainPage), args.Arguments))
        {
            throw new Exception("Failed to create initial page");
        }
    }
    // Ensure the current window is active
    Window.Current.Activate();
}

/// <summary>
/// Invoked when application execution is being suspended.  Application state is saved
/// without knowing whether the application will be terminated or
/// resumed with the contents
/// of memory still intact.
/// </summary>
/// <param name="sender">The source of the suspend request.</param>
/// <param name="e">Details about the suspend request.</param>
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    //TODO: Save application state and stop any background activity
    deferral.Complete();
}

/// <summary>
/// Invoked when the application is activated to display search results.
/// </summary>
/// <param name="args">Details about the activation request.</param>
protected async override void OnSearchActivated(Windows.ApplicationModel.Activation.
          SearchActivatedEventArgs args)
{
```

```
            // TODO: Register the Windows.ApplicationModel.Search.SearchPane.
                 GetForCurrentView().QuerySubmitted
            // event in OnWindowCreated to speed up searches once the application is already
                 running

            // If the Window isn't already using Frame navigation, insert our own Frame
            var previousContent = Window.Current.Content;
            var frame = previousContent as Frame;

            // If the app does not contain a top-level frame, it is possible that this
            // is the initial launch of the app. Typically this method and OnLaunched
            // in App.xaml.cs can call a common method.
            if (frame == null)
            {
                // Create a Frame to act as the navigation context and associate it with
                // a SuspensionManager key
                frame = new Frame();
                MyFirstApp.Common.SuspensionManager.RegisterFrame(frame, "AppFrame");

                if (args.PreviousExecutionState == ApplicationExecutionState.Terminated)
                {
                    // Restore the saved session state only when appropriate
                    try
                    {
                        await MyFirstApp.Common.SuspensionManager.RestoreAsync();
                    }
                    catch (MyFirstApp.Common.SuspensionManagerException)
                    {
                        //Something went wrong restoring state.
                        //Assume there is no state and continue
                    }
                }
            }

            frame.Navigate(typeof(SearchPeople), args.QueryText);
            Window.Current.Content = frame;

            // Ensure the current window is active
            Window.Current.Activate();
        }
    }
}
```

The *OnLaunched* method is the standard code suggested by the Windows Store Application template and is needed to activate the main page when the user launches the application. An application is "launched" when its state is not running.

The *OnSearchActivated* method is the code for the Search Contract default implementation. The code instantiates the designated page and calls the *Activate* custom method to pass the received arguments.

The *SearchActivatedEventArgs* used by the *OnSearchActivated* method and the *LaunchActivated EventArgs* used by the *OnLaunched* methods both implement the *IActivatedEventArgs* interface.

The first property of the interface is *Kind,* and it can be one of the values defined in the *Activation-Kind* enumeration. This property lets the developer ask for the kind of activation during launching; for instance, if the application is launched by the user, this property will be *ActivationKind.Launch*. However, if the application is launched by the system when the user designates it as search target, the property will be *ActivationKind.Search*. If the application is activated to receive something from other applications using a Share Contract, the property will be *ActivationKind.ShareTarget*.

The *QueryText* property of the *SearchActivatedEventArgs* contains the text entered by the user in the Search pane. This property is used in the default *OnSearchActivated* method during the navigation to the search page, as you can see in the following excerpt.

```
frame.Navigate(typeof(SearchPeople), args.QueryText);
Window.Current.Content = frame;

// Ensure the current window is active
Window.Current.Activate();
```

As you can see, the search terms are received in the *navigationParameter* parameter of the *LoadState* method of the SearchPeople.xaml.cs page and used to build the *QueryText* property of the user interface in the *DefaultViewModel* property of the page. Listing 3-6 shows the code for this method.

LISTING 3-6 Extract of SearchPeople.xaml.cs code behind

```
protected override void LoadState(Object navigationParameter, Dictionary<String, Object>
pageState)
{
    var queryText = navigationParameter as String;

    // TODO: Application-specific searching logic.  The search process is responsible for
    //       creating a list of user-selectable result categories:
    //
    //       filterList.Add(new Filter("<filter name>", <result count>));
    //
    //       Only the first filter, typically "All", should pass true as a third argument in
    //       order to start in an active state.  Results for the active filter are provided
    //       in Filter_SelectionChanged below.

    var filterList = new List<Filter>();
    filterList.Add(new Filter("All", 0, true));

    // Communicate results through the view model
    this.DefaultViewModel["QueryText"] = '\u201c' + queryText + '\u201d';
    this.DefaultViewModel["Filters"] = filterList;
    this.DefaultViewModel["ShowFilters"] = filterList.Count > 1;
}
```

The code in Listing 3-6 is relatively simple. The first line defines a local variable called queryText to host the text entered by the user in the search box. This text is passed in the search contract as the *QueryText* property of the *SearchActivatedEventArgs*.

The placeholder lets you choose the business logic to look for the text in your data and represents the most important part of this code.

The last three lines of code are useful if you decide to use the default layout to display the search results. The code assigns the text for the query, the filters list and a Boolean to indicate whether to show the filters list in the bindable dictionary (*IObservableMap* in fact derives from *IDictionary*). Let's try to implement the search by reusing the business layer you saw at the beginning of this chapter.

Implement the search logic

In the following procedure, you will implement the logic for retrieving the list of people. Although you can implement the logic using a LINQ (Language Integrated Query) query on the results from the business logic component *List* method, consider passing the search parameter to the business logic component to perform the search in lower layers. Generally speaking, it is a bad idea to filter the entire set of data in memory in the user interface layer. For the sake of simplicity, this sample application has no persistence layer. Thus, you will implement the search in memory inside the business layer.

1. Add a method to the business logic component (Biz.cs) to filter the data source using the following code:

```
public List<Person> GetPeople(String search)
{
        var list = this.GetPeople();
        return list.Where(p => p.FullName.Contains(search)).ToList();
}
```

2. Add a call to the new *GetPeople* method from the SearchPeople.xaml.cs *LoadState* method and assign the result to the *DefaultViewModel* property. Use the following code as a reference (the lines to add are in bold).

