



Professional SharePoint® 2010 Cloud-Based Solutions

Steve Fox, Donovan Follette, Girish Raja, Paul Stubbs

PROFESSIONAL SHAREPOINT® 2010 CLOUD-BASED SOLUTIONS

INTRODUCTION	xxiii
CHAPTER 1 Getting Started with SharePoint and the Cloud.....	1
CHAPTER 2 Using SQL Azure for Business Intelligence.....	21
CHAPTER 3 Building a Windows Azure-Based Service	51
CHAPTER 4 Creating an Aggregated Solution Using Cloud-Based Data	83
CHAPTER 5 Connecting LinkedIn and SharePoint Profile Data.....	113
CHAPTER 6 Using Twitter in Your SharePoint Solutions	139
CHAPTER 7 Using Bing Maps in Your SharePoint Business Solutions.....	173
CHAPTER 8 Financial Modeling with Excel Services and Windows Azure	205
CHAPTER 9 Creating a Training Application in SharePoint Online.....	257
CHAPTER 10 Managing Customer Information in SharePoint Online.....	275
CHAPTER 11 Securing Cloud Solutions Using Claims-Based Authentication	309
INDEX	337

PROFESSIONAL

SharePoint® 2010 Cloud-Based Solutions

PROFESSIONAL

SharePoint® 2010 Cloud-Based Solutions

Steve Fox
Donovan Follette
Girish Raja
Paul Stubbs



WILEY

John Wiley & Sons, Inc.

Professional SharePoint® 2010 Cloud-Based Solutions

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2012 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-07657-6
ISBN: 978-1-118-22265-2 (ebk)
ISBN: 978-1-118-22967-5 (ebk)
ISBN: 978-1-118-22951-4 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Not all content that is available in standard print versions of this book may appear or be packaged in all book formats. If you have purchased a version of this book that did not include media that is referenced by or accompanies a standard print version, you may request this media by visiting <http://booksupport.wiley.com>. For more information about Wiley products, visit us at www.wiley.com.

Library of Congress Control Number: 2011939643

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. SharePoint is a registered trademark of Microsoft Corporation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

To my mother and father.

— STEVE

To Lorie, my love always.

—DONOVAN

*To my wonderful parents and loving wife, Seetha, for
always being there for me.*

—GIRISH

To my wife, Rosa

—PAUL

ABOUT THE AUTHORS



STEVE FOX is the Director of Developer & Platform Evangelism (DPE) for SharePoint and Windows Azure. He regularly speaks to developers about SharePoint, Office, and cloud development, at both domestic and international conferences. Steve has authored many articles and books, including *Beginning SharePoint 2010 Development* (Wrox, 2010) and *Developing Microsoft SharePoint Applications on Windows Azure* (MSPress, 2011). He has worked at Microsoft for 12 years and currently lives in Seattle, Washington.



DONOVAN FOLLETTE is a Sr. Technical Evangelist for Microsoft with more than 25 years of experience developing enterprise business applications on a variety of platforms. He currently specializes in helping to enable developers in building integrated line-of-business (LOB) solutions with Microsoft Office and SharePoint 2010.



GIRISH RAJA is a Technical Evangelist at Microsoft's Redmond campus. In this role, Girish helps the community learn various development features of SharePoint and provides guidance for building applications using its platform. He also works on demonstrating how well SharePoint integrates with other Microsoft technologies through his blog and various channel9 videos. Prior to this role, he was an Evangelist for Dynamics CRM.



PAUL STUBBS is a Microsoft Technical Evangelist for Azure, SharePoint, and Office, where he focuses on the information worker development community around SharePoint and Office, Silverlight, and cloud computing. He has authored several books on solution development using Microsoft Office, SharePoint, and Silverlight, as well as several articles for *MSDN Magazine*, and has also spoken at Microsoft Tech-Ed, PDC, SharePoint Conference, DevConnections, and MIX conferences. Paul has also worked as a Senior Program Manager on Visual Studio in Redmond, Washington. Paul is a Microsoft Certified Trainer (MCT) and frequently participates in the developer community on the Microsoft forums. Visit Paul's blog at blogs.msdn.com/pstubbs for deep SharePoint developer information.

ABOUT THE TECHNICAL EDITOR



LARRY RIEMANN has more than 17 years of experience architecting and creating business applications for some of the world's largest companies. Larry is an independent consultant who owns Indigo Integrations and does SharePoint consulting exclusively through SharePoint911. He writes articles for publication, is a co-author of *Professional SharePoint 2010 Branding and User Interface Design* (Wrox, 2010) and a contributing author on *Professional SharePoint 2010 Administration* (Wrox, 2010), and is an occasional speaker. For the last several years he has been focused on SharePoint, creating and extending functionality where SharePoint leaves off. In addition to working with SharePoint, Larry is an accomplished .Net Architect and has extensive expertise in systems integration, enterprise architecture, and high-availability solutions. You can find him on his blog at <http://lriemann.blogspot.com>.

CREDITS

ACQUISITIONS EDITOR

Paul Reese

PROJECT EDITOR

John Sleeva

TECHNICAL EDITOR

Larry Riemann

PRODUCTION EDITOR

Kathleen Wisor

COPY EDITOR

Luann Rouff

EDITORIAL MANAGER

Mary Beth Wakefield

FREELANCER EDITORIAL MANAGER

Rosemarie Graham

ASSOCIATE DIRECTOR OF MARKETING

David Mayhew

MARKETING MANAGER

Ashley Zurcher

BUSINESS MANAGER

Amy Knies

PRODUCTION MANAGER

Tim Tate

**VICE PRESIDENT AND EXECUTIVE GROUP
PUBLISHER**

Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER

Neil Edde

ASSOCIATE PUBLISHER

Jim Minatel

PROJECT COORDINATOR, COVER

Katie Crocker

PROOFREADER

Jen Larsen, Word One

INDEXER

Robert Swanson

COVER DESIGNER

Ryan Sneed

COVER IMAGE

© iStock / Stephen Goodwin

ACKNOWLEDGMENTS

THANKS TO PAUL REESE AND JOHN SLEEVA for helping manage and drive this book to its completion. Also, thanks to Paul, Donovan, and Girish for helping create what is a growing evolution for the SharePoint community. And, lastly, thanks to the many folks at Wrox that were a part of making this book a completed product.

—STEVE FOX

IT'S ALWAYS A PLEASURE TO BE PART of a great team, and that has been my experience as a member of the Microsoft Developer & Platform Evangelism Productivity team. I want to thank Steve Fox for his exceptional leadership, and Paul Stubbs, Chris Mayo, Girish Raja, and Bruno Nowak for being amazing colleagues to work alongside. I'd also like to call out thanks to my former colleague in the Identity space, Vittorio Bertocci. You have all enriched my life, and I have gleaned much from each of you, both personally and through your technical expertise. Lastly, thanks to John Sleeva and the editors for their patience and help in bringing this book to you, the reader. Thank you for reading it — I trust it will open up new opportunities to land amazing productivity solutions with SharePoint and Azure. Enjoy!

—DONOVAN FOLLETTE

MANY THANKS TO STEVE FOX for encouraging me to do this. I'd also like to thank Donovan Follette and Paul Stubbs for sharing those small tips that made a big difference to me. Writing a book is never easy, and I owe a lot to my brother, Seenu, and all my family and friends for their never-ending support and guidance.

—GIRISH RAJA

I WOULD LIKE TO THANK STEVE FOX for leading me into the cloud space and continually pushing me to be better. Over the years, Steve and I have worked together on many writing projects, and it is always a very educational experience for me. Steve and I are always looking to learn and talk about the next technology that is coming over the horizon. And in this case, Windows Azure and cloud computing is the next wave that every developer must be fluent in; this is just the beginning of the total industry shift to the cloud paradigm. I would also like to thank the other co-authors, Donovan and Girish. It has been a pleasure working with both of them on SharePoint, Office, and CRM over the last couple of years. They are both always ready to help and offer assistance when things need to get done. It is a breath of fresh air to work with people who are all working selflessly towards a common goal and who have a passion for the productivity community.

—PAUL STUBBS

CONTENTS

INTRODUCTION

xxiii

CHAPTER 1: GETTING STARTED WITH SHAREPOINT AND THE CLOUD 1

Overview of the Cloud	1
Windows Azure	3
Web 2.0	4
Bing Services	5
Microsoft Dynamics CRM	6
Understanding the Importance of the Cloud	6
Integrating the Cloud with SharePoint	8
Setting Up Your Development Environment	9
Setting Up a Virtualized Environment	9
Installing Hyper-V	10
Setting Up a Native Environment	11
Creating Your First SharePoint Cloud Application	13
Summary	19
Additional References	20

CHAPTER 2: USING SQL AZURE FOR BUSINESS INTELLIGENCE 21

Overview of SQL Azure	21
Uses of SQL Azure	22
Migrating SQL Server to SQL Azure	22
Interacting with SQL Azure with SQL Server 2008 R2 Management Studio	23
BI Solutions Using SQL Azure	25
No-Code BI Solutions in SharePoint	25
Code-Based Solutions in SharePoint	26
Creating a BI Dashboard Using SQL Azure and SharePoint	27
The Solution Architecture	27
Creating the Sales BI Application	28
Creating the SQL Azure Database	29
Creating an External List	31
Creating the WCF Service	35
Creating the Sales Web Part	40
Creating the Sales Dashboard UI	46

Summary	48
Additional References	49
CHAPTER 3: BUILDING A WINDOWS AZURE-BASED SERVICE	51
<hr/>	
Getting Started	52
Installing SharePoint 2010	52
Installing Windows Azure SDK and Tools	52
Creating an SSH Tunnel	52
Initializing the Windows Azure Compute and Storage Emulators	53
Using Windows Azure Table Storage	54
Creating the Windows Azure Web Role	55
Creating the Entity Data Model	56
Populating the Windows Azure Table	58
Creating a WCF Service Endpoint	69
Building a Windows Azure Service	69
Accessing Table Storage Data	71
Creating a SharePoint BCS External Content Type	77
Consuming a Windows Azure Service Using BCS	77
Summary	81
Additional References	82
CHAPTER 4: CREATING AN AGGREGATED SOLUTION USING CLOUD-BASED DATA	83
<hr/>	
Overview of Mashups	83
Creating Mashups Using Yahoo Pipes	84
Using the Pipes Editor	85
Creating Pipes	87
Consuming Pipes	89
Joining Multiple Feed Sources	92
Sorting and Filtering Feed Sources	94
Using Existing Pipes	97
Extending Your Pipes with Yahoo Query Language	101
Using the YQL Console	102
Creating a YQL Pipe Web Part	109
Summary	111
Additional References	111

CHAPTER 5: CONNECTING LINKEDIN AND SHAREPOINT PROFILE DATA	113
Overview of SharePoint Social Computing	114
My Site	114
SharePoint Profiles	115
Adding Custom User Profile Properties	116
Social Tagging and Notes	117
Activity Feeds	118
Blogs	118
Wikis	119
Understanding the LinkedIn Social Networking Programming Model	120
LinkedIn Profiles	121
Creating a LinkedIn Silverlight Web Part	121
Authenticating in LinkedIn	123
Reading LinkedIn Profile Properties	124
Reading SharePoint Profile Properties	128
Posting LinkedIn Status Messages	133
Summary	137
Additional References	138
CHAPTER 6: USING TWITTER IN YOUR SHAREPOINT SOLUTIONS	139
Overview of Twitter	139
Developing Applications Using Twitter	141
Using OAuth in Your Twitter Applications	141
Building Applications That Require Authorization	142
Building Applications That Do Not Require Authorization	147
Using the Twitter Trends Data	149
Integrating Twitter with SharePoint	152
The Solution Architecture	152
Creating the Twitter and SharePoint Solution	153
Creating the Twitter Service	153
Creating the Silverlight Client Application	166
Deploying the Silverlight Application	171
Summary	172
Additional References	172

CHAPTER 7: USING BING MAPS IN YOUR SHAREPOINT BUSINESS SOLUTIONS	173
Overview of Bing Maps	173
Bing Maps Controls	178
Leveraging Geo-Location Web Services	179
Integrating the Bing Maps API with SharePoint	180
Leveraging the Client Object Model	181
The Solution Architecture	182
Creating a SharePoint Solution Using Bing Maps	183
Creating the Bing Map Application	183
Creating the Application UI	186
Creating a Store and Sales List in SharePoint	200
Deploying the Silverlight Bing Application	201
Summary	203
Additional References	204
CHAPTER 8: FINANCIAL MODELING WITH EXCEL SERVICES AND WINDOWS AZURE	205
Overview of Excel Financial Models	206
Integrating Windows Azure with Office	207
Office Clients and Azure	208
Office Services and Azure	209
The Solution Architecture	209
Accessing Azure Table Data via WCF Data Services	211
Using a WCF Service to Provide Azure Table Data	219
Extending an Excel Financial Model to Leverage Azure Table Data	236
Creating an Excel Client Add-In	237
Using the ECMAScript Object Model with the Excel Web Access Web Part	245
Summary	254
Additional References	255
CHAPTER 9: CREATING A TRAINING APPLICATION IN SHAREPOINT ONLINE	257
Overview of the Training Application	258
The Solution Architecture	258
Understanding the SharePoint Online Platform	258
Scalable Storage in the Cloud with Azure	260

Building the Video Library	260
Creating the Data Model in SharePoint Online	261
Uploading Videos to Windows Azure	262
Creating an Integrated User Experience	266
Creating the Video Player	267
Enabling Social Experiences in Training	271
Summary	274
Additional References	274
 CHAPTER 10: MANAGING CUSTOMER INFORMATION IN SHAREPOINT ONLINE	 275
Overview of Microsoft Dynamics CRM	276
The Solution Architecture	279
Using Silverlight with SharePoint	280
Security and Authentication	281
Building the Dashboard	281
Connecting Document Libraries to CRM Online	282
Building the Windows Azure Proxy	283
Creating the Silverlight Grid and Chart	291
Bringing It All Together	306
Summary	308
Additional References	308
 CHAPTER 11: SECURING CLOUD SOLUTIONS USING CLAIMS-BASED AUTHENTICATION	 309
Overview of Claims-Based Identity	310
WS-Federation vs. WS-Trust	311
The Power of Claims-Based Identity	312
SharePoint and Claims	313
Relying Party Applications, Federation Providers, and Identity Providers	314
Relying Party Applications	315
Window Azure AppFabric Access Control Service	316
Active Directory Federation Services v2.0	316
The Solution Architecture	316
Accessing Claims-Based Azure Services from SharePoint	318
Setting Up an Access Control Service Namespace	318
Establishing AD FS v2.0 As a Trusted Identity Provider for ACS	318
Creating a Relying Party Azure Web Application	321

CONTENTS

Wiring the Azure Web Application to Trust ACS	325
Accessing the Azure Web Application from SharePoint via Claims-Based Authentication	333
Summary	335
Additional References	335
<i>INDEX</i>	337

INTRODUCTION

THE “CLOUD” IS ONE OF THE MOST tossed-around words these days in any IT conversation. But what does it mean? And what are the implications of the cloud for SharePoint? In fact, there are many implications for SharePoint with the cloud, ranging from extending existing solutions to use code or services that are deployed to the cloud to getting code off of the server and having that live in the cloud. And each of these is interesting in its own way and more generally marks the importance of cloud computing as an area.

In this book, we hope to shine some light not only on what the cloud and cloud computing mean, but also what it means to integrate the cloud with SharePoint. We’ll focus predominantly on SharePoint on-premises (or SharePoint Server 2010), but you’ll see there are many lessons that can be applied to SharePoint 2007 or, in some cases, to SharePoint Online (within the new Office 365 offering).

WHO THIS BOOK IS FOR

This book is primarily aimed towards SharePoint developers, IT pros, or enthusiasts who want to better understand different ways to integrate the cloud with their SharePoint installation(s). That said, you will get more out of this book if you’re a SharePoint developer who has had at least six month’s experience with SharePoint. You should also have some understanding of ASP.NET and broader web-development technologies. If you’re looking for an introduction to SharePoint, you might want to check out *Beginning SharePoint 2010 Development* (Wrox, 2010) or visit Microsoft’s developer resource site, at <http://sharepoint.microsoft.com/en-us/resources/Pages/Developer-Training-Guide.aspx>.

Each chapter is designed to shed some light on a specific type of cloud integration with SharePoint — ranging from Windows Azure to Web 2.0. Furthermore, we’ve structured each chapter in a similar way. For example, each chapter provides a small conceptual overview of the cloud technology. It then walks through a high-level solution architecture, which then segues into a practical walkthrough, where you get to put theory into practice. Thus, the goal for each chapter is to walk from concept to reality.

WHAT THIS BOOK COVERS

This book covers an array of cloud-based technologies and shows you how you can integrate them with SharePoint. The different cloud technologies that are covered are as follows:

- Windows Azure
- SQL Azure
- Bing Maps

- LinkedIn
- Twitter
- Excel Services
- Windows Azure Blob and Table services
- Microsoft Dynamics CRM

This book covers these areas by describing and walking through practical solutions with the end goal of providing some simple applications that integrate the cloud with SharePoint.

The chapters within this book are summarized as follows:

- **Chapter 1: Getting Started with SharePoint and the Cloud** — Provides an introduction to the concept of the cloud and cloud computing, introduces the main cloud technologies discussed in the book, and walks through a Hello World cloud-based application.
- **Chapter 2: Using SQL Azure for Business Intelligence** — Describes the ways in which you can create business intelligence solutions with SQL Azure and SharePoint and walks you through how to integrate external lists with SQL Azure using Business Connectivity Services (BCS) and then create a custom ASP.NET chart with SQL Azure data.
- **Chapter 3: Building a Windows Azure-Based Service** — Provides some background on the power of deploying your services and data to Windows Azure and teaches you how to leverage open-source government county tax data in Windows Azure and consume it in your SharePoint applications using BCS.
- **Chapter 4: Creating an Aggregated Solution Using Cloud-Based Data** — Describes how to decouple Internet data sources from your SharePoint application to easily locate, query, and manipulate cloud-based data through a composite application that uses SharePoint, Yahoo Pipes, and Yahoo Query Language.
- **Chapter 5: Connecting LinkedIn and SharePoint Profile Data** — Describes the importance of Web 2.0 and shows you how to extend SharePoint's social computing capabilities to integrate with LinkedIn.
- **Chapter 6: Using Twitter in Your SharePoint Solutions** — Provides an overview of how to use the Twitter REST APIs and then walks through how to create a simple trend report using Twitter data for SharePoint.
- **Chapter 7: Using Bing Maps in Your SharePoint Business Solutions** — Describes how you can get started with Bing Maps and geo services and then walks through how you can leverage the Bing Maps APIs to load and display SharePoint list data on a map.
- **Chapter 8: Financial Modeling with Excel Services and Windows Azure** — Describes the patterns for accessing Windows Azure table storage data from both an Excel client add-in and on SharePoint using Excel Services and its JavaScript object model.

- **Chapter 9: Creating a Training Application in SharePoint Online** — Walks you through building a simple application in SharePoint Online that holds a list of online training videos from Windows Azure and describes how to customize that list to provide a social viewing experience for the videos.
- **Chapter 10: Managing Customer Information in SharePoint Online** — Provides an overview of CRM Online and describes how to build a simple SharePoint Online application that has a unified view of customer data from CRM Online.
- **Chapter 11: Securing Cloud Solutions Using Claims-Based Authentication** — Shows how the Windows Identity Foundation (WIF), Active Directory Federation Services (ADFS), and the Windows Azure Access Control Services (ACS) come together to enable a seamless SSO end-user experience when navigating across Azure-based web applications, on-premises SharePoint, or a mash-up of both.

Each chapter provides sample code that will help you walk through the chapters and give you a sense for how things work under the covers. Treat the sample code as exemplifying patterns as, opposed to code that you should inject directly into your production environment.

WHAT YOU NEED TO USE THIS BOOK

While Chapter 1 provides you with an overview of the software you'll need to install to work through the samples (or where you can get pre-created environments, such as the Information Worker Virtual Machine or Easy Set-up Script), the core software that you'll want to install are as follows:

- Windows Server 2008 R2 (or Windows 7)
- SQL Server 2008 R2
- Microsoft Office 2010 Professional Plus
- SharePoint Designer 2010
- Visual Studio 2010 Professional
- Windows Azure SDK and Tools for Visual Studio
- SharePoint Server 2010

For each chapter, you may find that additional software must be installed, or developer keys obtained, to complete the exercises. For example, to complete the Bing Maps exercises, you'll need to obtain a developer key to use in your application. Each of these additional requirements will be outlined in the chapters.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.



The pencil icon indicates notes, tips, hints, tricks, or and asides to the current discussion.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

We use a monofont type with no highlighting for most code examples.

We use bold to emphasize code that is particularly important in the present context or to show changes from a previous code snippet.

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually, or to use the source code files that accompany the book. All the source code used in this book is available for download at www.wrox.com. When at the site, simply locate the book's title (use the Search box or one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book. Code that is included on the Web site is highlighted by the following icon:



Available for
download on
Wrox.com

Listings include the filename in the title. If it is just a code snippet, you'll find the filename in a code note such as this:



Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-1-118-07657-6.

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at www.wrox.com/dynamic/books/download.aspx to see the code available for this book and all other Wrox books.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time, you will be helping us provide even higher quality information.

To find the errata page for this book, go to www.wrox.com and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list, including links to each book's errata, is also available at www.wrox.com/misc-pages/booklist.shtml.

If you don't spot "your" error on the Book Errata page, go to www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At p2p.wrox.com, you will find a number of different forums that will help you, not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.



You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

PROFESSIONAL

SharePoint® 2010 Cloud-Based Solutions

1

Getting Started with SharePoint and the Cloud

WHAT'S IN THIS CHAPTER?

- Understanding the cloud and its importance
- Integrating the cloud with SharePoint
- Setting up your development environment
- Creating your first SharePoint and cloud application

In this first chapter, you'll learn about what the cloud is and why it's important. You'll also learn about the set of cloud technologies that this book uses to integrate with SharePoint: Windows Azure, Web 2.0, Bing Maps, and Microsoft Dynamics CRM. To get started with your development, you'll walk through how to get your development environment up and running, using one of two options: a native installation or a prepped virtual machine. Once your development environment is up and running, you'll finish the chapter by creating your first cloud-based application for SharePoint.

OVERVIEW OF THE CLOUD

Increasingly, “the cloud” is being discussed as a major inflection point for the next generation of IT. In fact, depending on what you read, “inflection point” might be an understatement; research institutions like Gartner and Forrester are characterizing the cloud as a “disruptive shift.” This is because of the shift from the traditional software development and deployment process (which predominantly lives in an on-premises data center or lab) is evolving more towards a services-driven approach in which software lives in the cloud.

Regardless of how we characterize the cloud, though, we must answer a common set of baseline questions before we can even think about building our first cloud application. For example, what is the cloud? How can you engage with it? How will it affect you and your business? And, critical to this book, how will it impact (or augment) your SharePoint practice and solutions?

While these seem like fairly basic questions, in reality the answers are not necessarily simple ones. For example, a recent *InfoWorld* article stated “. . . the problem is that . . . everyone seems to have a different definition” of the cloud and cloud computing (www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031). As the article points out, often the cloud is a metaphor for the Web — itself a vastly connected set of resources — which begs the question of where the cloud, or cloud computing, starts and where it ends. Despite the many definitions of the cloud and cloud computing, we should agree for the purposes of this book that the cloud does indeed represent a metaphor for the Web and that cloud computing represents using the Web as a connected set of resources to build and deploy your software.

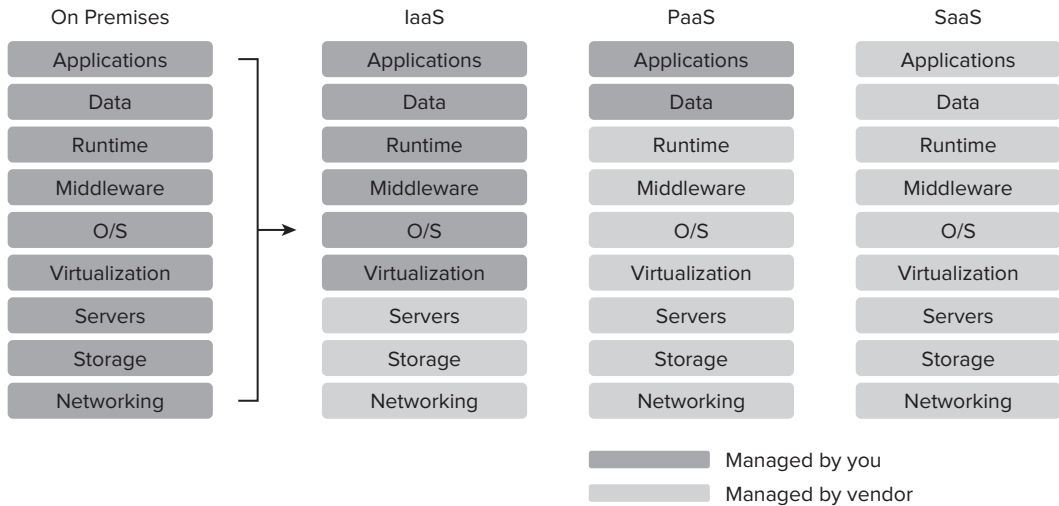
Deploying your software to the Web represents a model for ubiquitous application development and deployment. Among the advantages that it provides are the following:

- Services
- Shared computing resources and storage
- Quick provisioning of hardware and software
- Minimal management of IT infrastructure and software
- Reduced overhead and improved cost optimization

The goals of cloud computing include increasing your application’s speed to market; avoiding the hurdles that current IT departments currently face with ordering and managing hardware, deploying and managing software; and increasing or decreasing capacity based on real-time application demand. In essence, software deployment and management are moving off premises to data centers throughout the world.

Today, you can find many examples of cloud computing, including technologies such as Microsoft Dynamics CRM, Microsoft Office 365, Hotmail, SkyDrive, Salesforce.com, Google Apps, Amazon AWS, and Windows Azure. However, how are these cloud-based technologies different from what you’re managing on-premises today? For example, consider Figure 1-1. The leftmost column shows a typical representation of what a business manages within its on-premises data centers or labs: You run the applications, the staff, and the resources to manage the software and hardware; you have people building and deploying software into that environment; and so on. However, as you move right across the diagram, various services can handle some of these management needs.

IaaS (Infrastructure as a Service) provides you with virtualization capabilities, such as a virtual machine (VM) instance that enables you to host machines in the cloud. Amazon AWS and Windows Azure offer this type of cloud solution. In this scenario, you are beginning to manage fewer on-premise tasks, moving some of your computing into the cloud. PaaS (Platform as a Service) enables you to deploy code or data to the cloud. Windows Azure offers this type of service, as does Google. Here again, you’re moving more of your computing and data into the cloud.

**FIGURE 1-1**

Finally, SaaS (Software as a Service) provides you with everything you need, so you don't have to manage any on-premise IT infrastructure. This is a compelling option for many companies because you can choose to have everything managed for you or you can do it through browser-based applications that are directly tied to your cloud offering. Microsoft Office 365 and Salesforce.com are examples of this type of cloud offering.

The key takeaway here is that cloud computing provides you with different ways to build and deploy an application or solution and manage it over time within the cloud.

In this book, we want to explore this practice of deploying code to the cloud against integrating that code with an on-premises (or cloud-based) instance of SharePoint. Specifically, we want to discuss how a number of popular cloud technologies integrate with SharePoint Server 2010; therefore, a range of different cloud services are discussed. Of the cloud models just described, we'll focus predominantly on PaaS; that is, we'll either consume existing services, resources or data that already live in the cloud or create new services or data structures and deploy them to the cloud. These cloud technologies will cut across Windows Azure, Web 2.0, Bing Services, and Microsoft Dynamics CRM. The cloud technologies will then be tied back into SharePoint to demonstrate how you can extend your on-premise SharePoint solutions into the cloud.

Windows Azure

Windows Azure, Microsoft's cloud-computing platform, offers all the standard service types discussed earlier: IaaS, PaaS, and SaaS. Windows Azure is a flexible cloud-computing platform that provides services for virtualizing VMs, managing resources, data, and services, and building cloud-based applications and websites. As Microsoft's key cloud strategy, Windows Azure enables you to provision and scale resources to the cloud without having to worry about chasing and managing your on-premise hardware. When you use Windows Azure, for example, you not only get application scale (hardware needs expanding as your data and application needs grow), but you also get patching and operating-system management (your cloud-hosted environment is always up to date),

and a 99.9% uptime guarantee. As a developer, you'll also have a developer portal, which you can use to manage your applications.

Windows Azure currently comprises three main constituent parts:

- **Windows Azure** — Windows Azure offers a core set of service-management capabilities, developer tools, and a portal through which you can configure your applications, data and services, and different types of storage (table, BLOB and queue) for nonrelational data. In addition, the Windows Azure Marketplace DataMarket enables you to integrate directly with subscription-based data that can be consumed programmatically or via the Marketplace browser UI.
- **SQL Azure** — You can think of SQL Azure as SQL Server's sister product for the cloud. SQL Azure represents relational data storage for the cloud. Using SQL Azure, you can migrate or build relational databases that provide rich and queryable data sources for your cloud-based or hybrid applications. You can also synchronize your on-premise data with that which lives in SQL Azure and use the same skills you know today to integrate SQL Azure data with SharePoint.
- **Windows Azure AppFabric** — Windows Azure AppFabric offers a broad set of middleware services that help you manage security and identity (Access Control Service), service-based applications, workflow, and more. This set of middleware services provides a compelling way to connect on-premise data to your cloud-based solutions. One example of the power of these services is surfacing line-of-business (LOB) data that lives in your data centers or in your SharePoint instances to Office 365 or remote devices through the Windows Azure AppFabric service bus.

Together, these three parts provide a rich and deep set of technologies that enable you to manage your application development and deployment needs.

You'll be using Windows Azure in a number of chapters in the book, so you'll want to create an account for yourself. To do this, visit <https://windows.azure.com/default.aspx>. After you've created an account, you'll have access to the Windows Azure portal. Download the Windows Azure SDK and Visual Studio tools. At the time of writing, you can find the SDK and tools at www.microsoft.com/download/en/details.aspx?id=15658.

Web 2.0

Web 2.0 technologies are all about the social Web, and it's likely that many of you who are reading this book are already participating in Web 2.0. (In fact, some of you have likely seen articles on Web 3.0, or what is more commonly called the *semantic Web*.) Examples of Web 2.0 technologies include Facebook, LinkedIn, Twitter, and Google+. These technologies, and many others, are typically browser based (although many companies are building device-specific or rich-client applications that leverage Web 2.0 APIs), and they provide places for people to commune and collaborate across different contexts on the Web. For example, LinkedIn is a professional Web 2.0 site that lends itself well to people connecting and collaborating through their professional expertise. Conversely,

Facebook is more personal, although many companies have built Facebook sites to introduce their products and services into the virtual conscience. Twitter provides 140-character “tweets” that enable members to quickly communicate news, gossip, events, and so on. According to the company, more than 60 million tweets are sent every day.

While many Web 2.0 sites can operate in and of themselves, they typically have functionality that can be accessed programmatically and then integrated within SharePoint. For example, consider the huge volume of messages that are sent through Twitter; tweets are valuable data for many companies because they represent a collective, virtual conscience that can be measured both qualitatively and quantitatively. You can build small apps as part of a wider portal to view trending for specific topics. In addition, you might want to integrate LinkedIn profiles with your in-company profiles. Because many Web 2.0 applications and sites have open APIs, it’s entirely possible to build interesting integrations with SharePoint, given its collaborative nature.

In this book, you’ll build applications that live in SharePoint but integrate with LinkedIn and Twitter as two examples of leveraging Web 2.0.

Bing Services

While Windows Azure provides you with a platform to build and deploy your own custom services, Bing provides a set of services that are ready to consume. Because we’re living in an age in which we’re saturated with information, having ways to navigate that information, or even provide more context for the information we have, is increasingly important.

Bing offers a broad array of services, ranging from search to geo-mapping services. The Bing services used in this book predominantly revolve around the mapping and geo-location services. To access their services, Bing provides a rich set of APIs that you can use to programmatically integrate Bing functionality into your applications. For example, imagine that you have a list of customers and contact information in SharePoint but you want to be able to map those customers and then provide auto-generated driving directions for your customer visits. You can integrate Bing and SharePoint to provide this integrated experience with the native services provided by Bing, which are often provided in both REST-based and SOAP-based services.

As mentioned earlier, Bing provides a broad array of services (some of which are browser-based and some of which can be accessed programmatically) such as images and search services (which Bing is commonly known for), news, related search (search-filtering capabilities), translation, video, and flight status. The flight status service is another one of the Bing services that can be accessed programmatically. For example, if you have a list of the trips your employees are taking, you can build a small application that checks the status of the flight service, and then integrate it with SharePoint. Some of these services are being launched and managed through the Windows Azure Marketplace DataMarket as an additional way to subscribe to and use these increasingly important and data-driven services.

Before you can develop applications for Bing, you need to obtain a developer key. You can do so by visiting the developer portal, at www.microsoft.com/maps/developers/web.aspx. You’ll also find key links to SDK documents and code samples.

Microsoft Dynamics CRM

Microsoft Dynamics CRM is a multilingual customer relationship management (CRM) software package that is deployable both on-premise and in the cloud. CRM is a little misleading because Dynamics CRM can manage many different types of data, not just sales and marketing data. In fact, CRM is an excellent data-driven application for many types of data. With the latest release (2011), Microsoft has significantly enhanced the capabilities of Dynamics CRM to include both out-of-the-box functionality in the cloud and the capability to extend it with custom solutions through the XRM platform.

What makes CRM powerful is not just the capability to quickly create “entities” that represent your customer data, but also the fact that Dynamics CRM has a rich set of web services that enable you to programmatically access this data from other applications. This enables you to use Dynamics CRM for what it’s intended, data-driven and business-process applications, but still consume the data in collaborative environments like SharePoint.

You can access the Microsoft Dynamics CRM web services using a number of different methods in SharePoint. For example, you could consume the web services in common SharePoint artifacts, such as web parts and event receivers; or, if you want to deploy cloud-based applications, you could use JavaScript, jQuery, or Silverlight.

UNDERSTANDING THE IMPORTANCE OF THE CLOUD

So far, you’ve been introduced to the concept of the cloud and cloud computing and some of the technologies we’ll be discussing within this book. Let’s spend a few minutes talking about why the cloud is so important.