```
protected override void LoadState(Object navigationParameter,
    Dictionary<String, Object> pageState)
{
    var queryText = navigationParameter as String;

    // TODO: Application-specific searching logic.  The search process is
    //       responsible for
    //       creating a list of user-selectable result categories:
    //
    //       filterList.Add(new Filter("<filter name>", <result count>));
    //
    //       Only the first filter, typically "All", should pass true as a third
    //       argument
    //       in order to start in an active state.  Results for the active filter
    //       are provided in Filter_SelectionChanged below.

    var biz = new Biz();
    var people = biz.GetPeople(queryText);
    this.DefaultViewModel["Results"] = people;
```

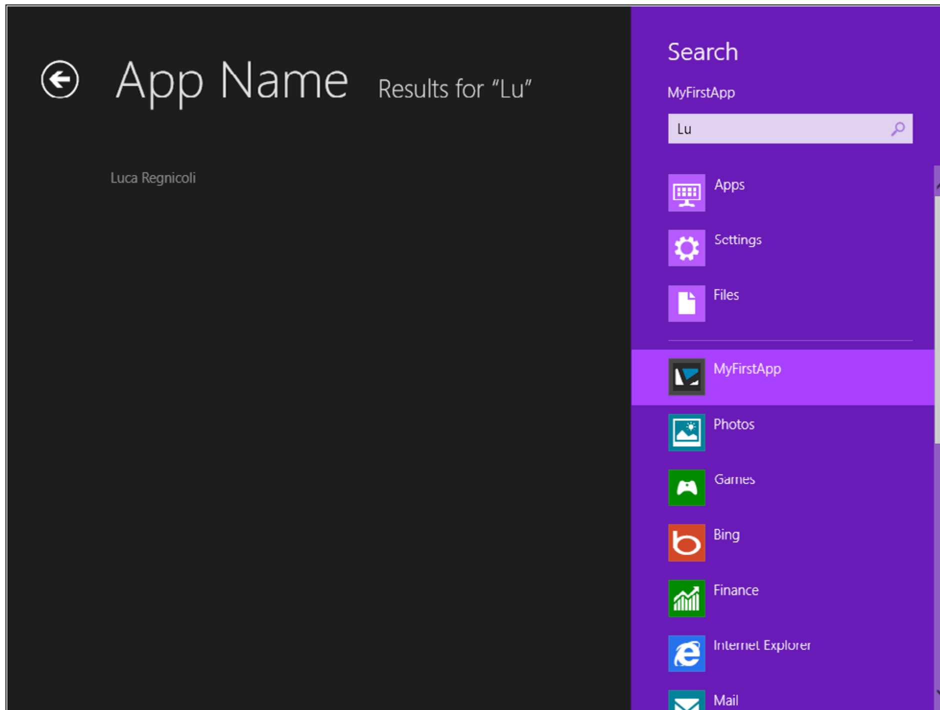
```
    var filterList = new List<Filter>();
    filterList.Add(new Filter("All", 0, true));

    // Communicate results through the view model
    this.DefaultViewModel["QueryText"] = '\u201c' + queryText + '\u201d';
    this.DefaultViewModel["Filters"] = filterList;
    this.DefaultViewModel["ShowFilters"] = filterList.Count > 1;
}
```

3. Open SearchPeople.xaml and find the *GridView* control named resultGridView. Remove the *ItemTemplate* default definition and define a new one to show the person name for each result. The following code shows the complete control's definition:

```
<GridView
  x:Name="resultsGridView"
  AutomationProperties.AutomationId="ResultsGridView"
  AutomationProperties.Name="Search Results"
  TabIndex="1"
  Grid.Row="1"
  Margin="0,-238,0,0"
  Padding="110,240,110,46"
  SelectionMode="None"
  IsSwipeEnabled="false"
  IsItemClickEnabled="True"
  ItemsSource="{Binding Source={StaticResource resultsViewSource}}">
           <GridView.ItemTemplate>
              <DataTemplate>
                 <TextBlock Text="{Binding FullName}" Margin="0,20,0,0"
                    Foreground="#7CFFFFFF" HorizontalAlignment="Left"  />
              </DataTemplate>
           </GridView.ItemTemplate>
           <GridView.ItemContainerStyle>
              <Style TargetType="Control">
                 <Setter Property="Height" Value="70"/>
                 <Setter Property="Margin" Value="0,0,38,8"/>
              </Style>
           </GridView.ItemContainerStyle>
</GridView>
```

4. Deploy the application and test a search from the Search pane, as you learned in the "Test the Default Search Component" procedure.



The last thing you need to do to complete the sample application is to change the *DefaultViewModel* property value to display the actual number of people retrieved by the search.

Modify the View Model properties

In this procedure, you will modify the code to show the actual number of people retrieved by the search. The procedure is very straightforward.

1. Modify the *LoadState* method as follows. The lines in bold represent the updated ones.

```
protected override void LoadState(Object navigationParameter,
    Dictionary<String, Object> pageState)
{
var queryText = navigationParameter as String;

// TODO: Application-specific searching logic.  The search process is responsible for
//       creating a list of user-selectable result categories:
//
//       filterList.Add(new Filter("<filter name>", <result count>));
//
//       Only the first filter, typically "All", should pass true as a third argument
//       in order to start in an active state.  Results for the active filter are
//       provided in Filter_SelectionChanged below.