One of the key benefits of the cloud is cost; it is less expensive than running your on-premise infrastructure. For example, in the case of Windows Azure, you’re paying for *usage*, as opposed to hardware and support costs, so you don’t wind up paying a cost rate of 100% for something you’re only using 20% of the time. This makes the cloud model very compelling for companies of all sizes, as any business wants to reduce IT costs but still maintain the capability to deploy and service applications.

Another key benefit that the cloud offers is speed and agility. That is, you can create and deploy applications much more quickly now than in the past. For example, with the quick provisioning of a web role in Windows Azure, you can have a complete web application deployed in a matter of a few minutes. For many subscription services, such as Facebook or Salesforce.com, you can also have accounts up and running and be using the software within a few minutes. For internal, departmental applications, speed and agility is great; it enables you to free up IT resources for all those projects that are triaged below the waterline. However, for external applications that end up in a marketplace, speed and agility suddenly become super-important; they are the difference between getting first or late to the market with either a new software product or updates to your existing products.

Thus, the cloud offers a competitive advantage by enabling companies to get their software to users quicker and, in many cases, in a way that can be monetized through a marketplace.

Reuse is another benefit of the cloud. For example, suppose you have an application that you've deployed internally and that you want to expose as an offering to your partners. The cloud is a viable method for sharing that application. Furthermore, with the ever-growing need to support multiple devices, reusability is an imperative — at least with the core application code. The cloud offers a place to deploy applications that can then be accessed by many different devices and form factors — both on-premise and cloud-based.

Another benefit is the capability to tap into a lot of the Web's ongoing creativity. Developers of all ages and backgrounds are engaging in software development, making some very compelling applications and services that are increasingly available for subscription use. These applications are often made available to you via services (such as REST-based or WCF services), so integration with your existing applications is easy to provision (through either a free or paid subscription license, for example) and you don't need to maintain any code (although you do need to track these third-party updates to ensure that your integrations don't break).

While there are many benefits to the cloud, it poses some challenges as well. For example, managing identity in the cloud can be more of an art than a science. With advances in, and increased adoption of, technologies such as OAuth and OpenID, an increasing number of developers and applications are using open standards. Furthermore, with ADFS 2.0, Windows Identity Federation (WIF), and Windows Azure's Access Control Service, you have a wide variety of options for managing your identity needs — a very important aspect of interacting with the cloud.

Working hand-in-glove with identity is security, and this cuts across many different fronts. Every organization has core datasets to protect; we're all aware of cases in which information such as credit card data is being hacked and used for nefarious purposes. Companies also need to protect employee data, sales data, prospect data, and other types of sensitive information. By putting this data in the cloud, companies must implement what are an evolving set of security measures and controls.

Security may be one pivot, but regulation and policy can also prohibit companies from putting their data in the cloud. Many different efforts are underway around the world to understand how regulation prohibits where data can be stored and how companies can work within regulation and policy to manage their resources within the paradigm of cloud computing. What's interesting is that while many organizations are regulated via either policy or law regarding where data resides, they are simultaneously looking for ways to surface views of the data so they have reporting mechanisms in the cloud. Thus, they can store their data on-premise but comply with regulations and laws by only surfacing sliced views of that on-premise data.

These challenges notwithstanding, there is significant cloud momentum. Companies are moving quickly to the cloud, which in turn is speeding development. Therefore, if you're not actively evaluating what the cloud can do for your organization in a broad sense, you may want to start. If you're unsure how to get started (especially in the context of SharePoint), that is what this book is about:

introducing you to how to integrate SharePoint with the cloud. By reading this book, you're beginning this process in the context of SharePoint.

INTEGRATING THE CLOUD WITH SHAREPOINT

Our assumption is that before picking up this book, you knew something about SharePoint. This book discusses how to build a short list of sample solutions that integrate in some way with the cloud — using the cloud technologies discussed in the previous section. At this point, you are likely wondering what the points of integration are. Because you're somewhat familiar with SharePoint, you already know that the answer to this question is wide and deep. However, let's try to whittle it down a little.

At a high level, you can integrate the cloud with many different SharePoint artifacts. For example, if you take the most common artifact, the web part, you can build custom web parts that provide some level of interactivity with the cloud. This could be a custom Twitter trends report, customer data driven from a CRM online system, or even be a profile viewer for sites like LinkedIn. What makes this development straightforward is that Visual Studio 2010 (and the community tools that are increasingly being released by the developer community) provides a rich set of web part templates for quick development, debugging, and deployment.

While the web part is one point of integration, it's not the only one. You can also use event receivers or workflows to reach out and query cloud-based services or applications and leverage that data from an application deployed to SharePoint. An example of this might be deploying custom services, such as a sales tracker, to Windows Azure. When events or workflow activities reach out to the service to query a specific sales figure, this can subsequently trigger actions in other parts of SharePoint. For example, a follow-up task may be created in SharePoint with a specific retail store when sales figures drop below a certain point. Nintex Live is a real-world example of this; it queries cloud-based services and then moves a workflow activity in a particular direction based on the information returned from that service.

This book provides numerous different examples that demonstrate how you can integrate the cloud with SharePoint. For example, you'll learn how you can use SQL Azure to build business intelligence solutions, and Windows Azure to build custom service scenarios that use Excel Services to service on-premise financial data. You'll also see examples that demonstrate how you can integrate Bing Maps with SharePoint to provide location information on data coming from a SharePoint list; how to integrate LinkedIn profile information with SharePoint and build Twitter trend reports; and how you can leverage ADFS to federate security to Windows Azure to seamlessly pass claims and authenticate users both on-premise and in the cloud. In short, each chapter serves as an introduction to how you can integrate different aspects of the cloud — be it customer information from Dynamics CRM or federated security — and walk through easy-to-follow solutions that illustrate an integrated pattern. Each chapter leverages a different aspect of SharePoint in some way, including security, web parts, and client-side applications such as Silverlight or JavaScript.

SETTING UP YOUR DEVELOPMENT ENVIRONMENT

You can set up your development environment in two ways. The first way is to download a virtual machine (called the *Information Worker Virtual Machine*, or IW VM) with everything installed on it. Microsoft has created this for you to use when you start developing for SharePoint; it is by far the easiest and the most fail-safe way to set up your development environment. (You will need Windows Server 2008 R2 to run the VM.) Although you may prefer the performance of your development efforts on a native installation, working within a VM for your development enables you to apply and revert to snapshots at any time you want. You'll realize that this one feature is very useful.

The second option is to install all the software you need locally on your machine (or natively, as we'll sometimes refer to it). The baseline software you're going to need to set up your development environment is as follows:

- A Windows 64-bit compliant operating system (e.g., Windows Server 2008 R2 or Windows 7)
- SharePoint Foundation 2010 and/or SharePoint Server 2010
- SharePoint Designer 2010
- Microsoft Office (Professional Plus) 2010
- Visual Studio 2010
- .NET Framework 4.0
- Microsoft Expression Blend (optional but recommended for Silverlight programming)
- SQL Server (Express) 2008

Having the preceding components will enable you to follow along with the coding examples throughout this book, and these are the baseline requirements to get up and running for SharePoint 2010 development in your organization. Note that in some cases you will also need to install (or sign up for) other software in each of the chapters to complete the exercises.



All the aforementioned software is preloaded onto the IW VM that you can download free and use for the 180-day trial period.

Let's first talk a bit more about setting up the virtualized environment and then discuss how you can set up a native installation on your local machine.

Setting Up a Virtualized Environment

In Windows 2008 R2 (64-bit), you can use the Hyper-V technologies to manage your VMs. The environment is a role that you set up when configuring your Windows operating system. For

example, after you install Windows Server 2008 R2, you can add the Hyper-V role through the Server Manager. Figure 1-2 shows an example of the Add Roles Wizard at the Server Roles step in the wizard. When you invoke the wizard, you can click the checkbox beside the Hyper-V role and Windows will install it for you. Note that I've already added the Hyper-V role to my machine.



FIGURE 1-2

Installing Hyper-V

Assuming you already have your Windows operating system in place, use the following steps to install Hyper-V.

1. Click Start ⇨ Administrative Tools ⇨ Server Manager.
2. In the Server Manager, scroll to the Roles Summary and then click Add Roles and select Hyper-V from the list.
3. Server Manager takes you through a number of steps. Accept the default options and click Next until the Install button is enabled.
4. Click Install to complete the Hyper-V installation. Note that Windows will prompt you for a system restart. Restart your computer to complete the Hyper-V installation.
5. After you have Hyper-V installed, you can then add a Hyper-V-compliant .vhd file if your team has already prepared one, or you can go about creating one from scratch.

6. To add an existing image, open the Hyper-V snap-in by clicking Start ⇨ Administrative Tools ⇨ Hyper-V Manager.
7. Under Actions, click New ⇨ Virtual Machine, and then specify a Name and Location for the image and click Next.
8. You'll then need to assign a level of RAM to the image — specify 6,500 MB or more.
9. Accept the default option for Configure Networking and click Next.
10. Click the radio button beside Use an Existing Hard Disk, and then browse to that disk and click Finish.

Setting Up a Native Environment

If you want to install the prerequisite software on your local machine, you can do that as well. There are a couple of ways to do this. The first way is to get all the software mentioned earlier and install that on your local machine. The second way is to use the SharePoint 2010 Easy Setup Script, which installs trial versions of all the prerequisite software onto your local machine. You can download the script from www.microsoft.com/download/en/details.aspx?id=23415. Once you download the script, you can start it and go get a coffee (or two), as it takes a while to install everything on your machine. It is, however, a streamlined way to install all the software in the proper sequence, and it saves you the effort of gathering all the software yourself.



Chris Johnson provides a good blog post on how to use the Easy Setup Script here: <http://blogs.msdn.com/b/cjohnson/archive/2010/10/28/announcing-sharepoint-easy-setup-for-developers.aspx>.

If you choose to install the software manually, you can install a standalone SharePoint 2010 instance to use as a trial environment, as shown in the following steps. (Because you'll be creating small solutions in this book, you won't need to set up a multi-server farm; this is more for a production environment.)

1. Click the `Setup.exe` file on your SharePoint 2010 installation DVD (or from your installation location).
2. You'll be prompted to Agree to the License Terms. Click the "I accept the terms of this agreement" checkbox, and then click Continue.
3. Next, you have the option to select different installation options. Click the Standalone button to invoke the Standalone installation. SharePoint will then begin the installation process (which includes a software installation process and a configuration process).
4. When the initial software installation process has completed, you will be prompted with a dialog where you can choose to run the configuration wizard, which configures things like

the services, content database, and so on for first-time use (see Figure 1-3). Click the “Run the SharePoint Products and Technologies Configuration Wizard now” checkbox, and then click Close. The configuration wizard will be automatically invoked upon closing this dialog.



FIGURE 1-3

SharePoint works through a series of ten configuration steps to complete the configuration process.

5. When the configuration process has completed, you will be prompted with a Configuration Successful dialog. Click Finish.
6. SharePoint should automatically prompt you with the standalone SharePoint instance you created and, upon first opening, ask you to select a type of site and have you set permissions for the site. Explore the different site templates that are available, but choose Team Site and click OK, and then verify and set the security options. Here you can leave the default options and click OK. Your site will then be created and will open at the default landing page (see Figure 1-4).

At this point, your SharePoint environment should be set up and ready to go. However, because you're developing for the cloud, you'll find that you often need to obtain a developer key or create an account to begin using a specific cloud technology. For example, you will need to create a LinkedIn

or Twitter account, obtain a Bing Maps developer key, create a Windows Azure account, or create a Dynamics CRM account. As mentioned earlier, subsequent chapters provide the information you need to sign up for each of these as a part of the exercises.

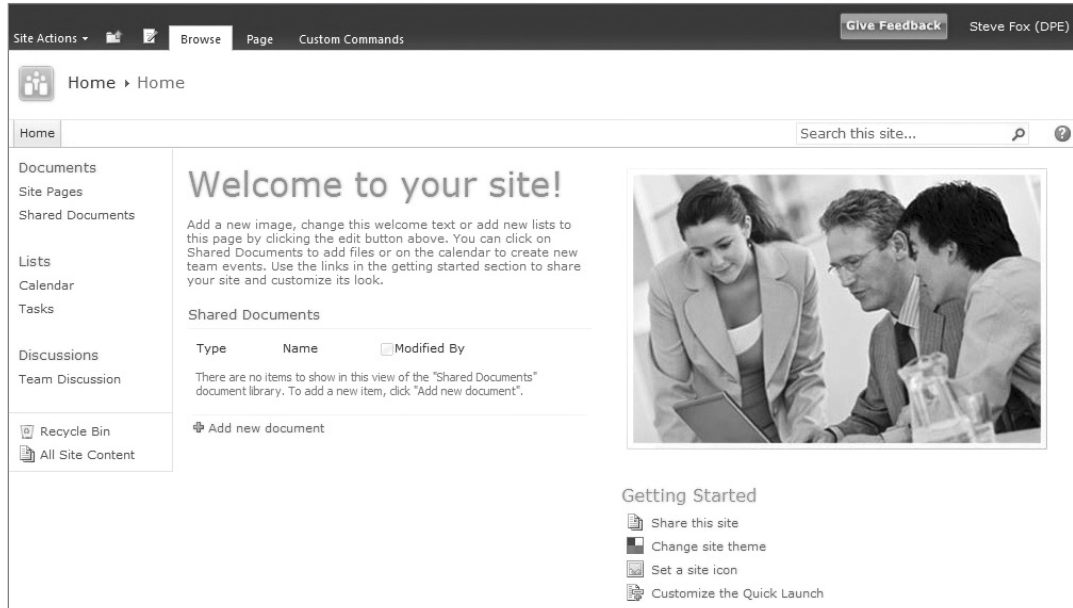


FIGURE 1-4

CREATING YOUR FIRST SHAREPOINT CLOUD APPLICATION

What better way to get your feet wet than to build a cloud-based application in the first chapter! Don't worry, we won't throw you into the deep end; this will be a simple application (we'll call it Map Me!) that shows how to integrate SharePoint and Bing Maps.

Before you start this exercise, you'll need to get a developer account key — a simple and quick step. First, sign into the Bing Maps portal, at www.bingmapsportal.com. As shown in Figure 1-5, you can sign in using your Live ID as an existing user (if you already have a Bing account) or click Create to create a new developer account.

If you click Create, you'll sign in with your Live ID and then walk through a wizard to add an account name and email address, and then you'll be asked to agree to the Bing Maps agreement. Click Save to save your account details. Once your account is provisioned, you need to obtain a developer key for your application. In the main portal, click "Create or view keys." You can create a key by entering an application name and selecting a specific type of application (see Figure 1-6).

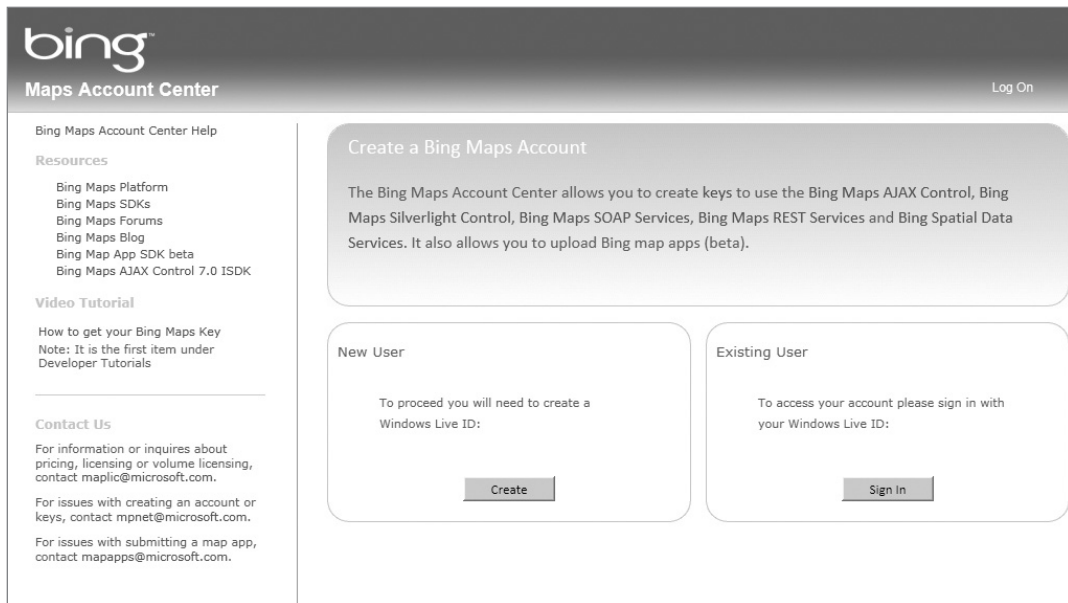


FIGURE 1-5

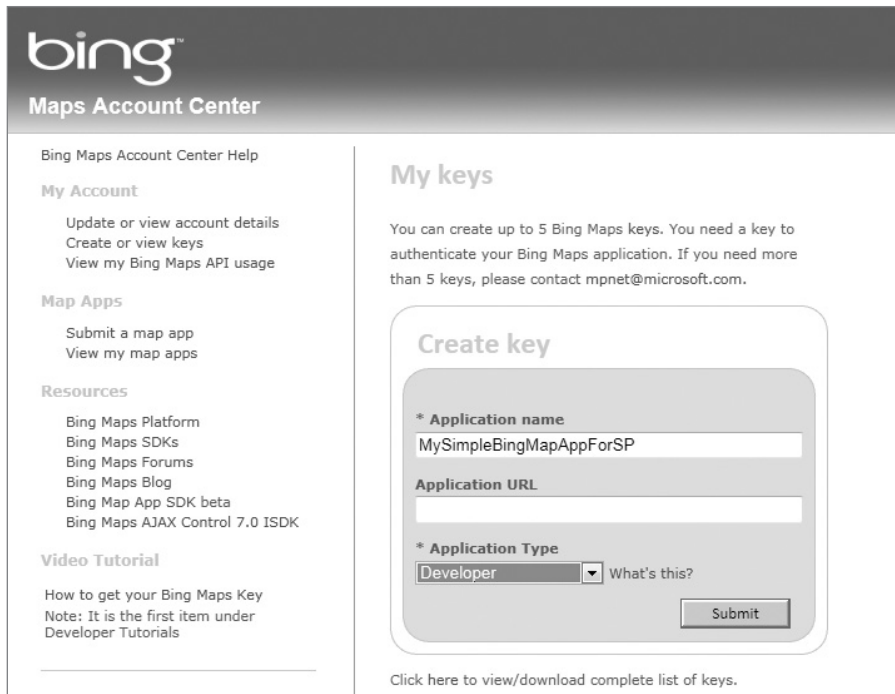


FIGURE 1-6

After you click Submit, you'll be provided with a developer key that you'll use in your Bing Maps application.

With your developer key in hand, complete the following steps to create your first cloud application for SharePoint:

1. Open Visual Studio 2010 and click File ⇨ New Project.
2. Under Installed Templates, select Other Project Types ⇨ Visual Studio Solutions ⇨ Blank Solution.
3. Provide a name for your project (e.g., **MyFirstCloudApplication**) and then click OK.
4. After Visual Studio creates the solution, right-click the new project and select Add ⇨ New Project.
5. In the Add New Project template, select Silverlight Application. Provide a name for the application (e.g., **SimpleBingMap**) and click OK.
6. When prompted, check the “Host the Silverlight application in a new or existing Web site in the solution” checkbox, and click OK.
7. Right-click the Silverlight application and select Add Reference. In the Add Reference dialog, click Browse.
8. Browse to the `Microsoft.Maps.MapControl.dll` library on your local machine, select it, and click OK.
9. Right-click the `MainPage.xaml` file and select View Designer.
10. Add the following bolded code to the XAML view of your Silverlight application:

```
<UserControl x:Class="SimpleBingMap.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="clr-namespace:Microsoft.Maps.MapControl;assembly=Microsoft.Maps.MapControl"
    mc:Ignorable="d"
    d:DesignHeight="414" d:DesignWidth="888">

    <Grid x:Name="LayoutRoot" Background="White" HorizontalAlignment="Stretch"
        VerticalAlignment="Stretch" Height="394" Width="888">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="888*" />
            <ColumnDefinition Width="314*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <StackPanel Grid.ColumnSpan="2">
            <TextBlock Text="Map Me!" FontWeight="Bold" FontSize="14" Width="888" />
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Stretch"
                Margin="12,0,12,0">
                <TextBox x:Name="txtbxLatitude"
                    Width="322"
                    Text="Type your latitude here, e.g. 47.7656"
                    FontSize="10"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Stretch"
```

```

        Margin="12,0,12,0">
            <TextBox x:Name="txtbxLongitude"
                Width="322"
                Text="Type your longitude here, e.g. -122.9957 "
                FontSize="10" />
        </StackPanel>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Stretch"
            Margin="12,0,12,0">
            <Button x:Name="btnFindMe" Content="Map It!" Width="65"
                Click="btnFindMe_Click" />
        </StackPanel>
    </StackPanel>
    <m:Map CredentialsProvider="Your Developer Key Here."
        x:Name="MyMap" Grid.Row="1" Mode="AerialWithLabels"
        HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
        Margin="0,14,0,0"
        Grid.ColumnSpan="2">
        <m:Map.Children>
            <m:MapLayer x:Name="PushPinLayer" />
        </m:Map.Children>
    </m:Map>
</Grid>
</UserControl>

```

Your application should now look something like Figure 1-7.

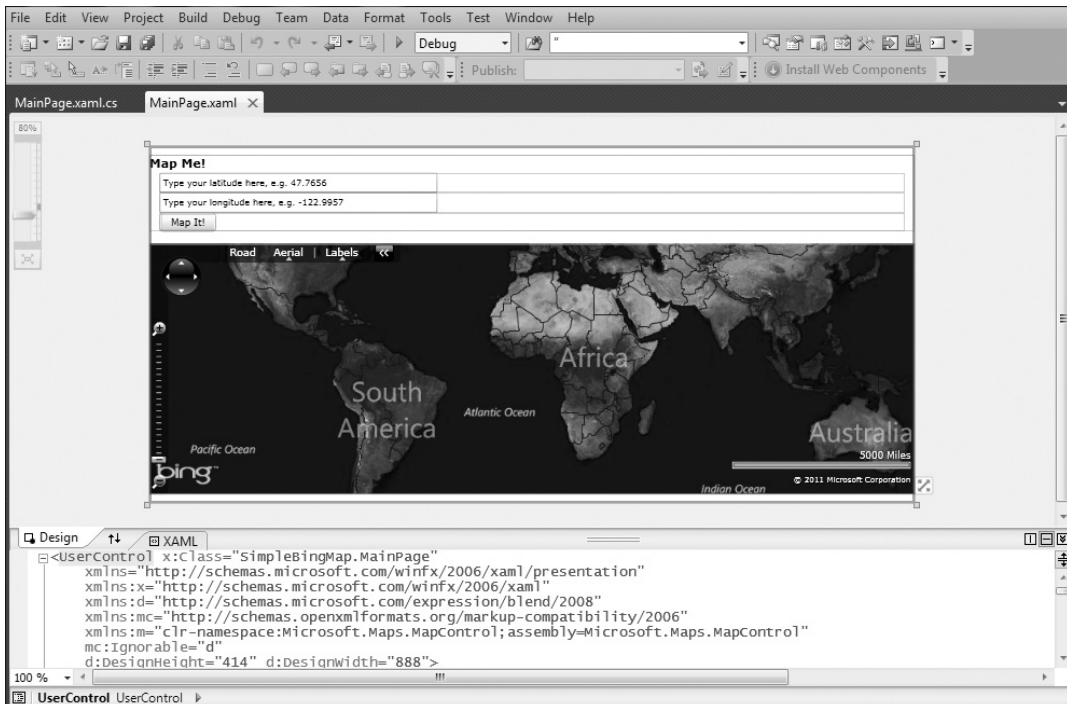


FIGURE 1-7

11. Double-click the button (btnFindMe) and add the following bolded code to the MainPage.xaml.cs file:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Maps.MapControl;

namespace SimpleBingMap
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void btnFindMe_Click(object sender, RoutedEventArgs e)
        {
            //Create a new location comprising latitude and longitude.
            Location myLocation = new Location();
            myLocation.Latitude = Double.Parse(txtbxLatitude.Text);
            myLocation.Longitude = Double.Parse(txtbxLongitude.Text);

            //Create a new pushpin to add to the map.
            Pushpin myPPin = new Pushpin();
            myPPin.Width = 7;
            myPPin.Height = 10;
            myPPin.Location = myLocation;
            PushPinLayer.AddChild(myPPin, myLocation, PositionOrigin.Center);