var biz = new Biz();
var people = biz.GetPeople(queryText);
this.DefaultViewModel["Results"] = people;
```

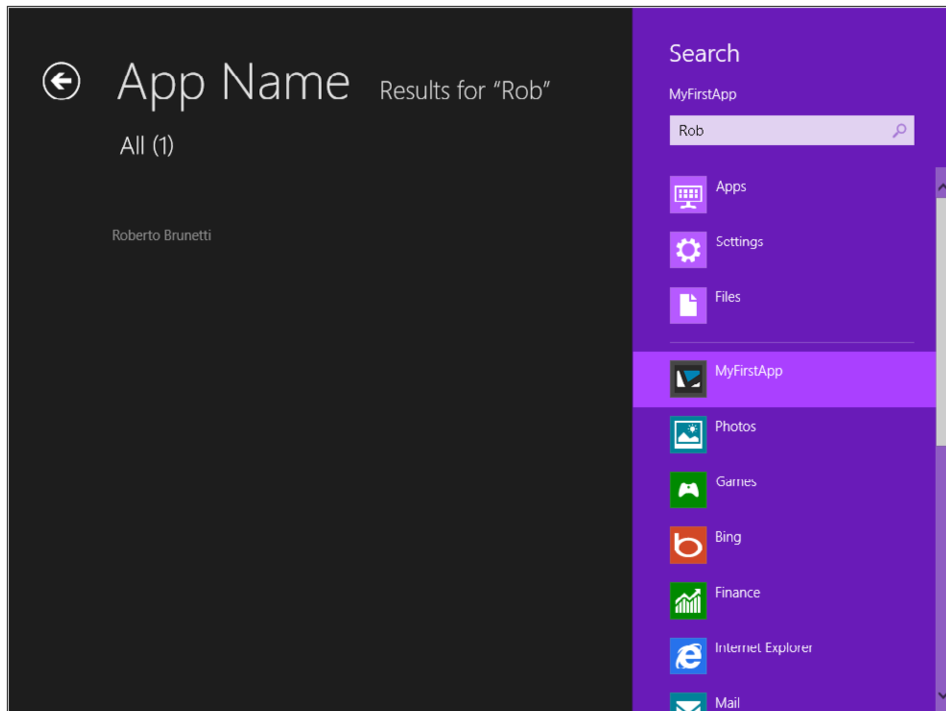
```
var filterList = new List<Filter>();
filterList.Add(new Filter("All", people.Count, true));