            //Set the main view of the map using the location with a zoom level.
            MyMap.SetView(myLocation, 10);
        }
    }
}
```

code snippet 076576 Ch01_Code.zip/SimpleBingMap.cs

The code here is fairly straightforward; you're using the Bing Maps API (Microsoft.Maps.MapControl) to create a Location object. The Location object has a Latitude property and Longitude property, which you set and then assign to the PushPin object. You also set some additional properties to the PushPin object, which is then added to the PushPinLayer using the AddChild method.

12. Click F5 to build and debug the application in your default browser.
13. You should now see something similar to Figure 1-8. Enter a latitude and longitude into the two text boxes, and then click the Map It! button.

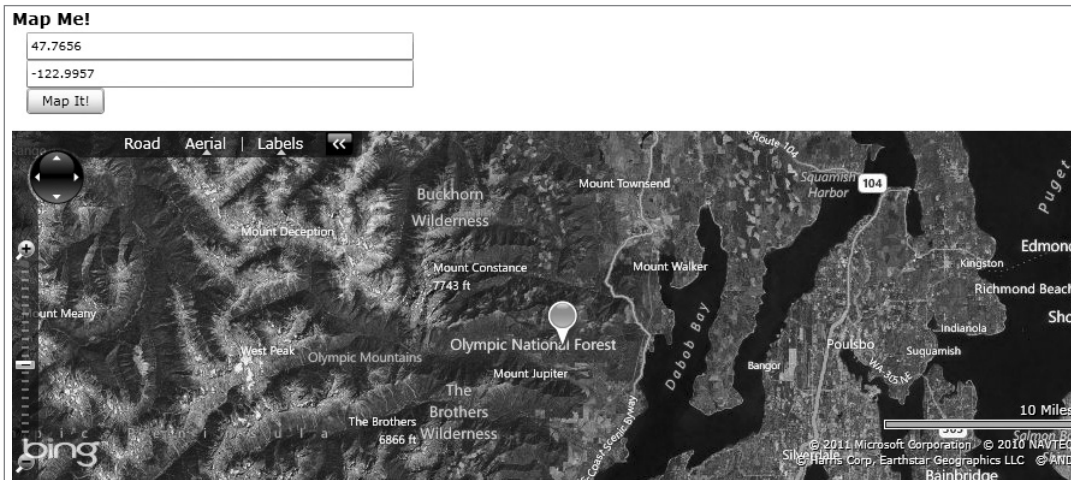


FIGURE 1-8

14. After you've successfully built the Silverlight application, right-click the project and select Open in Windows Explorer. Navigate to where the XAP file (e.g., SimpleBingMap.xap) is located and copy the folder path to the clipboard.
15. Open your SharePoint site and create a new document library called XAPS. Once it's created, click Add document, click the Browse button, and then paste the folder path into the Choose File to Upload dialog.
16. After the file has been uploaded into the new SharePoint document library, right-click the link to the XAP file and select Copy Shortcut.
17. Navigate to the top level of the SharePoint site that was created when you installed and configured SharePoint. Click Site Actions and Edit Page.
18. Click the Insert tab and then click the Web Part button in the ribbon. In the Categories, select Media and Content, and then click the Silverlight Web Part.
19. Click Add, and then paste the shortcut to the XAP file that you uploaded into the XAPS document library. You will need to resize the web part, which you can do by clicking Edit web part and changing the Width to 900 and the Height to 400.

Congratulations! You've created your first cloud application that integrates with SharePoint. When loaded properly, the Map Me! application should look something like Figure 1-9. Admittedly, this was more of a Hello World application, but it gives you some idea of just what a simple cloud

application can do. This application could absolutely be extended to interact with SharePoint lists, and you could tap into the extensive Bing Maps APIs (which you will do later in the book).

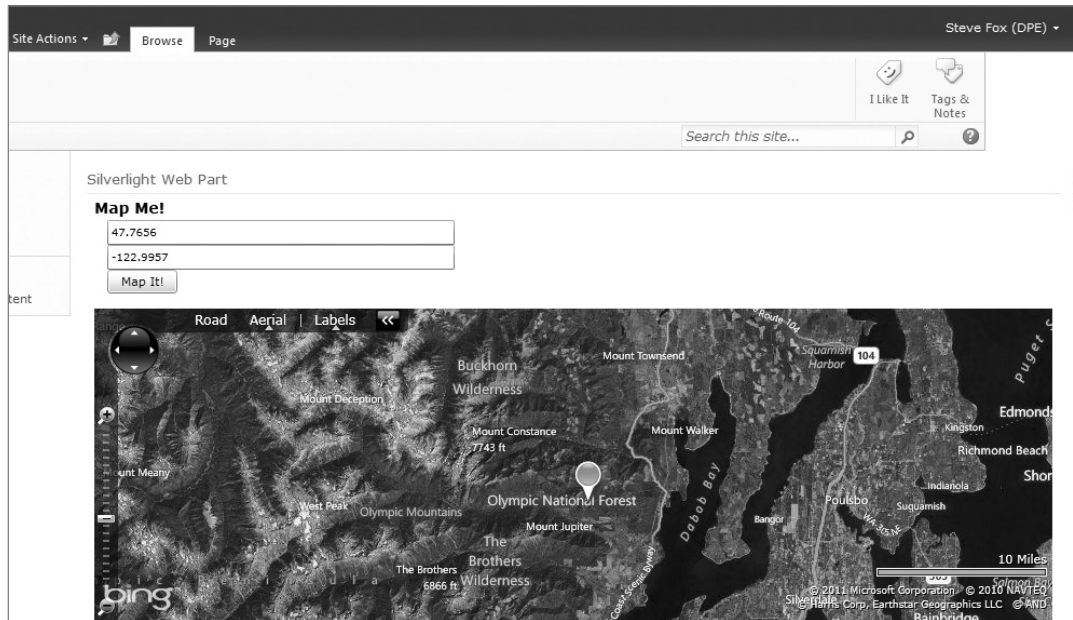


FIGURE 1-9

SUMMARY

This chapter introduced you to the core concept of the cloud, reinforcing the idea that the cloud represents a significant opportunity to engage in software development in a new and compelling way. Cloud computing is an oft-disputed concept, but for this book we've defined the cloud as a metaphor for the Web and that cloud computing represents using the Web as a connected set of resources to build and deploy your software. This chapter also introduced a number of different cloud technologies, such as Windows Azure, Web 2.0, Bing Maps, and Microsoft Dynamics CRM. These are the core cloud technologies that we'll focus on throughout the course of this book. In fact, each subsequent chapter will in some way use one of these technologies as a point of integration with SharePoint to build cloud-based applications.

The cloud is a novel and exciting place for application development, and one that's full of potential. This potential is compounded when you begin to look at what you can do with SharePoint, as the cloud is certain to play a very important role in SharePoint's future. This can be seen in the short term with the growing popularity of Office 365. We hope this book helps you get started with both developing for the cloud and integrating your cloud-based solutions with SharePoint.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- *Cloud Computing: A Practical Approach*. Velte, Velte, and Elsenpeter (McGraw Hill, 2010).
- **Windows Azure** — www.microsoft.com/windowsazure
- **Windows Azure Free Trial Offer** — www.microsoft.com/windowsazure/free-trial/sharepoint-integration/
- **Bing Maps Developer Center** — www.microsoft.com/maps/developers/web.aspx
- **Overview of Microsoft Dynamics CRM** — <http://social.technet.microsoft.com/wiki/contents/articles/microsoft-dynamics-crm-overview.aspx>
- **What Is Web 2.0** — <http://oreilly.com/web2/archive/what-is-web-20.html>

2

Using SQL Azure for Business Intelligence

WHAT'S IN THIS CHAPTER?

- Understanding the basics of SQL Azure
- Understanding the different types of business intelligence solutions you can build using SharePoint and SQL Azure
- Building a business intelligence solution using SQL Azure

The goal of this chapter is to introduce you to SQL Azure and describe how you can build a business intelligence (BI) solution using SQL Azure and SharePoint 2010 (or just SharePoint in this chapter). Similar to the other chapters in this book, this chapter first provides some grounding, then walks through a high-level solution architecture, and then describes the practical how-to guidance to create the solution. Because it is possible to build both no-code and code-based solutions using SQL Azure and SharePoint, the solution incorporates both of these into the final solution.

OVERVIEW OF SQL AZURE

SQL Azure is one of the core parts of Windows Azure and represents Microsoft's transactional database offering in the cloud. It is also a highly available cloud-based relational database service that is architected similar to SQL Server. Similar to Windows Azure, SQL Azure represents a pay-as-you-go service that scales out with your needs. If you've used SQL Server before, you'll easily transition into using SQL Azure. Because SQL Azure is hosted in a data center, you don't need to do anything to manage it (e.g., upgrade, configure, patch, and so on). Also, because SQL

Azure is scalable and elastic, it grows as you use it; you don't need to worry about running into storage-capacity issues, as you would need to manage within an on-premises data center or lab.

SQL Azure supports many of the same features as SQL Server. For example, you can manage tables, databases and primary keys; issue stored procedures; create views; query data; and much more. SQL Azure also uses a set of familiar tools. As you'll see in this chapter, you primarily use SQL Server 2008 R2 Management Studio to manage your SQL Azure database. Many of the same functions that you use with on-premises databases are available to you with SQL Azure; however, some menu options may not be available in the current release (which at the time of writing was the May 2011 release). You can use Transact-SQL (T-SQL) scripts in situations where menu options don't exist to issue commands against your SQL Azure database, such as creating databases and tables and populating tables with data.

In this chapter, you'll use SQL Azure as your cloud-based data storage engine, and you'll store fictional sales data in it. The goal of the chapter is to surface the sales data within a no-code BI application and a code-based BI application, which together comprise the chapter's BI solution.

Uses of SQL Azure

SQL Azure is your transactional and relational database in the cloud, and, as such, there exist many uses for it. For example, you can store different types of data in SQL Azure and then use this data as a back-end to websites, web applications, and web services. You can also use SQL Azure to mirror your on-premises data so that you have a replicated cloud instance of your database available to remote users or partner extranets. SQL Azure also serves as a temporary storage facility to manage transactional records that don't require long-term storage.

How you manage data in SQL Azure is very similar to how you would interact with data in other relational databases. For example, you can create, access, and manage tables, views, roles, stored procedures, and functions using SQL Azure. You can also issue queries and then join together multiple tables. Furthermore, you can insert, update, and delete data from SQL Azure, as well as perform many other core relational database functions.

As a developer, you can also program against SQL Azure in a number of different ways. For example, you can use Open Database Connectivity (ODBC) to open communication with SQL Azure and create simple SQL queries or leverage stored procedures in code. In the "Code-Based Solutions in SharePoint" section later in this chapter, you'll learn how you can programmatically interact with SQL Azure in greater detail, and throughout the chapter you'll get hands-on coding experience that will enable you to put this programming into practice.

Migrating SQL Server to SQL Azure

In this chapter, you'll create a small database and table and then populate that table with some fictional sales data. However, because you use SQL Azure as your data source, you don't necessarily need to start from scratch; you *can* migrate data from your on-premises instance of SQL Server to SQL Azure. Furthermore, you have the option to use either the native tools and scripts that are built into SQL Server Management Studio 2008 R2 or one of a growing set of migration tools.

One way to migrate your data is to export the database schema of your on-premises SQL Server database and then transfer the data to SQL Azure. This requires exporting a script from SQL Server

Management Studio and configuring your export options, which then creates the schema in the SQL Azure database. You can also use the Generate and Publish Scripts Wizard to transfer a database from your local machine to a SQL Azure database. This wizard creates T-SQL scripts for your local database, which then enables you to move the data to SQL Azure. For more information, visit www.msdn.microsoft.com/en-us/library/ee621790.aspx.

Another way to migrate your data from SQL Server to SQL Azure is to use the Microsoft Sync Framework 2.1, which supports synchronization between on-premises and SQL Azure servers. In essence, Microsoft Sync Framework 2.1 enables you to extend the schema of your on-premises database to the cloud. Microsoft Sync Framework 2.1 comes with a great set of documentation and an SDK, which can be found at <http://msdn.microsoft.com/en-us/sync/default.aspx>.

The SQL Azure Migration Wizard, a CodePlex project, also supports migrating data to SQL Azure by enabling you to select specific SQL objects, create SQL scripts that are conversant with SQL Azure, and move data from on-premises to the cloud. You can also move data across SQL Azure databases. For more information, see <http://sqlazuremw.codeplex.com/>.

Finally, you can also use the SQL Server Integration Services (SSIS) to move data in and out of SQL Azure. Using SQL Server 2008 R2, the Import and Export Data Wizard provides support for migrating data into SQL Azure. For more information, visit <http://msdn.microsoft.com/en-us/library/ms141026.aspx>.



A great resource is Roger Doherty, a longtime SQL Server evangelist/technologist. You can find his blog at <http://blogs.msdn.com/b/rdoherty/>.

Interacting with SQL Azure with SQL Server 2008 R2 Management Studio

SQL Server 2008 R2 Management Studio is one of the main ways in which you can interact with SQL Server data. It is an integrated environment that enables you to log in to your SQL Azure database and configure, manage, administer, and develop against your data. In essence, it represents your GUI interface with SQL Azure. When you launch Management Studio, you'll need to select the server type (e.g., Database Engine), enter your server name (e.g., server-name.database.windows.net), select the authentication type (e.g., SQL Server Authentication), provide a login username (e.g., johndoe), and, finally, your password. Figure 2-1 illustrates the Connect to Server dialog that connects you to your SQL Azure instance.



FIGURE 2-1

After logging into SQL Azure, you can use many of the core SQL Server Management Studio functions to interact with your data. This functionality includes creating queries against your data (e.g., to create tables, insert data, query and display data, etc.). Management Studio also provides right-click functionality in the 2008 R2 version, enabling you to use shortcuts to common functions. For example, Figure 2-2 illustrates the shortcut menu that is available for a table. As you can see, you can dynamically create generic T-SQL scripts that perform different functions, ranging from CREATE to SELECT.

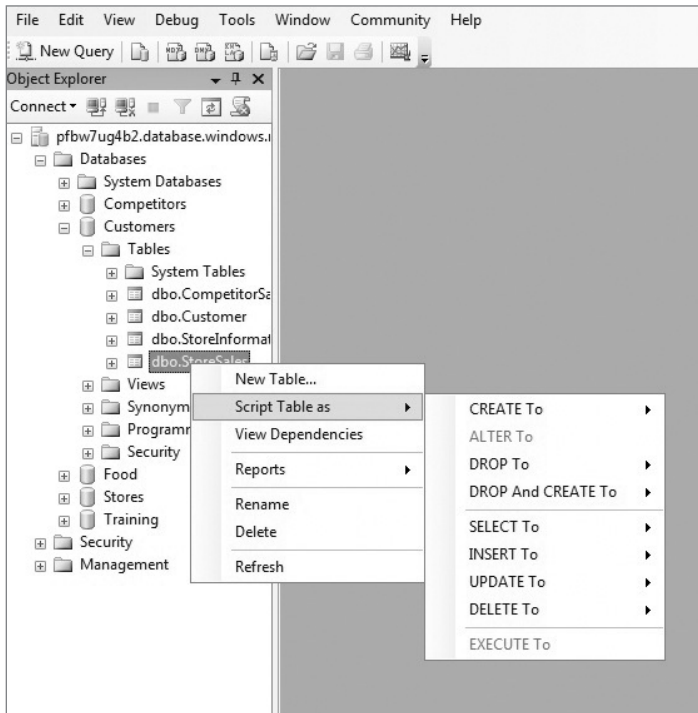


FIGURE 2-2

Querying data in SQL Azure using the SQL Server 2008 R2 Management Studio GUI is slightly different from its SQL Server counterpart. For example, when you right-click the table and select Script Table as ⇒ SELECT To ⇒ New Query Editor Window, the result is a generic `SELECT *` statement that queries your data (see Figure 2-3). Thus, the shortcut menus offer an easy way to run some default T-SQL script against your database in the cloud.

As you explore SQL Server 2008 R2 Management Studio to manage your SQL Azure data, you'll surely discover many ways to interact with your cloud-based data.

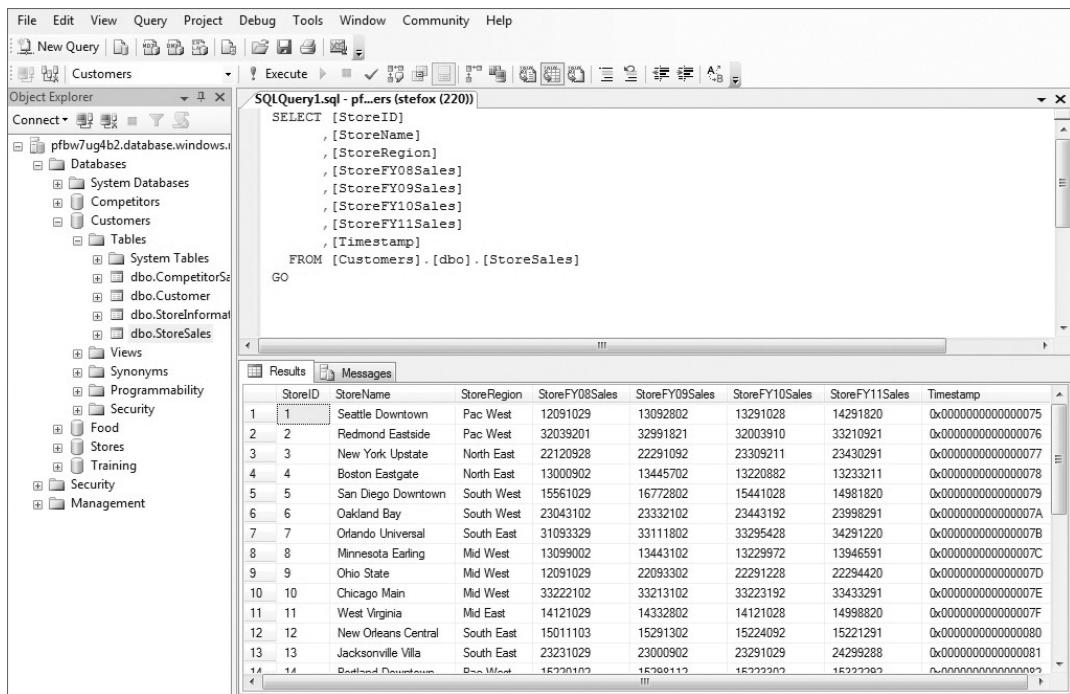


FIGURE 2-3

BI SOLUTIONS USING SQL AZURE

SQL Azure is your relational database in the cloud; it represents a starting point for managing data for your organization. However, managing relational data is but one aspect of a BI solution. A second and equally critical aspect of the BI solution is presenting that data in such a way that it is meaningful and relevant (and perhaps filterable and manageable). Accomplishing this requires not only the careful storage of data, but also the capability to render and manipulate the data on the client in a variety of ways. SharePoint provides many ways for you to interact with SQL Azure — some of these are no-code solutions and others are code-centric solutions.

No-Code BI Solutions in SharePoint

No-code BI solutions in SharePoint provide a range of integrated capabilities that can help you manage your SQL Azure data. Some of these solutions are built into the native SharePoint capabilities, while others require some configuration or declarative development to create and deploy.

Out-of-the-box solutions for SharePoint vary. For example, one possibility is the Chart web part, which enables you to connect to data and expose that data in different ways and types of charts

in a web part. The types of data you can connect to range from SQL Server data to SharePoint list data. The Chart web part uses the `System.Web.DataVisualization` library (the ASP.NET Chart control) to create an in-browser configuration experience to create some simple, but effective, charting. Another possibility are the Key Performance Indicators (KPIs), which enable you to set status indicators using a range of data types, ranging from SharePoint list data to Microsoft Excel data to SQL Server data. Finally, Excel Services enables you to bridge SQL Azure data with Excel and then populate Excel Web Access web parts with that cloud data.

Another example of a no-code solution is the use of SQL Server Reporting Services (SSRS) to build form-based views of SQL Azure data in SharePoint. Some configuration is required for SSRS, but after you get the SSRS bits installed and configured, you can create many different reports for your SQL Azure data.

Another no-code example, and one that is more commonly used with external data systems, is using Business Connectivity Services (BCS). BCS provides you with the capability to integrate directly with external systems — both ADO.NET-based and web-service-based systems. This enables you to connect to SQL Azure natively using SharePoint Designer 2010 to declaratively build what are called *external content types* (XML files that define the relationship between SharePoint and the external data system).

Code-Based Solutions in SharePoint

When you build a code-based solution, first you need to get the data from SQL Azure, and then you need to do something with it within SharePoint. If you've done any development before, you know that data-driven programming can take many different forms. For example, you can use ODBC or ADO.NET to interact with your database. This can take the form of SQL query strings (e.g., to connect and query a database), or it can take the form of WCF Data Services (formerly ADO.NET Data Services), which provides a richer form of data-binding and querying. For example, you can leverage the WCF Data Services to create an entity data model of the SQL Azure data construct. This approach enables you to easily query the data from SQL Azure using LINQ. When using the WCF Data Services, you should be aware that not all .NET client technologies supported in SharePoint support WCF Data Services. For example, if you want to use Silverlight, you may want to either choose a different data connectivity method or abstract the call to SQL Azure using a WCF service proxy (which does support entity data models).

For more query-string-driven applications, you can also use the `SqlDataAdapter` class to build and execute SQL connection strings and queries against your SQL Azure database. Although you can do this, you must generate the query strings in your code, which can sometimes be cumbersome with elaborate queries. Using the `SqlDataAdapter` class provides a more universal approach to your data connections; that is, it's widely supported. This mitigates the need for WCF service proxies, although it doesn't completely eliminate the possibility of using them. For more information on the `SqlDataAdapter` class, visit <http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx>.

Of course, after you are able to connect and query the data in SQL Azure, you must do something with the data when you retrieve it. This is where you dig a little deeper into SharePoint. Using SharePoint, you can, for example, leverage Silverlight, web parts, BCS (e.g., external lists), ASP.NET charting, and so on, to surface data from SQL Azure. Each artifact supported by SharePoint

provides slightly different results, though. For example, Silverlight is very handy when it comes to building BI solutions, given the smooth and dynamic UI experience that can be easily built and deployed. Examples of this are theming and the chart toolkit for Silverlight, which provide simple design augmentation and different graphical charting capabilities. Web parts, one of the most commonly developed artifacts within SharePoint, also serve as a great way to display data. You can use web parts to create filterable and queryable views to SQL Azure data. BCS provides a native connection to SQL Azure, which enables end users to manage data into and out of SQL Azure, while also allowing you to programmatically connect to and leverage the SQL Azure data in the external list through the SharePoint client object model. The ASP.NET Chart control provides a rich and programmatic way to interact with SQL Azure.

In this chapter, you'll use a combination of the preceding approaches — except for Silverlight, which is discussed in Chapter 7 — to build your BI solution. Specifically, you'll first build a no-code solution using BCS to create an external list that provides the native connection to SQL Azure. Then, you'll also build a code-based web part that uses the ASP.NET Chart control programmatically to aggregate and display data from SQL Azure. Although the application you'll build could have used the server object model or even the client object model to interact with the external list data (i.e., to query it), we think it's more useful to walk through abstracting the data connection and query layer with a WCF service proxy. This way, you can repurpose the WCF service across multiple applications. As you work beyond this chapter, though, keep in mind that the SharePoint object model is also a possibility within this solution architecture. Similarly, rather than use, say, the `DataSet` data construct, you'll use a custom class to manage and bind data within the web part.

CREATING A BI DASHBOARD USING SQL AZURE AND SHAREPOINT

Dashboards provide a quick-and-easy way to scan and understand how specific metrics are performing. For example, sales professionals and executives often track the sales of products through sales dashboards, which, as you can imagine, provide sales figures across time, potentially regions, for specific products. Dashboards can be created in any number of ways, many of which have been mentioned already in this book. The BI dashboard you'll create in this chapter will integrate an external list with an ASP.NET Chart control within a web part.

The Solution Architecture

The design of the solution you'll build in this chapter is straightforward. It uses SQL Azure as the back-end data source and incorporates a no-code *and* code-centric approach to building the two core BI applications that will make up the solution. The no-code approach leverages BCS to integrate SQL Azure with SharePoint, and the code-centric approach leverages the ASP.NET Chart control to build a custom connection to the SQL Azure data. The use of BCS enables you to connect to SQL Azure directly, and the ASP.NET Chart control leverages a WCF service proxy that queries the SQL Azure data and dynamically generates a chart in SharePoint.

Figure 2-4 provides a high-level overview of the solution architecture. In the diagram, the ASP.NET Chart control consumes a WCF service, which further queries the SQL Azure database. Note that the WCF service is deployed to the SharePoint server; however, you can also deploy the service to

Windows Azure. Deploying the service to Windows Azure would enable you to leverage the WCF service in other applications and platforms such as SharePoint Online/Office 365 (e.g., developing a Silverlight-based chart application using the same data in a SharePoint Online application). This is indicated by the parts of the diagram marked with dotted lines. The core web method within the WCF service is the `GetAllSalesData` method, which will do the heavy lifting to retrieve the sales data from the SQL Azure database. The BCS is built directly against the SQL Azure data source, which provides a read-write external list for you to manage the sales data.

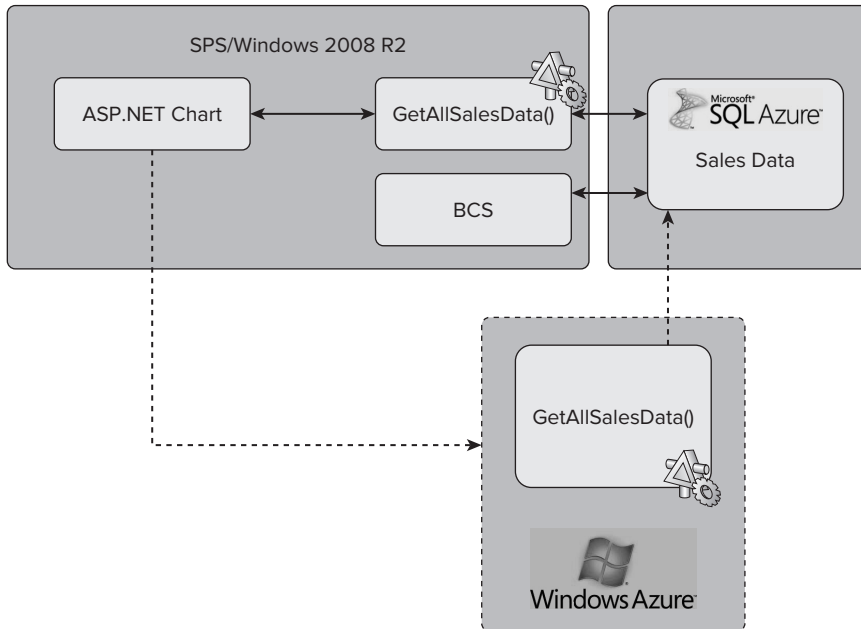


FIGURE 2-4

As you work through the example, you'll notice that the bridge between the SQL Azure database and the external list is managed through an application ID, which helps manage two sets of credentials. Because SQL Azure has a different set of credentials from Windows credentials, you cannot pass the active user's credentials across; you must provide a way to authenticate the call to SQL Azure.

The result of this architecture is manifested in two parts to the BI solution: an external list that provides read/write access to the SQL Azure database, and an ASP.NET Chart web part that provides an aggregate sales view of the data.

Creating the Sales BI Application

The first step in building the BI application is to create the sales data in SQL Azure. To complete this step, you must have a Windows Azure account provisioned. To get a trial account, visit www.microsoft.com/windowsazure/free-trial/sharepoint-integration/.

Creating the SQL Azure Database

You need to create the SQL Azure database to store your sales data in the cloud.

1. Navigate to <http://windows.azure.com> and sign in using your Live ID.
2. In the left navigation pane, click Hosted Services, Storage Accounts & CDN, and then select New Storage Account.
3. Map the new storage account to your subscription and provide a namespace for the storage account. You'll also need to select a region. When done, click Create.
4. Click the Firewall Rules control to set the firewall rule for the storage account (see Figure 2-5).

This rule enables your machine to connect and interact with the storage account. In a production environment, you should always ensure that this is set to a strictly defined IP, but for a proof of concept or demo you can set the IP rule to be more open. Provide a name for the firewall rule (e.g., MyServerFWRule), and then select the IP range (e.g., 0.0.0.0-255.255.255.255).

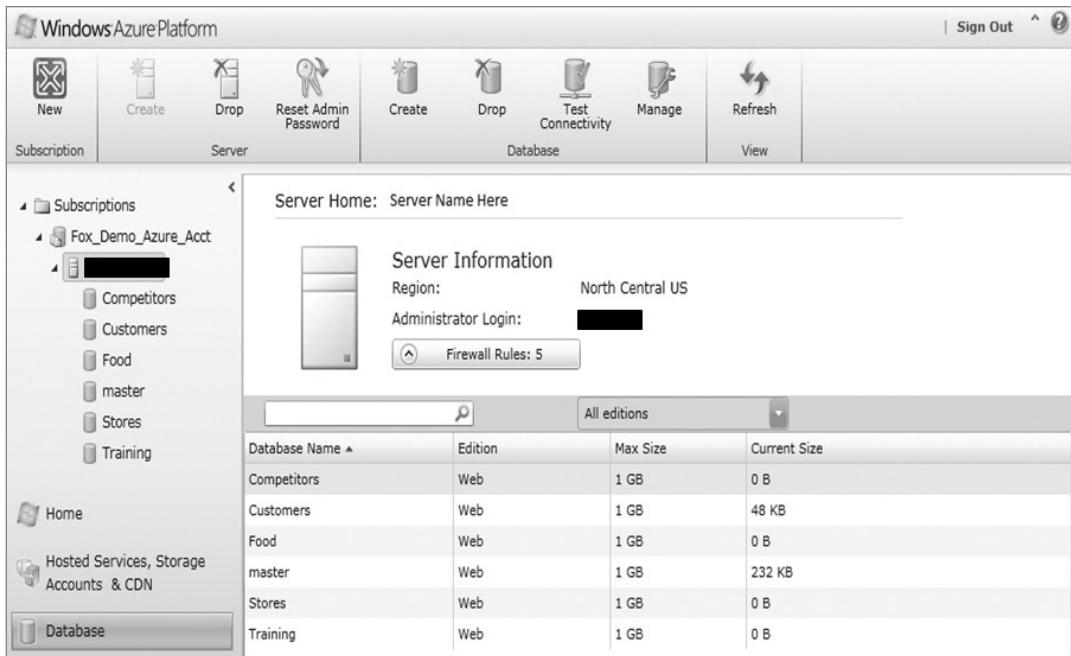


FIGURE 2-5

5. Click the Database tab in your portal, and then select the subscription where you want to create your Sales database.

6. Create a new database by clicking Create. Provide a name (e.g., TotalSales) and select the edition (select Web, which is the scaled-down database for smaller, more low-cost usage). Make a note of the administrator username and password.
7. Open SQL Server Management Studio 2008 R2 and log into your new SQL Azure instance by providing the username and password credentials. When done, click Connect.
8. When connected, click the New Query button.
9. Type the following query into the Query window, and then click Execute Query. This will create a new table in your TotalSales database called StoreSales.

```
CREATE TABLE [StoreSales](
    [StoreID] [int] IDENTITY(1,1)NOT NULL PRIMARY KEY CLUSTERED,
    [StoreName] [nvarchar](50)NULL,
    [StoreRegion] [nvarchar](50)NOT NULL,
    [StoreFY08Sales] [nvarchar](50)NOT NULL,
    [StoreFY09Sales] [nvarchar](50)NULL,
    [StoreFY10Sales] [nvarchar](30)NULL,
    [StoreFY11Sales] [nvarchar](30)NULL,
    [Timestamp] [timestamp] NOT NULL
)
```

Now that you have a StoreSales table, you'll want to populate that table with some data. To do this, you'll create another SQL script to add several records to it.

10. Click the New Query button and type the following into the Query window. This will create a set of new records in your StoreSales table.

```
INSERT INTO [StoreSales]

([StoreName],[StoreRegion],[StoreFY08Sales],[StoreFY09Sales],[StoreFY10Sales],[StoreFY11Sales])

VALUES

('Seattle Downtown', 'Pac West','12091029','13092802','13291028','14291820'),
('Redmond Eastside','Pac West', '32039201','32991821','32003910','33210921'),
('New York Upstate','North East','22120928','22291092','23309211','23430291'),
('Boston Eastgate','North East', '13000902','13445702','13220882','13233211'),
('San Diego Downtown', 'South West','15561029','16772802','15441028','14981820'),
('Oakland Bay','South West', '23043102','23332102','23443192','23998291'),
('Orlando Universal', 'South East','31093329','33111802','33295428','34291220'),
('Minnesota Earling','Mid West', '13099002','13443102','13229972','13946591'),
('Ohio State', 'Mid West','12091029','22093302','22291228','22294420'),
('Chicago Main','Mid West', '33222102','33213102','33223192','33433291'),
('West Virginia', 'Mid East','14121029','14332802','14121028','14998820'),
('New Orleans Central','South East', '15011103','15291302','15224092','15221291'),
('Jacksonville Villa', 'South East','23231029','23000902','23291029','24299288'),
('Portland Downtown','Pac West', '15220102','15298112','15223302','15332292'),
('San Francisco Bay', 'South West','12091029','13092802','13291028','14291820'),
('Bellingham North','Pac West', '10112102','1022202','10220121','10993311'),
('Houston Main', 'Central','12091029','13033202','13112128','14432820'),
('Vancouver South WA','Pac West', '16654102','16755102','16344192','16332291'),
```

```
( 'Kansas City Downtown', 'Mid West', '22392012', '22302910', '23302903', '23403920' ),
( 'Los Angeles West', 'South West', '12998098', '12899878', '12998190', '13201920' ),
( 'Redwood South', 'South West', '20029102', '20123321', '21221092', '21221998' ),
( 'Georgetown Central', 'Central', '31029901', '32910928', '33321112', '34302910' ),
( 'Washington DC', 'Central', '13022212', '13118102', '13998192', '13443291' ),
( 'Madison South', 'Mid West', '14191229', '14432802', '14039028', '14221820' ),
( 'Grand Forks', 'Mid West', '22109102', '22009102', '23100192', '21108291' ),
( 'Tusla South', 'Mid West', '44531029', '45443802', '45665028', '46654820' ),
( 'Dallas Downtown', 'Central', '19989002', '18998902', '19887992', '19129891' ),
( 'Bellevue Lincoln', 'Pac West', '12094454', '13009802', '13341028', '13111820' ),
( 'Detroit', 'Central', '18998902', '18556502', '18776792', '18334391' )
```

11. Right-click the StoreSales table and select Script Table as ⇌ SELECT to ⇌ New Query Editor Window to auto-generate a generic SQL SELECT statement (see Figure 2-6).

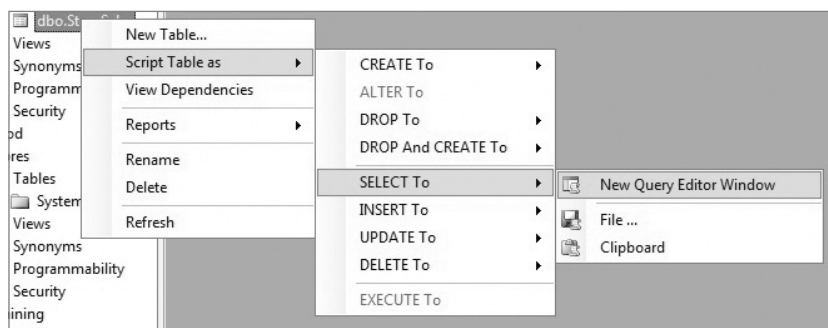


FIGURE 2-6

The result of this query will display all the data you just entered using the INSERT SQL script (see Figure 2-7).

Your SQL Azure database is now populated with some fictional sales data. In the following section, you'll create an external list that connects to the SQL Azure database (using BCS) and provides a read/write way to manage the sales data.

Creating an External List

An external list dynamically loads external data to enable users to create, read, update, and delete records in the external system.

1. You first need to create an application ID, which mediates the security handshake with SQL Azure. To create the application ID, open SharePoint Central Administration. Under Application Management, select Manage Service Applications.
2. Under Secure Store Service, click the Secure Store Service proxy link.
3. Click New in the ribbon to create a new application ID, which invokes a browser-based wizard comprising three steps.
4. On the Target Application Settings page, enter a target application ID (e.g., MyAppID), a display name (e.g., My App ID), and contact e-mail, and then click Next. (You can leave all other default options.)

SQLQuery1.sql - (...MOND\stefox (55))

/***** Script for SelectTopNRows command from SSMS *****/

```

SELECT TOP 1000 [StoreID]
, [StoreName]
, [StoreRegion]
, [StoreFY08Sales]
, [StoreFY09Sales]
, [StoreFY10Sales]
, [StoreFY11Sales]
, [Timestamp]
FROM [FabrikamSales].[dbo].[StoreSales]

```

Results Messages

	StoreID	StoreName	StoreRegion	StoreFY08Sales	StoreFY09Sales	StoreFY10Sales	StoreFY11Sales	Timestamp
1	1	Seattle Downtown	Pac West	12091029	13092802	13291028	14291820	0x000000000000007D1
2	2	Redmond Eastside	Pac West	32039201	32991821	32003910	33210921	0x000000000000007D2
3	3	New York Upstate	North East	22120928	22291092	23309211	23430291	0x000000000000007D3
4	4	Boston Eastgate	North East	13000902	13445702	13220882	13233211	0x000000000000007D4
5	5	San Diego Downtown	South West	15561029	16772802	15441028	14981820	0x000000000000007D5
6	6	Oakland Bay	South West	23043102	23332102	23443192	23998291	0x000000000000007D6
7	7	Orlando Universal	South East	31093329	33111802	33295428	34291220	0x000000000000007D7
8	8	Minnesota Earling	Mid West	13099002	13443102	13229972	13946591	0x000000000000007D8
9	9	Ohio State	Mid West	12091029	22093302	22291228	22294420	0x000000000000007D9
10	10	Chicago Main	Mid West	33222102	33213102	33223192	33433291	0x000000000000007DA
11	11	West Virginia	Mid East	14121029	14332802	14121028	14998820	0x000000000000007DB
12	12	New Orleans Central	South East	15011103	15291302	15224092	15221291	0x000000000000007DC
13	13	Jacksonville Villa	South East	23231029	23000902	23291029	24299288	0x000000000000007DD
14	14	Portland Downtown	Pac West	15220102	15298112	15223302	15332292	0x000000000000007DE
15	15	San Francisco Bay	South West	12091029	13092802	13291028	14291820	0x000000000000007DF
16	16	Bellingham North	Pac West	10112102	10222802	10220121	10882311	0x000000000000007E0

FIGURE 2-7

5. In the Field page, add a username and password. Select User Name and Password in the Field Type drop-down lists.
6. Enter a valid user from your SharePoint site (e.g., administrator) to be the application ID administrator. Click OK to complete the creation of the application ID. Your new application ID should now be listed in the application ID list.
7. Navigate to the root-level SharePoint site and select Site Actions ➤ Edit Site in SharePoint Designer.
8. Click External Content Types in the Navigation pane.
9. Click External Content Type.
10. Provide a name and display name (e.g., MySQLAzureECT), leave the default options, and then select “Click Here to Discover External Data Sources and Define Operations.”
11. Click Add Connection, select SQL Server from the drop-down list, and then click OK (see Figure 2-8).
12. When prompted, enter the name of the SQL Azure server and the database name, and then provide a display name for the external content type. Click the Connect with Impersonated

Custom Identity option and then enter the name of the application ID you created earlier (e.g., MyAppID) in the Secure Store Application ID field.

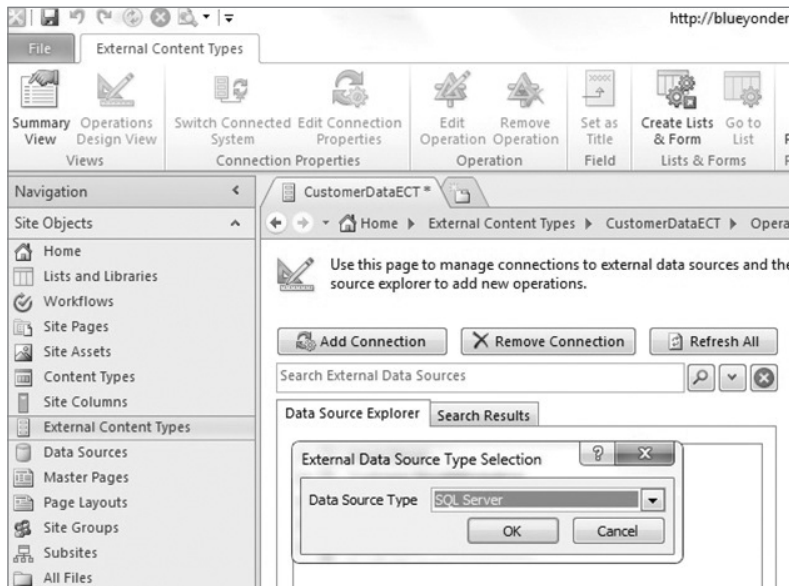


FIGURE 2-8

- 13.** Click OK to connect to SQL Azure. (Note that you will now be prompted for your SQL Azure credentials.)
- 14.** When SharePoint Designer loads the data connection, right-click the StoreSales table and select Create All Operations.
- 15.** To create the external content type, you must work through the wizard. Be sure to click the StoreID and then click Map To Identifier. You can then click Next twice or, optionally, click Finish.
- 16.** When done, click Save.
- 17.** You can now create a new list using the external content type by clicking Create Lists and Form. Add a name for the list in the List Name field, leave the other default options, and then click OK.
- 18.** Now that you've created the external content type that connects SharePoint to SQL Azure, you must set the permissions that allow specified users to access the list. To do this, navigate to SharePoint Central Administration and click Business Data Connectivity Services under Manage Service Applications.
- 19.** Click the external content type you just created and then click Set Object Permissions.

- 20.** In the dialog that appears, enter the user and permissions you want to assign to that person, as shown in Figure 2-9.

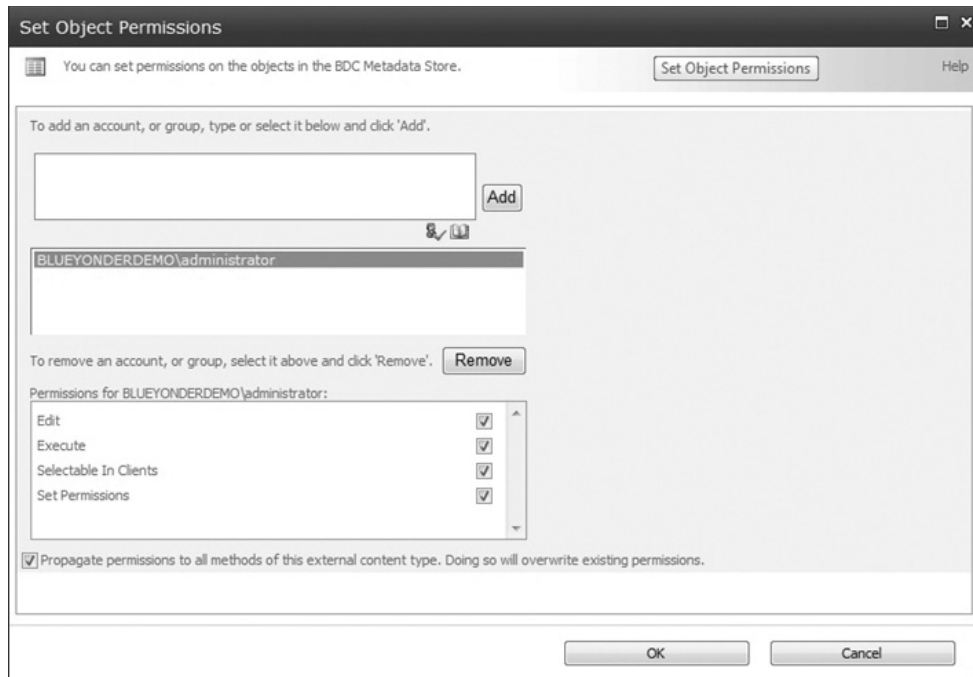


FIGURE 2-9

You can now navigate back to the SharePoint list and reload it. You will need to enter the SQL Azure permissions to authenticate (see Figure 2-10), after which the credentials will be cached on the server.

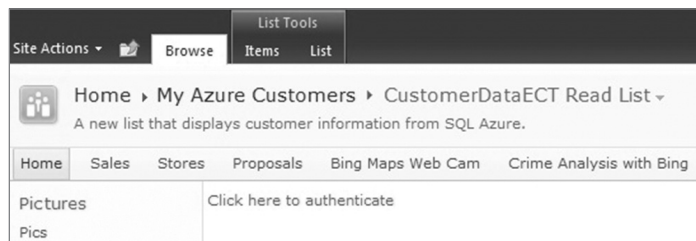
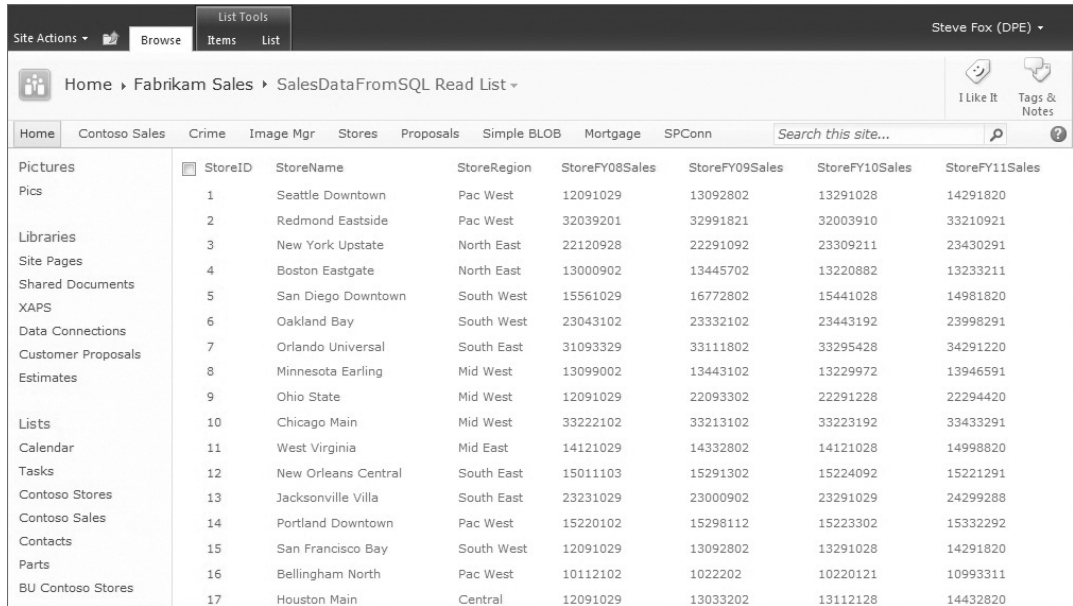


FIGURE 2-10

The final external list should now look something like Figure 2-11.

Now that you've completed the external list, which is the no-code part of the solution, let's move on to a more code-centric approach. In the next part of the chapter, you'll first create a WCF service to interact with the SQL Azure database, and then you'll create a web part to consume the service.



	StoreID	StoreName	StoreRegion	StoreFY08Sales	StoreFY09Sales	StoreFY10Sales	StoreFY11Sales
	1	Seattle Downtown	Pac West	12091029	13092802	13291028	14291820
	2	Redmond Eastside	Pac West	32039201	32991821	32003910	33210921
	3	New York Upstate	North East	22120928	22291092	23309211	23430291
	4	Boston Eastgate	North East	13000902	13445702	13220882	13233211
	5	San Diego Downtown	South West	15561029	16772802	15441028	14981820
	6	Oakland Bay	South West	23043102	23332102	23443192	23998291
	7	Orlando Universal	South East	31093329	33111802	33295428	34291220
	8	Minnesota Earling	Mid West	13099002	13443102	13229972	13946591
	9	Ohio State	Mid West	12091029	22093302	22291228	22294420
	10	Chicago Main	Mid West	33222102	33213102	33223192	33433291
	11	West Virginia	Mid East	14121029	14332802	14121028	14998820
	12	New Orleans Central	South East	15011103	15291302	15224092	15221291
	13	Jacksonville Villa	South East	23231029	23000902	23291029	24299288
	14	Portland Downtown	Pac West	15220102	15298112	15223302	15332292
	15	San Francisco Bay	South West	12091029	13092802	13291028	14291820
	16	Bellingham North	Pac West	10112102	1022202	10220121	10993311
	17	Houston Main	Central	12091029	13033202	13112128	14432820

FIGURE 2-11

Creating the WCF Service

The WCF service will be used to query the SQL Azure database.

1. Open Visual Studio 2010 and click File ⇨ New ⇨ Project ⇨ Blank Solution. Provide a name for the solution (e.g., SalesDataSolution) and click OK.
2. When Visual Studio creates the new solution, right-click the solution and select Add ⇨ New Project.
3. Select WCF ⇨ WCF Service Application. Provide a name for the project (e.g., SalesDataService) and click Add.
4. When the project has been added, double-click the `Service.svc` file and then right-click the default Service1 in the code and select Refactor ⇨ Rename. Rename the service to something that is more intuitive (e.g., SalesDataSvc). Do the same for IService1 (e.g., ISalesDataSvc).
5. Double-click your interface file (ISalesDataSvc) and amend the service contract code as per the following bolded code:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
```

```

namespace SalesDataService
{
    [ServiceContract]
    public interface ISalesDataSvc
    {
        [OperationContract]
        List<SalesData> GetAllSalesData();
    }
}