// Communicate results through the view model
this.DefaultViewModel["QueryText"] = '\u201c' + queryText + '\u201d';
this.DefaultViewModel["Filters"] = filterList;
this.DefaultViewModel["ShowFilters"] = filterList.Count >= 1;
}
```

In practice, the first filter that shows the "All" keyword will contain the actual number of retrieved results and the *ShowFilters* boolean property indicates whether to show the various filters to the user. Obviously, you have to implement the various filters and the corresponding code.

2. Kill the application using the Task Manager because the process is probably already running from the previous procedure.

3. Deploy the application and test it again using the Search pane.



Summary

In this chapter, you saw the complete cycle for creating, testing, and deploying a simple Windows 8 application. You learned about the available templates and how to describe the application using the manifest. Finally, you added the code to implement the search contract using a provided template.

The next chapter is dedicated to application life cycle management. You will learn the details of the application manifest: how to package, test, and deploy an application, and how Windows 8 manages the launch, suspension, and termination of an application.

Quick reference

To	Do This
Arrange controls inside a flexible grid area	Use the *Grid* control.
Arrange child elements into a single line that can be oriented horizontally or vertically	Use the *StackPanel* control.
Deploy a Windows Store application	Use the deployment feature of Visual Studio 2012.
Deploy and test the application	In Visual Studio, press F5.
Implement the Search Contract	Use the SDK template called Search Contract that adds the search result page, the manifest declaration, and some sample code to the solution.
Define application features	Use the Visual Studio IDE Designer and open the Package.appxmanifest file.
Close an application	Stop the debugger, in case you are debugging it; or press Alt+F4; use the closing gesture; or use the new Task Manager to terminate the process.

Index

Symbols

* (star) character, 202, 206

A

access token (access_token) parameter, 327
Account picture provider extension, 25
ActivatableClassId (registry key), 152
Activated activation (WinRT), 120
activation, 118–120
  OnSearchActivated method, 118–120
ActivationKind enumeration (IActivatedEventArgs interface), 115
Add-AppxDevPakage.bat, 106
Akzidenz Grotesk font, 33
alerts, Toasts as, 17
AllowCropping property (CameraCaptureUI class), 171
Alt+Tab functionality, 49
Always on top command (Windows 8 Simulator), 27
Animation Library, 58
animations, 58
APIs, hardware specific, in Windows 8, 142
app.config file, 99
app development/design
  architecture, 295–328
  control locations, considerations for, 51
  fill view, 60–63
  graphics, resolution and scale considerations, 60
  landscape vs. portrait layouts, 60
  scalability, 58–60
  snapped view, 60–63
  thumb-reach map for touchscreens, 51–52
  touch-first designing, 54
  WinRT and, 14
Appearance property group, 209
  Common property group, 225
  StrokeThickness property, 224
Application Bar control, 8–13, 259–262
  limit on number of, 259
  top/bottom, purpose of, 8
ApplicationData class, 317
application lifecycle management (ALM), 121
application manifest
  Add-AppxDevPakage.bat, 106
  <App Name_Version_Compilation>.appxsym, 106
  <App Name_Version_Compilation>.appxupload, 106
  <App Name_Version_Compilation>.cer, 106
  Capabilities section, 101
  package.appxmanifest, 287
  Properties section, 100
  Search Declaration, adding, 74–75
  tile definitions in, 286–288
  VisualElements tag, 102
Application UI tab (Visual Studio designer), 74
<App Name_Version_Compilation>.appxsym, 106
<App Name_Version_Compilation>.appxupload, 106
<App Name_Version_Compilation>.cer, 106
apps
  architecture of, 295–328, 298–299
  Border class, 207–210
  Canvas control, 189–192
  characteristics of, 41–63
    animations, 58
    comfort/touch, 51–55
    edges, 49–51
    form factors, 58–60
    full screen layout, 47–49
    Semantic Zoom, 56–58
    silhouette, 56–58
    snapped/fill view, 60–63
  consuming data from, 310–316
  controls, customizing appearance of, 214–228
  databases, connecting to, 300
  drawing in Visual Studio, 185–188

Grid control, 198–207
installing behavior, 106
layout, creating, 189–214
lifecycle of, 99–132
activation, 118–120
launching, 111–118
resume, 126–132
suspension, 121–126
manifest, 100–103
Margin property, 210–214
package, 103–107
contents of, 106
running from Visual Studio, 103
ScrollViewer control, 194–197
search contract, adding, 87–88
search functionality, adding, 86–97
searching for, 11
search logic, implementing, 94–96
Silverlight vs. Windows 8 apps, 70–71
StackPanel control, 192–194
storage/cache, implementing, 316–320
testing changes to, on the fly, 77–78
TextBlocks, adding, 78
tool bar location considerations, 50
UI elements, adding, 75–86
unistall behavior, 106
unpinning from Start screen, 71
Windows Live IDs, 298
Windows Store, 107
WinRT registration of, 150–153
WPF vs. Windows 8 apps, 70–71
App.xaml.cs file, 111
OnLaunched method, 112
search contracts and, 90
architecture, 295–328
data, consuming, 310
data layer, implementing, 299–301
N-tier solutions, 295
OData communication layer, implementing, 306–310
SOAP communication layer, implementing, 302–306
storage/cache, implementing, 316–320
Associate App With The Store menu (Visual Studio 2012), 110
AsTask method (.NET), 247, 254
asynchronous methods, 237–242
implementing, 239–242
asynchronous operations, 231–258
async keyword, 231–237
await keyword, 231–237
canceling, 246–248
CancellationToken class and, 247
CancellationTokenSource class and, 247
events, waiting for, 243–244
exception handling, 244–246
exceptions, behavior of, 245
implementing, 237–242
progress, tracking, 249–253
SynchronizationContext library and, 257
synchronizing multiple, 253–256
async keyword, 231–237
usage, 234–237
Windows.Storage.FileIO.ReadTextAsync method, 236
attached properties, 191
AuthenticateAsync method (WebAuthenticationBroker), 326
authentication
against other webservices, 324–328
OData services and, 324–328
SOAP service, consuming with, 321–323
SOAP service, validating, 323–324
AutoPlay extension, 25
availability (concept), 297
await keyword, 231–237
getting around, 243
usage, 234–237

B

BackgroundTaskBuilder class, 21
background tasks, 20–22
creating, 21
Lock screen apps and, 22
resource management and, 22
Background tasks extension, 25
Badges, 15–20
basicHttpBinding, 320–324
Basic touch mode command (Windows 8 Simulator), 27
battery consumption and background tasks, 22
Bauhaus style, 38–41
applied to software, 33
Blank App (XAML) template, 66
BorderBrush (Properties window), 208
Border class, 207–210
BorderBrush property, 208
BorderThickness property, 209
CornerRadius property, 209

BottomAppBar node (Document Outline), 260
Brush property, 221
business layer (BIZ), 296
Bustamante, Michele Leroux, 303
Button controls, 190

C

C#, 14
C++
  background tasks and, 22
  custom WinMD libraries, consuming, 145
  WinRT and, 14, 135
  WinRT Camera API and, 137–138
Cached File Updater contract, 24
caching, implementing, 316–320
Camera API (WinRT), 136–138
CameraCaptureUI class, 165
  Windows.Media.winmd and, 139
CameraCaptureUIMode parameter (CaptureFileAsync method), 165
Camera settings extension, 25
CancellationToken class (.NET), 247
CancellationTokenSource class (.NET), 247
Canvas control
  Left property, 191
  Top property, 191
  usage, 189–192
Capabilities tab (Package.appxmanifest file), 101, 166
CaptureFileAsync method (CameraCaptureUI type), 140, 165
Chakra engine, 141
Change resolution command (Windows 8 Simulator), 28
characteristics of apps, 41–63
  animations, 58
  comfort/touch, 51–55
  edges, 49–51
  form factors, 58–60
  full screen design, 47–49
  Semantic Zoom, 56–58
  silhouette, 41–47
  snapped/fill view, 60–63
Charms, 8–13
  adding features to, 12
  displaying, 11
  edges and, 49
  sharing, 23
Class Library (Windows Store apps) template, 67
client-server software, 295
ClosedByUser application state, 116
CLR apps
  background tasks and, 22
  WinRT and, 136–137
  XML nodes, accessing, 142
ColumnDefinitions property (Grid control), 200–201
  Width property, 205
COM (Component Object Model) Interop, 133
comfort/touch characteristics, 51–55
Common Language infrastructure (CLI), 135
common language runtime apps
  background tasks and, 22
  WinRT and, 136–137
  XML nodes, accesssing, 142
Common property group (Appearance property), 225
Common XAML Controls section (Visual Studio Toolbox), 261
communication layer
  OData, implementing with, 306–310
  SOAP, implementing with, 302–306
ConfigureAwait method (Task object), 257
Contact picker extension, 25
context menu (Solution Explorer), 264
Contracts (WinRT), 23–25
  Data Transfer Manager, 172
  extenstions, 25
  native applications and, 173–174
  result page, implementing, 181–183
  Share charm, 172
  source application, implementing, 175–179
  source app responsibilities, 172
  target application, implementing, 179–181
  target app responsibilities, 172
ControlChannelReset (SystemTrigger events)
  background tasks and, 21
ControlChannelTrigger (for background tasks)
  background tasks and, 21
  Lock Screen and, 22
  resource consumption and, 22
controls, 259–278
  appearance of, customizing, 214–228
  Application Bar, 259–262
  deleting unwanted, in Design View, 187
  FlipView, 271–274
  GridView, 268–271
  ListView, 264–268
  moving/shaping in Design View, 187
  predefined styles, customizing, 216–218

predefined templates, usage, 226–227
SemanticZoom, 274–278
sytles and, 215–216
templates and, 223–228
templates, creating, 223–226
templates, customizing predefined for, 227–228
WebView, 263–264
Copy screenshot command (Windows 8 Simulator), 28
CornerRadius property (Border element), 209
Create App Package feature (Visual Studio 2012), 104
Windows Live ID, 105
Create Data Binding for [FlipView].ItemsSource modal window, 272
Create Data Binding for [Image].Source modal window
FlipView control and, 273
GridView control and, 270
Create Data Binding For [ListView].ItemsSource modal window, 266
Create Data Binding for [TextBlock].Text modal window
FlipView control and, 273
GridView control and, 270
ListView control and, 267
Create DataTemplate Resource modal window
FlipView control and, 272
GridView control and, 269
ListView control and, 266
Create Style Resource modal window, 221
CredentialCache object, 325
Credentials property, 325
CustomAttributes (registry key), 153

D

data access layer (DAL), 296
database persistence layer, 295
data, consuming from Windows 8 apps, 310–316
DataContext property, 265
data layer, implementing, 299–301
DataTransferManager (WinRT class)
sharing contracts and, 172
usage, 176
DatePicker Calendar Control, 155
DatePicker control (WPF), 155
Deadline property (SuspendingOperation), 125
Declarations tab (Visual Studio designer), 74
Declaration tab (Package.appxmanifest file), 179
deployment, 297
Design View window (Visual Studio), 186
controls, moving/shaping, 187
Grid controls and, 199
Properties window, 187–188
Toolbox tab, 186
direct references, 86
DisplayMemberPath property (ListView control), 82
Display Name (of applications), 101
Document Library property, 163
Document Outline tab (Design View window)
customizing styles and, 219
keeping active, 260

E

ECMA-335 (CLI metadata definition language), 135
edges
Alt+Tab functionality and, 49
characteristics, 49–51
Entity Framework 5
usage, 299–301
EntitySetRights enumeration, 307
enumerable collections (WinRT), 135
error handling
asynchronous methods and, 244–246
essential iconography, 36–37
event handlers
for app events, 112
exception handling
in asynchronous code, 244–246
ExePath (registry key), 153
Extensible Application Markup Language (XAML) framework, 155
extensions, 25

F

Facebook, 326–328
File Activated activation (WinRT), 120
File activation extension, 25
FileOpenPicker class, 156–163
File Picker Activated activation (WinRT), 120
File Picker contract, 24
files, searching for, 11
fill view, 60–63
flexible layouts
designing, 278–285
MainPage.xaml.cs for, 280–281

MainPage.xaml for, 279–280
testing in Device tab, 281–285
FlipView control, 271–274
fluidity, concept of, 58
FontSize property (TextBlock control), 188
Foreground property (Brush property), 221
Foreground property (Properties window), 228
form factors, 58–60
full screen design, 47
functionalism, 32

G

Game Explorer extension, 25
gestures
pinch, 56–58
reversibility of, 55
stretch, 56–58
swipe, 49–51, 55
GetIids method (WinRT objects), 141
GetRuntimeClassName method (WinRT objects), 141
GetTrustLevel method (WinRT objects), 141
Globally Unique Identifier (GUID), 106
graphic assests
scalability and, 60
Grid App (XAML) template (Visual Studio), 43–47
details page, 46
homepage of, 44–45
as Windows Store project template, 66
Grid.Column property, 80, 201