```

code snippet 076576 Ch02_Code.zip/ISalesDataSvc.cs

6. Click the Data tab and select Add New Data Source.
7. In the Choose a New Data Type dialog, select Database, and then click Next.
8. In the Choose a Data Model dialog, select Entity Database Model, and then click Next.
9. In the Choose Model Contents dialog, select Generate from Database, and then Click Next.
10. In the Choose Your Database Connection dialog, click New Connection. Enter the server name of your SQL Azure database. Select Use SQL Server Authentication, and enter the user-name and password for your SQL Azure database.
11. In the Select or enter a database name drop-down list, select the TotalSales database.
12. Click Test Connection to test the authenticated connection to your SQL Azure database.
13. In the Save entity connection settings in Web.config As field, enter **FabrikamSalesEntities**, and then click Next.
14. In the Choose Your Database Objects dialog, expand the Tables option, select the StoreSales table, and then click Finish.
15. Right-click the project and select Add ⇄ Class. Provide a name for the class (e.g., Sales Data) and amend the class properties as per the following bolded code:



Available for
download on
Wrox.com

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace SalesDataService
{
    public class SalesData
    {
        public string StoreID { get; set; }
        public string StoreName { get; set; }
        public string StoreRegion { get; set; }
        public string StoreFY08Sales { get; set; }
        public string StoreFY09Sales { get; set; }
        public string StoreFY10Sales { get; set; }
        public string StoreFY11Sales { get; set; }
    }
}

```



```
    }
}
```

code snippet 076576 Ch02_Code.zip/SalesData.cs

- 16.** Double-click the main service file (SalesDataSvc) and amend the service code, as per the following bolded code:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace SalesDataService
{
    public class SalesDataSvc : ISalesDataSvc
    {
        List<SalesData> listOfReturnSales = new List<SalesData>();
        FabrikamSalesEntities dc = new FabrikamSalesEntities();

        public List<SalesData> GetAllSalesData()
        {
            var sales = from salesData in dc.StoreSales
                        select salesData;

            GenerateReturnDataObject(sales);

            return listOfReturnSales;
        }

        private void GenerateReturnDataObject(IQueryable<StoreSale> sales)
        {
            foreach (var item in sales)
            {
                SalesData tempObj = new SalesData();
                tempObj.StoreID = item.StoreID.ToString();
                tempObj.StoreName = item.StoreName.ToString();
                tempObj.StoreRegion = item.StoreRegion.ToString();
                tempObj.StoreFY08Sales = item.StoreFY08Sales.ToString();
                tempObj.StoreFY09Sales = item.StoreFY09Sales.ToString();
                tempObj.StoreFY10Sales = item.StoreFY10Sales.ToString();
                tempObj.StoreFY11Sales = item.StoreFY11Sales.ToString();
                listOfReturnSales.Add(tempObj);
            }
        }
    }
}
```

code snippet 076576 Ch02_Code.zip/SalesDataService.cs

In the preceding code snippet, your WCF service is the handshake between your calling application (which will be a web part) and the SQL Azure data source. You use the entity data model to help provide a queryable layer against the SQL Azure database. The following line of code creates an instance of the data context that you'll use to query the data you added to SQL Azure:

```
...
    FabrikamSalesEntities dc = new FabrikamSalesEntities();
...
```

In this case you are using the service within a web part, but you can also use this service with other applications, such as Silverlight applications or .NET applications. This extends the use of your SQL Azure database to other applications.

You're using a list collection object (`listOfReturnSales`) that returns a collection of populated `SalesData` objects. Using a LINQ query (using the entity data model layer), you query the `StoreSales` table in the data context object (`dc`) to return the data in the table. Note that the `GenerateReturnDataObject` method is called to populate the list collection. This design enables you to add other filtered queries (e.g., to get sales from a specific region or company), and you can optimize your code by leveraging one method to populate the in-memory object. In the following code listing, you can see how the `foreach` block iterates through each of the items in the `sales` object to populate the list collection (`listOfReturnSales`).

```
...
private void GenerateReturnDataObject(IQueryable<StoreSale> sales)
{
    foreach (var item in sales)
    {
        SalesData tempObj = new SalesData();
        tempObj.StoreID = item.StoreID.ToString();
        tempObj.StoreName = item.StoreName.ToString();
        tempObj.StoreRegion = item.StoreRegion.ToString();
        tempObj.StoreFY08Sales = item.StoreFY08Sales.ToString();
        tempObj.StoreFY09Sales = item.StoreFY09Sales.ToString();
        tempObj.StoreFY10Sales = item.StoreFY10Sales.ToString();
        tempObj.StoreFY11Sales = item.StoreFY11Sales.ToString();
        listOfReturnSales.Add(tempObj);
    }
}
...
```

Finally, the WCF service returns the list collection to the calling application.

17. You can now deploy your WCF service. In this walkthrough, you'll deploy to your local IIS; however, you also have the option to deploy this service to Windows Azure. To deploy locally, create a folder on your server (e.g., `c:/MyNewService`).
18. Right-click the project and select Publish. In the Publish method drop-down, select the File System option and then browse to the folder you just created. Click Publish when done.
19. Open IIS Manager 7.0, right-click Sites, and then select Add Web Site.

20. Provide a name for the site (e.g., `SQLAzureWCFService`), browse to the new folder location where you just published your code, and then click **Connect as** ⇌ **Specific user** ⇌ **Set** — to set your username and password to connect to the service. Finally, change the Port number to something other than the default port 80.
21. Click the **Content** tab, right-click the `.svc` file in IIS, and then select **View in Browser**.

Note that you may raise an error if your application pool is not set to the correct version of the .Net Framework. To fix this error, click the application pool that corresponds to your service and select **Basic Settings**. In the **.Net Framework Version** drop-down, select the correct version.

You should now see something similar to Figure 2-12 — a service deployed to IIS that enables you to query SQL Azure.

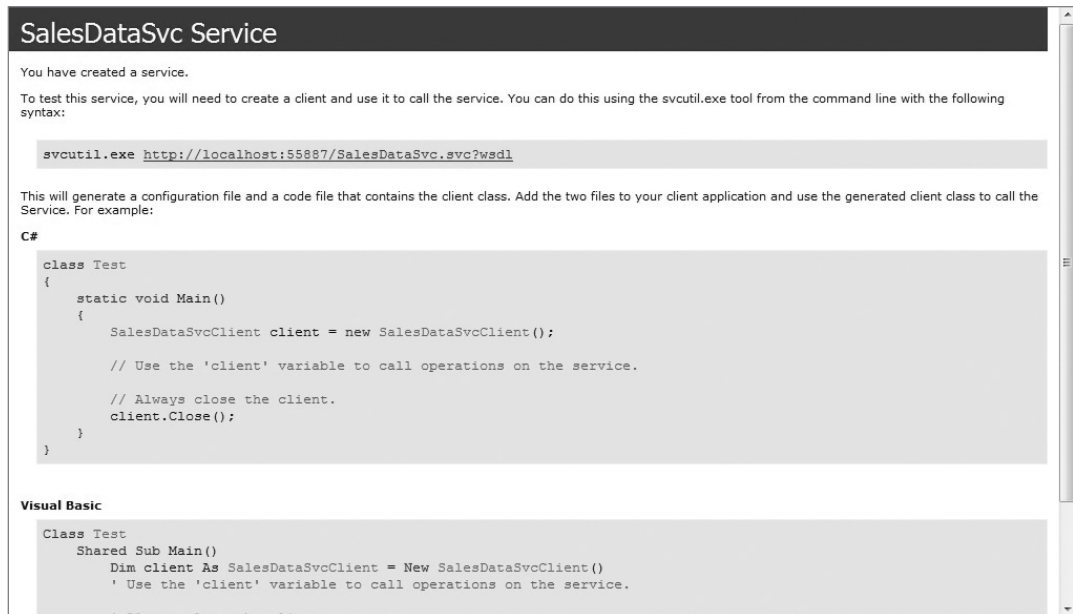


FIGURE 2-12

Now that you've created a WCF service, you can create a simple Windows Forms test application. For example, the following code snippet illustrates a Windows Forms application with three controls: a `DataGrid` (`datagridSalesData`) and two buttons (`btnExit` and `btnGetSales`). This small snippet of code enables you to add a service reference to your Windows Forms application and then test the service call. The returned data (results) will be bound to the `DataGrid` when the data is successfully returned from the web service call. Creating a small test application (either Windows Forms or console) enables you to both inspect a successful call and understand the data structure returned to the calling application.

```
...
using ServiceTestApplication.SQLAzureServiceReference;
using System.ServiceModel;
```

```

namespace ServiceTestApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnGetSales_Click(object sender, EventArgs e)
        {
            SalesDataSvcClient myWCFProxy = new SalesDataSvcClient();
            var results = myWCFProxy.GetAllSalesData();
            datagrdSalesData.DataSource = results;
        }
    }
}

```

If you're comfortable progressing without a test application, you can move to the next step, which will implement the WCF service in a web part using the ASP.NET Chart control.

Creating the Sales Web Part

The Sales web part will retrieve data using the WCF service and then data-bind the sales data to an ASP.NET Chart control.

1. Open the Visual Studio solution and right-click it.
2. Select Add ➤ New Project ➤ SharePoint, and then click Empty SharePoint Project. In the SharePoint Customization wizard, select Deploy as Farm Solution, and then click Finish. Provide a name for the new project (e.g., SalesDataChart) and click OK.
3. Right-click the newly added project and select Add ➤ New Item. Select SharePoint ➤ Web Part, and provide a name for the new web part (e.g., SalesData), and then click Add.
4. Right-click the project, select Add Reference, and add the *System.Web.DataVisualization.dll* to your project. (You may need to browse to the GAC to load the DLL — e.g., *c:/windows/assembly/System.Web.DataVisualization.dll*.)
5. You will create an in-memory object to data-bind, so you now need to add a custom class to your project. Right-click the project and select Add ➤ Class. Provide a name for the class (e.g., Sales). Amend the properties of the class as per the following bolded code:

```

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

```



Available for
download on
Wrox.com

```

namespace SalesDataChart
{
    class Sales
    {
        public string StoreID { get; set; }
        public string StoreName { get; set; }
        public string StoreRegion { get; set; }
        public Int32 FY08Sales { get; set; }
        public Int32 FY09Sales { get; set; }
        public Int32 FY10Sales { get; set; }
        public Int32 FY11Sales { get; set; }
        public Int32 TotalSales { get; set; }
    }
}

```

code snippet 076576 Ch02_Code.zip/SalesDataChart.cs

6. After you've added the ASP.NET Chart control library (System.Web.UI.DataVisualization.Charting.dll) and the custom class, you need to add the service reference to the WCF service. To do this, right-click the SharePoint project and select Add Service Reference. Add the service URL in the Address field and click Go. When the service resolves, provide a namespace for the service in the Namespace field (e.g., SQLAzureSalesDataService). Click OK to add the service reference to the project. You may need to add System.Drawing.dll and System.Collections.dll to your project, as well.
7. Double-click the main web part class (e.g., SalesData.cs) and then amend the code in the web part as per the following bolded code:



```

using System;
using System.ComponentModel;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using System.Web.UI.DataVisualization.Charting;
using System.Drawing;
using System.Collections.Generic;
using System.Collections;
using SalesDataChart.SQLAzureSalesDataService;
using System.ServiceModel;
using System.Linq;

namespace SalesDataChart.SalesData
{
    [ToolboxItemAttribute(false)]
    public class SalesData : WebPart
    {
        List<Sales> returnSalesData = new List<Sales>();

        protected override void OnLoad(EventArgs e)
        {