Grid controls, 198–207
ColumnDefinitions property, 200–201
default values for, 80
Design View window and, 199
Margin property, 81
usage, 78–86
Grid.RowDefinitions property, 80
Grid.Row property, 80, 205
grid system, 39
Grid Systems in Graphic Design (Müller-Brockmann), 34
GridView control, 268–271
customizing, 83–86

H

Height property (RowDefinition), 205
Height property (StackPanel control), 193
Help command (Windows 8 Simulator), 28
Helvetica font, 33
HKEY_CLASSES_ROOT\Extensions\ContractId\Windows.Launch key, 152
HomeAppBarButtonStyle, 260, 261
HSTRING (WinRT), 135
HTML5
Chakra engine and, 141
consuming custom WinMD libraries with, 148–150
WinRT and, 134
HTTP authentication, 320

I

IActivatedEventArgs interface, 115
IAsyncActionWithProgress<TProgress> object, 249
IAsyncInfo interface (.NET), 247
IBackgroundTask interface, 21
IInspectable interface, 140
Image controls, 163
INT32 (WinRT), 135
Integrated Development Environment IDE designer, 235
interface definition, 303
Intermediate Language Disassembler (ILDASM) tool, 139
international language, 35
testing for, 40
international language conventions, 40
International Typographic Style, 33
InternetAvailable (SystemTrigger events)
background tasks and, 21
condition of, checking for, 22
Internet Explorer
as source application, 173
XML, viewing in, 309
InternetNotAvailable condition
background tasks, checking for, 22
IProgress<T> interface (Task objects), 249
IRandomAccessStream interface, 165
ItemTemplate property (ListView control), 82
IUnknown interface and IInspectable interface, 140

J

JavaScript, WinRT and, 14
JSON (JavaScript Object Notation), 302

K

keyboard
testing, 53
touch screens, splitting for thumbing, 53
keyboard shortcuts (Charms features), 11
Kind property (IActivatedEventArgs interface), 115

L

LaunchActivatedEventArgs
IActivatedEventArgs interface, 115
OnLaunched method and, 115
launch type, 115–116
Layout property (Properties window), 196
Margin property and, 211
Padding property, 228
layouts, flexible
creating, 189–214
designing, 278–285
MainPage.xaml.cs for, 280–281
MainPage.xaml for, 279–280
testing in Device tab, 281–285
Learning WCF (Bustamante), 303
lifecycle (of apps), 99–132
activation, 118–120
launching, 111–118
resume, 126–132
suspension, 121–126
ListBox control
FileOpenPicker picker and, 157
ListView controls, 264–268
DataContext property, 265
DisplayMemberPath property, 82
ItemTemplate property, 82
Orientation property and, 83
Live Tiles, 15–20
defining, 288–292
LoadState method, 96
LocalFolder property (ApplicationData class), 317
LocalSettings property of applciatios, 126
Lock screen, 15–20
accessing settings for, 19
background tasks and, 22
CPU management and, 22
limits on number of apps in, 19
LockScreenApplicationAdded (SystemTrigger events), background tasks and, 21
LockScreenApplicationRemoved (SystemTrigger events), background tasks and, 21

M

Mail (as target application), 174
MainPage.xaml.cs file, 280
MainPage.xaml file
Design View window and, 186
for flexible layouts, 279
Image control, adding, 163
maintainability, 297
MaintenanceTrigger (for background tasks), 21
Manifest Designer (Visual Studio 2012), 101
Display Name, 101
MANIFEST (WinMD file), 140
Margin property (Grid), 81, 210–214
method stubs, generating, 77
Microsoft Management Console tool, 322
Microsoft OpenType, 39
Microsoft Silverlight vs. Windows 8 apps, 70–71
Microsoft Visual Studio 2010, 28
Microsoft Visual Studio 2012
Add To Source Control check box, 67
Basic touch mode (Windows 8 Simulator), 54
Create App Package feature, 104
debugger behavior on suspension, 121
"drawing" apps in, 185–188
filenames, changing, 311
graphical apps, creating, 185–186
installing, 65–66
Intermediate Language Disassembler (ILDASM) tool, 139
Manifest Designer, 101
Package.appxmanifest designer, 70
project templates, 66–75
rebuilding solutions, 305
running applications from, 103–107
Search Contract Template, 90
silhouette and templates in, 43
Start screen and, 151
Store menu, 104
suspension/resume, debugging with, 130
UI, creating in, 186–188
Windows 8 SDK, 25–28
WinMD libraries, creating in, 144–145
Microsoft Windows Azure Access Control Service (ACS), 328
Mouse mode command (Windows 8 Simulator), 27
mouse support, testing, 53–64
Müller-Brockmann, Josef, 34

N

native applications, sharing, 173–184
.NET
  WinRT and, 14
.NET applications
  WinRT and, 14
  XML nodes, APIs for accessing, 142
.NET Framework version 4.5, 65
netTcpBinding binding, 320
network bandwidth, background tasks and, 22
NetworkInformation type, 319
  cache, 319
NetworkStateChange (SystemTrigger events), 21
Northwind sample database, installing, 299
NotRunning execution state, 116
N-tier solutions, 295

O

OAuth (Open Authentication), 325
object relational mapping (ORM) technology, 299
OData Client Tools for Windows Store Apps, 313
OData service
  communication layer, implementing with, 306–310
  consuming, 313–316
  EntitySetRights enumeration, 307
  security infrastructure, 324–328
OnLaunched method (WinRT), 111–118
  LaunchActivatedEventArgs and, 115
  OnSearchActivated method vs., 118
OnlineIdConnectedStateChange (SystemTrigger events), 21
OnSearchActivated method, 118–120
  Search Contract and, 92
OnSuspending method, 124
Orientation property (Properties window)
  ListView controls and, 83
  setting, 194
  StackPanel control and, 192
ORM (object relational mapping) technology, 299
OtherActivity method (SynchronizationContext), 257

P

Package.appxmanifest file, 70, 100, 287
  designer in Visual Studio 2012, 70
  Search contract, adding to, 119
  Search declaration and, 74
  Windows Registry and, 151
  WMAppManifest.xml vs., 101
PackageId (registry key), 152
Package name property, 152
Padding property (Layout property), 228
Pialorsi, Paolo, 301
pickers, 155–163
  DatePicker Calendar Control, 155
  DatePicker control (WPF), 155
  FileOpenPicker class, 156
pinch gesture, 56–58
pinch/zoom touch mode command (Windows 8 Simulator), 27
P/Invoke (Platform Invoke), 133
Play To contract, 24
portability, 1
POX (Plain Old XML) messages,, 302
presentation layer, 296
press and hold gesture, 55
Print task settings extension, 25
"Programming Microsoft LINQ in Microsoft .NET Framework 4" (Pialorsi/Russo, 301
projection layer of WinRT, 135
projects (Visual Studio), creating, 67–71
Properties window (Design View), 187–188
  Appearance property group, 209
  Foreground property, 228
Protocol activation extension, 25
PushNotificationTrigger event, 22
  background tasks and, 21
  Lock screen and, 22

R

rasterized assets, 60
Rationalism, 32
ReadToEnd method, 231
RefreshAppBarButtonStyle, 260, 261
registry, WinRT types and, 140
Report method (Progress<T> class), 251
resource management
  battery power, 22
  suspension and, 121
result page, implementing, 181–183
Resume lifecycle event, 126–132. *See also* Suspension lifecycle event
  implementation of, 123
  refreshing data on, 128–130
  Resuming event, 126

RoamingFolder property (ApplicationData class), 317
RoamingSettings property, 126
RoamingStorageQuota property
(ApplicationData class)*(, 318
Rotate clockwise command
(Windows 8 Simulator), 27
Rotate counterclockwise command
(Windows 8 Simulator), 27
Rotation touch mode command
(Windows 8 Simulator), 27
RowDefinitions property (Grid), 205–206
Height property, 205
* character and, 80
RSS (Really Simple Syndication), 302
RSS (Rich Site Summary), 302
Runtime Broker (WinRT), 135
runtime (Windows), 14–15
Russo, Marco, 301

S

SaveAppBarButtonStyle, 260, 261
scalable software, 296
screen resolution, app design and, 58–60
Screenshot settings command
(Windows 8 Simulator), 28
ScrollViewer control, 194–197
Layout property, 196
search
functionality, adding to apps, 86–97
implementing logic for, 94–96
LoadState method and, 96
OnSearchActivated method, 92
OS behavior of, 87
Search Activated activation (WinRT), 120
Search Contract, 24
adding to apps, 87–88
OnSearchActivated method, 118–120
Search Contract template (Visual Studio 2012), 90,
118
Search Declaration, 74–75
search functionality
Charms and, 11
testing, 88–94
security
basicHttpBinding, 320
corporate infrastructure and, 1
n-layer solutions, 297
OData service, infrastructure for, 324–328
SOAP services, infrastructure for, 320–324
Segoe UI font, 39
SelectionChanged event
Live Tiles and, 289
Toasts and, 292
SemanticZoom control, 56–58, 274–278
ServicingComplete (SystemTrigger events)
background tasks and, 21
SessionConnected (SystemTrigger events)
background tasks and, 21
background tasks, checking for, 22
SessionDisconnected (SystemTrigger events)
background tasks and, 21
background tasks, checking for, 22
Set location command (Windows 8 Simulator), 28
Settings contract, 24
Settings Panel
keyboard shortcut for, 11
opening, 169
Share charm, 172
activating, 173
Share contract, 24
Share pane, 172
accessing, 173
Sharing Target Activated activation (WinRT), 120
silhouette
characteristics, 41–47
defined, 41
Visual Studio 2012 templates and, 43
SmsReceived (SystemTrigger events), 21
Snap view
applications, putting into the, 60–63
designing for, 5–7
SOAP services
authentication, consuming with, 321–323
authentication, validating, 323–324
communication layer, implementing with,
302–306
security infrastructure, 320–324
SSL certificates and, 322
Solution Explorer
Windows Authentication property, 324
source application, implementing, 175–179
source control, 67
Split App (XAML) template (Visual Studio), 43, 66
SSL certificates
apps, testing/installing, 322
extension, 25
Staatliches Bauhaus, 31

StackPanel control (Document Outline), 192–194, 261
  Orientation property and, 192, 194
StandardStyles.xaml file (Common), 216
StandardStyle.xaml file (App bar), 260
* (star) character, 202, 206
Start screen, 2–3
  tiles, defining appearance of, 286–288
  Tiles, moving and grouping, 7
  unpinning apps, 71
state, finding previous, 116–118
storage, implementing, 316–320
Store menu (Visual Studio 2012), 104
  Associate App With The Store menu, 110
StreamReader class, 233
stretch gesture, 56–58
StrokeThickness property (Apperance property), 224
styles
  custom controls and, 215–216
  customizing copy of predefined, 218–220
  new, creating, 221–222
  predefined, customizing, 216–218
SuspendedTime key (LocalSettings property), 126
  resume and, 130
SuspendingDeferral.Complete method, 125
Suspending event, 123–124
SuspendingOperation.GetDeferral method, 125
Suspension lifecycle event, 121–126. *See also* Resume lifecycle event
  implementation of, 123
  requesting more time for, 125–126
swipe gesture, 55
  edges and, 49–51
Swiss Design, 33–37
SynchronizationContext library
  asynchronous operations and, 257
  OtherActivity method, 257
System.Configuration namespace, 99
SystemEventTrigger (for background tasks), 21
System.Runtime.Serialization assembly, 302
System.ServiceModel assembly, 302
SystemTrigger events, listed, 21

T

tablets
  designing for, 51–55
  thumb-use map (of touchscreens), 51–52
target application, implementing, 179–181
Task.ConfigureAwait method, 257
Task Manager, killing processes with, 72–73
Task objects
  asynch methods, as return value for, 236
  asynchronous operations and, 232
  async statements and, 233
  IProgress<T> interface, 249
  Windows.Storage.FileIO.ReadTextAsync method, 236
Task.WaitAny method, 254
Task.WhenAll function, 254
Task.WhenAny function, 254
templates
  controls and, 223–228
  controls, creating for, 223–226
  controls, customizing predefined for, 227–228
  predefined, for controls, 226–227
TemporaryFolder option (ApplicationData class), 318
Three-tier solutions, 296
Tiles
  function and design considerations, 38
  live features, turning off, 3
  moving and grouping, 7
  resizing, 3
  usage, 285–294
  VisualElements tag and, 102
TimeTrigger event
  background tasks and, 21
  Lock screen and, 22
TimeZoneChange (SystemTrigger events)
  background tasks and, 21
Title property (DataTransferManager class), 176
Toasts, 15–20
  creating/scheduling, 292–294
  sending from the cloud, 294
Toolbox tab (Design View), 186
touch keyboard
  splitting for thumbing, 53
  testing, 53–55
touch screen support, testing, 53–64
transparency, 209
TransportWithMessageCredentials configuration, 320

U

UINT64 (WinRT), 135
UI (User Interface), 31–64
  App Bar, 8–13
  apps, adding elements to, 75–86

- Bauhaus style and, 38–41
- Border class, 207–210
- Canvas control and, 189–192
- Charms, 8–13
- color choices, 40
- controls, 259–278
- controls, customizing appearance of, 214–228
- creating in Visual Studio, 186–188
- desktop vs. touch-screen, 4
- flexible layouts, designing, 278–285
- functionality vs. container, 38
- Grid control, 198–207
- grid system, 39
- iconography, 40
- influences on, 31–40
- Live Tiles, creating, 285–294
- Margin property, 210–214
- photos vs. drawings, 40
- projects vs. products, 39
- ScrollViewer control, 194–197
- snapping, 5–6
- StackPanel control, 192–194
- Start screen, 2–3
- toasts, creating/scheduling, 292–294
- typography, 39

Unit Test Library (Windows Store apps) template, 67
Univers font, 33
UserAway (SystemTrigger events), 21
user experience, 1–8
- App Bar, 8–13
- Charms, 8–13
- uniformity of, with Windows 8, 1

UserNotPresent condition, 22
UserPresent (SystemTrigger events)
- background tasks and, 21–22

V

validation, 108–109
vector art, 60
versioning, WinRT and, 141
ViewModel, modifying properties of, 96–97
Visual Basic, WinRT and, 14
VisualElements tag (manifest), 102
Visual State property (Device tab), 282
Visual Studio 2012. *See* Microsoft Visual Studio 2012
Visual Studio 2012 Express edition, 25

W

WaitAll method (WaitHandle class), 254
WaitAny method (Task object), 254
WaitHandle.WaitAll method, 254
WaitHandle.WaitAny method, 254
WaitOne method (Task), 244
WCF Service code template, 305
Weather App, 39
WebAuthenticationBroker class, 325
Web Browser Application (WPF), 99
webcam API, 163–171
Web Services Interoperability Organization (WS-I) Basic Profile specification, 320
WebView control, 263–264
WhenAll function (Task object), 254
Width property (ColumnDefinition), 205
- setting to Auto, 211

Width property (Properties window), 273
Window.Current.SizeChanged event, 281
Windows 7, developing for Windows 8 on, 26
Windows 8
- downloading, 65–66
- exploring apps in, 71–73
- fluidity, concept of, 58
- Search interfaces, opening, 74
- Task Manager, 72–73
- touch gestures, listed, 54

Windows 8 samples (MSDN), 292
Windows 8 SDK, 25–28
- Windows 8 Simulator, 27

Windows 8 Simulator, 27
- Basic touch mode in, 54
- command list for, 27
- flexible layouts, testing in, 282–285
- Help command, 28
- pinch/zoom touch mode, 27
- resolution, changing, 212, 284
- Rotating, commands for, 27

Windows Application Cert Kit, 108
Windows.ApplicationModel.winmd, 138
Windows+C keyboard shortcut, 11
Windows Communication Foundation (WCF), 143
Windows.Data.winmd, 138
Windows.Devices.winmd, 138
Windows+F keyboard shortcut, 11
Windows.Foundation.winmd, 138
Windows.Globalization.winmd, 138
Windows.Graphics.winmd, 138
Windows+H keyboard shortcut, 11

Windows+I keyboard shortcut, 11
Windows Live IDs, 105, 298
Windows.Management.winmd, 138
Windows.Media.winmd, 138
Windows Metadata. *See* WinMD (Windows Metadata)
Windows.Networking.winmd, 139
Windows Notification Service (WNS), 294
Windows Presentation Foundation (WPF) applications, 99
  Windows 8 applications vs., 70–71
Windows+Q keyboard shortcut, 11
Windows Registry, WinRT and, 150–153
Windows Runtime APIs (WinRT). *See* WinRT
Windows Runtime Component template, 67
Windows Runtime core engine, 135
Windows.Security.winmd, 139
Windows Server 2012
  Windows 8 and, 1
Windows.Storage.FileIO.ReadTextAsync method, 236
Windows.Storage WinMD library (WinRT), 139, 316
Windows Store, 107
  account for, creating, 107
  certification requirements, 7
  fiscal data required for, 108
  information required about app, 110
  logo for app, in manifest, 102
  validation for, 108
  Windows Application Cert Kit, 108
Windows Store Apps
  App.xaml.cs file, 111
  guidelines for, 17
Windows Store app template (Visual Studio 2012), 111
Windows.System.winmd, 139
Windows.UI.ViewManagement.ApplicationView object
  flexible layouts and, 281
Windows.UI.winmd, 139
Windows.UI.Xaml.winmd, 139
Windows.Web.winmd, 139
WinJS
  Chakra engine and, 141
  consuming custom WinMD libraries with, 148–150
  WinRT and, 134
WinJS (Windows Library for JavaScript) library, 155
WinMD (Windows Metadata), 138–141
  C++, consuming custom libraries, 145–147
  custom libraries, requirements for, 143
  folder contents of, 138
  HTML5, consuming custom libraries with, 148–150
  libraries, creating, 143–150
  WinJS, consuming custom libraries with, 148–150
WinRT, 14–15, 133–154, 155–184
  activation types for, 120
  app registration, 150–153
  clients, basicHTTPbinding and, 320
  Contracts, 23–25, 171–183
  DataTransferManager class, 176
  design requirements for, 142–143
  enumerable collections, 135
  IInspectable interface, 140
  implementation of, 138–141
  .NET Framework v 4.5 and, 65
  numeric types for, 135
  pickers, 155–163
  Runtime Broker, 135
  schema, structure of, 140
  suspension and, 121
  types, registry keys for, 140
  versioning, 141
  webcam API, 163–171
  Windows.Storage WinMD library, 316
WMAppManifest.xml vs. Package.appxmanifest, 101
wsFederationHttpBinding, 320
wsHttpBinding, 320

X

XAML Controls
  expanding, 186
  layout system, 185–230
XML
  accessing nodes from .NET applications, 142
  Live Tiles and, 290
  viewing in Internet Explorer, 309

Z

ZoomedInView property (SemanticZoom control), 275
ZoomedOutView property (SemanticZoom control), 275

About the Authors

LUCA REGNICOLI is a consultant, trainer, and author who has specialized in user interface technologies for .NET applications since 2003. He developed the presentation tier of many enterprise applications in Windows Presentation Foundation, Silverlight, and Windows Phone. Luca is a co-founder of DevLeap, a company focused on providing high-value content and consulting services to professional developers. He is the author of a book in Italian language about ASP.NET. He is also a regular speaker at major conferences since 2001.

PAOLO PIALORSI is a consultant, trainer, and author who specializes in developing distributed applications architectures and Microsoft SharePoint enterprise solutions. He is the author of about 10 books, which include *Programming Microsoft LINQ in Microsoft .NET Framework 4* and *Microsoft SharePoint 2010 Developer Reference*. Paolo is a founder of DevLeap, a company focused on providing content and consulting to professional developers. He is also a popular speaker at industry conferences.

ROBERTO BRUNETTI is a consultant, trainer, and author with experience in enterprise applications since 1997. Roberto is a founder of DevLeap, together with Paolo Pialorsi, Marco Russo, and Luca Regnicoli, a company focused on providing high-value content and consulting services to professional developers. He is the author of a couple of books: one about ASP.NET, published in 2003, another about Windows Azure Beta, and the last one on Windows Azure published by Microsoft Press in 2011. He is also a regular speaker at major conferences since 1996 and he works closely with Microsoft in events and training courses.

What do you think of this book?

We want to hear from you!

To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

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

Thank you in advance for your input!

Microsoft Press