```

```

        BasicHttpBinding mySvcbinding = new BasicHttpBinding();
        //You will need to amend the URI to be your local domain.
        UriBuilder serviceURI = new
            UriBuilder("http://blueyonderdemo:55887/SalesDataSvc.svc");
        SalesDataSvcClient myWCFProxy = new SalesDataSvcClient(mySvcbinding, new
            EndpointAddress(serviceURI.Uri));

        var salesData = myWCFProxy.GetAllSalesData();

        foreach (var item in salesData)
        {
            Sales tempSalesObj = new Sales();
            tempSalesObj.StoreID = item.StoreID;
            tempSalesObj.StoreName = item.StoreName;
            tempSalesObj.StoreRegion = item.StoreRegion;
            tempSalesObj.FY08Sales = Int32.Parse(item.StoreFY08Sales);
            tempSalesObj.FY09Sales = Int32.Parse(item.StoreFY09Sales);
            tempSalesObj.FY10Sales = Int32.Parse(item.StoreFY10Sales);
            tempSalesObj.FY11Sales = Int32.Parse(item.StoreFY11Sales);
            tempSalesObj.TotalSales = tempSalesObj.FY08Sales +
                tempSalesObj.FY09Sales +
                tempSalesObj.FY10Sales + tempSalesObj.FY11Sales;
            returnSalesData.Add(tempSalesObj);
        }

        myWCFProxy.Close();
    }

    protected override void CreateChildControls()
    {
        Chart chrtSalesData = new Chart();
        chrtSalesData.ImageStorageMode = ImageStorageMode.UseImageLocation;

        chrtSalesData.Legends.Add("Legend");
        chrtSalesData.Width = 800;
        chrtSalesData.Height = 400;
        chrtSalesData.RenderType = RenderType.ImageTag;
        //This refers to a path that you will need to manually create.
        string imagePath = "~/_layouts/ChartImages/";
        chrtSalesData.ImageLocation = imagePath + "ChartPic_#SEQ(200,30)";
        chrtSalesData.Palette = ChartColorPalette.Berry;

        Title chartTitle = new Title("Store Sales", Docking.Top,
            new Font("Calibri", 12,
                FontStyle.Bold), Color.FromArgb(26, 59, 105));
        chrtSalesData.Titles.Add(chartTitle);
        chrtSalesData.ChartAreas.Add("Sales");

        chrtSalesData.Series.Add("Total Sales");

        foreach (Sales salesItem in returnSalesData)
        {
            chrtSalesData.Series["Total Sales"].Points.AddY(salesItem.
TotalSales);

```

```

    }

    chrtSalesData.BorderSkin.SkinStyle = BorderSkinStyle.Emboss;
    chrtSalesData.BorderColor = Color.FromArgb(26, 59, 105);
    chrtSalesData.BorderlineDashStyle = ChartDashStyle.Solid;
    chrtSalesData.BorderWidth = 1;
    this.Controls.Add(chrtSalesData);
}
}
}
}

```

code snippet 076576 Ch02_Code.zip/SalesData.cs

The preceding code snippet performs two main functions: It calls the WCF service to get the sales data stored in the SQL Azure database, and it data-binds the returned data to a Chart control.

The calling of the WCF service is done in an event that executes as the web part loads. This execution is accomplished using the OnLoad event, where you first create an instance of the service using the BasicHttpBinding object, then call the GetAllSalesData method using the service proxy (myWCFProxy), and then iterate through the returned data to populate a class-level list collection (returnSalesData):

```

...
    List<Sales> returnSalesData = new List<Sales>();

    protected override void OnLoad(EventArgs e)
    {
        BasicHttpBinding mySvcbinding = new BasicHttpBinding();
        UriBuilder serviceURI = new
UriBuilder("http://blueyonderdemo:55887/SalesDataSvc.svc");
        SalesDataSvcClient myWCFProxy = new SalesDataSvcClient(mySvcbinding, new
EndpointAddress(serviceURI.Uri));

        var salesData = myWCFProxy.GetAllSalesData();

        foreach (var item in salesData)
        {
            Sales tempSalesObj = new Sales();
            tempSalesObj.StoreID = item.StoreID;
            tempSalesObj.StoreName = item.StoreName;
            tempSalesObj.StoreRegion = item.StoreRegion;
            tempSalesObj.FY08Sales = Int32.Parse(item.StoreFY08Sales);
            tempSalesObj.FY09Sales = Int32.Parse(item.StoreFY09Sales);
            tempSalesObj.FY10Sales = Int32.Parse(item.StoreFY10Sales);
            tempSalesObj.FY11Sales = Int32.Parse(item.StoreFY11Sales);
            tempSalesObj.TotalSales = tempSalesObj.FY08Sales + tempSalesObj.
FY09Sales + tempSalesObj.FY10Sales + tempSalesObj.FY11Sales;
            returnSalesData.Add(tempSalesObj);
        }

        myWCFProxy.Close();
    }
}
...

```

Because the list collection is a class-level object, you can also leverage it in the second part of the code snippet. This is executed in the `CreateChildControls` method, which first creates an instance of the `Chart` object (which derives from `System.Web.DataVizualization.dll`) and then sets a number of properties for the `Chart`. Note that these properties range from width and height to chart graphic location (which in this app is in the `_layouts` directory), to title and formatting. After you've created an instance of the chart, you need to data-bind the sales data to it. This is done by iterating through the `returnSalesData` list collection and, for each item in the list, adding a new series in the chart using the `AddY` method. This new series represents the total sales that were calculated from the data returned from the WCF service:

```
...
    protected override void CreateChildControls()
    {
        Chart chrtSalesData = new Chart();
        chrtSalesData.ImageStorageMode = ImageStorageMode.UseImageLocation;

        chrtSalesData.Legends.Add("Legend");
        chrtSalesData.Width = 800;
        chrtSalesData.Height = 400;
        chrtSalesData.RenderType = RenderType.ImageTag;
        string imagePath = "~/_layouts/ChartImages/";
        chrtSalesData.ImageLocation = imagePath + "ChartPic_#SEQ(200,30)";
        chrtSalesData.Palette = ChartColorPalette.Berry;

        Title chartTitle = new Title("Store Sales", Docking.Top,
            new Font("Calibri", 12,
                FontStyle.Bold), Color.FromArgb(26, 59, 105));
        chrtSalesData.Titles.Add(chartTitle);
        chrtSalesData.ChartAreas.Add("Sales");

        chrtSalesData.Series.Add("Total Sales");

        foreach (Sales salesItem in returnSalesData)
        {
            chrtSalesData.Series["Total Sales"].Points.AddY(salesItem.
TotalSales);
        }

        chrtSalesData.BorderSkin.SkinStyle = BorderSkinStyle.Emboss;
        chrtSalesData.BorderColor = Color.FromArgb(26, 59, 105);
        chrtSalesData.BorderlineDashStyle = ChartDashStyle.Solid;
        chrtSalesData.BorderWidth = 1;
        this.Controls.Add(chrtSalesData);
    }
...

```

8. Optionally, you can edit the `elements.xml` and `SalesData.webpart` files to provide users who may want to find and add your web part with a more intuitive experience. For example, the first XML code snippet shows how you can amend the `elements.xml` file, and the second is the `SalesData.webpart` file:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/" >
    <Module Name="SalesData" List="113" Url="_catalogs/wp">

```



```

    <File Path="SalesData\SalesData.webpart" Url="SalesData.webpart"
Type="GhostableInLibrary">
    <Property Name="Group" Value="SQL Azure Sales Data" />
    </File>
</Module>
</Elements>

<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="SalesDataChart.SalesData.SalesData, $SharePoint.Project.
AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">Azure Sales Web Part</property>
        <property name="Description" type="string">Sales data from SQL Azure that uses
the ASP.NET Chart control.</property>
      </properties>
    </data>
  </webPart>
</webParts>

```

9. You can now build and deploy the web part to SharePoint by right-clicking the project and selecting Deploy.
10. After you deploy successfully, open SharePoint and click Site Actions ➤ Edit Page. Select Add a Web Part (or simply click the Insert tab) and then navigate to your newly added web part and click Add.

After you add the web part to SharePoint, the web part loads and, at the same time, calls the WCF service to retrieve sales data. It then binds the data from the WCF service to the Chart control, which should resemble what is shown in Figure 2-13.

Note that you may need to amend the `web.config` file of your SharePoint site. The following illustrates the three recommended amendments and the sections within which you would need to make those updates:

```

...
<httpHandlers>
<add path="ChartImg.axd" verb="GET,HEAD,POST"
type="System.Web.UI.DataVisualization.Charting.ChartHttpHandler,
System.Web.DataVisualization, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
validate="false" />
</httpHandlers>
...
<handlers>
<add name="ChartImageHandler" preCondition="integratedMode" verb="GET,HEAD,POST"
path="ChartImg.axd" type="System.Web.UI.DataVisualization.Charting.ChartHttpHandler,
System.Web.DataVisualization, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />

```

```

</handlers>
...<appSettings>
<add key="ChartImageHandler" value="storage=file;timeout=20;dir=c:\Temp\" />
</appSettings>
...

```

At this point, you've created the two separate pieces of the BI solution; now you need to put them together. In the next section, you'll create a SharePoint site that combines both the external list and the Sales web part to provide a concerted view of these artifacts.

Creating the Sales Dashboard UI

Because you've created the components of the BI solution, creating a dashboard will be very straightforward; all you need to do is create a new site in SharePoint (e.g., a Team Site) and then add the external list on the left-hand side of the landing page and add the web part to the right-hand side of the page.

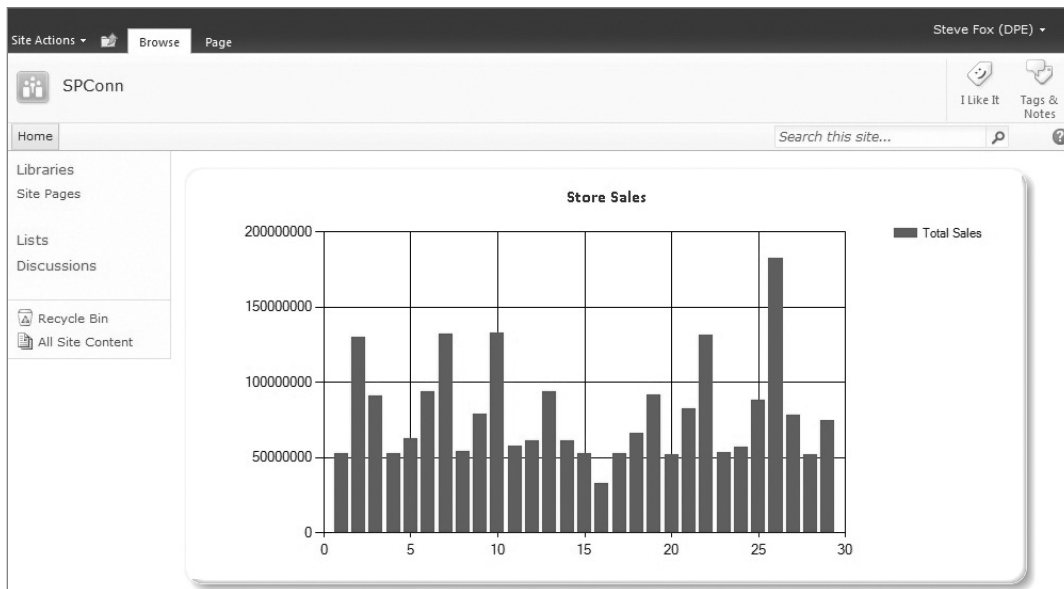


FIGURE 2-13

The Sales dashboard will render the no-code BCS external list and the code-based web part.

1. Navigate to your SharePoint site. Create a new site by clicking View All Site Content ⇨ Create ⇨ Blank Site. Provide a name for your new site (e.g., Sales) and a URL suffix (e.g., sales).
2. Navigate to your external list (e.g., Store Sales). Click the List tab and then click Create View. Select Standard View, as shown in Figure 2-14.
3. Provide a name for the view (e.g., Summary Sales Data), select Create a Public View, check the fields you want to expose in the view, and then click OK. Figure 2-15 illustrates this process.
4. Return to your newly created site (e.g., Sales), and click Site Actions ⇨ Edit Page.



FIGURE 2-14

5. Select Insert ⇄ Existing List. Select Store Sales from the options and click Add.
6. Load the view of the external list you just created by clicking Edit Web Part and then selecting the newly created view.

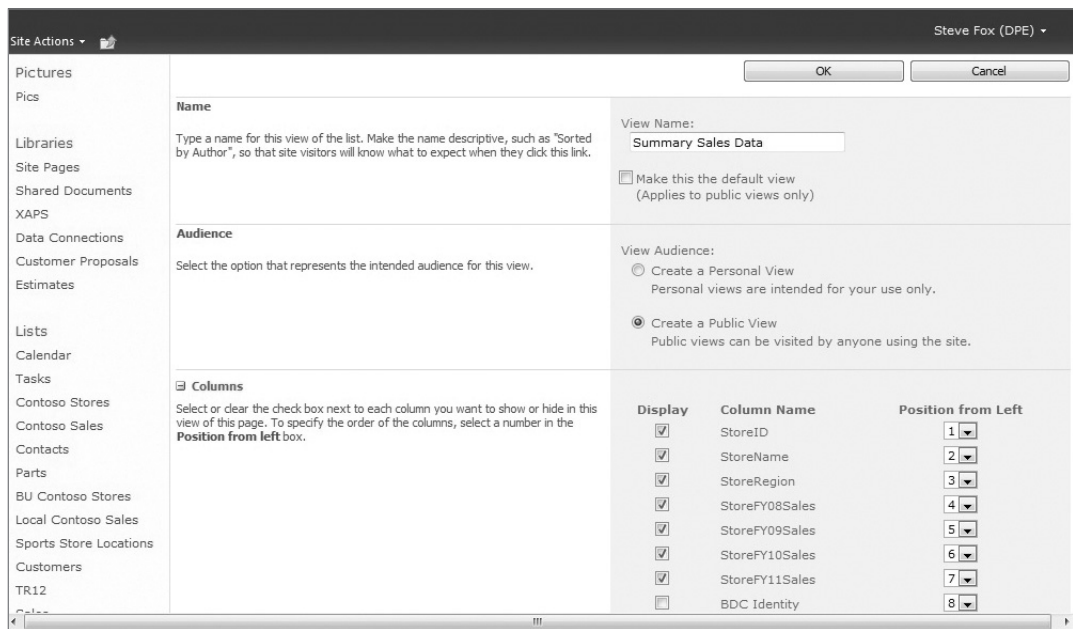


FIGURE 2-15

7. Click Site Actions ⇄ Edit Page and then either click Add a Web Part or click within a specific region on the web page. Select the Insert tab, click Web Part, and then navigate to the SQL Azure Sales Data category and add the Sales Chart web part by clicking Add.

The result of creating the no-code solution and the code-based solution should look similar to Figure 2-16: an external list on the left (that enables you to create, read, update, and delete records from SQL Azure), and the ASP.NET Chart control in a web part on the right (that uses the WCF service to query data from SQL Azure and then dynamically generate a new chart).

Congratulations! At this point, you've finished creating the BI solution. The next question you might have is how to extend it. While we'll leave you mostly to your own creativity in that regard,

here are a couple of suggestions. The first is to extend the WCF service to provide additional web methods for passing filters. You could then provide a richer Chart web part experience by enabling service calls that dynamically generate different views on the data. Alternatively, you could also use Silverlight to create a chart-based view (using the Chart Toolkit); this is an excellent and very straightforward way to create a compelling BI experience.

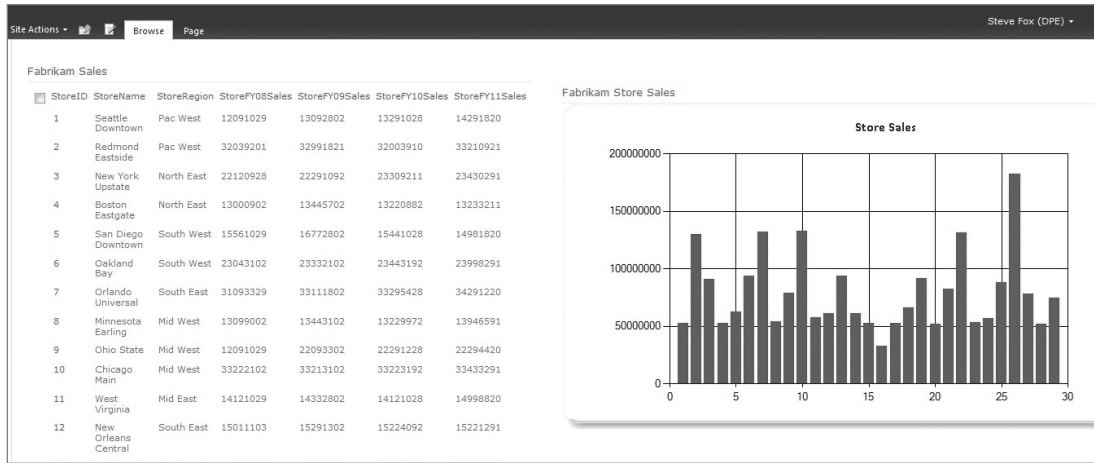


FIGURE 2-16

You could also deploy the WCF service and entity data model to Windows Azure; as mentioned earlier, this would enable you to leverage the SQL Azure data not only in an on-premises solution, but also in the cloud within SharePoint Online and Office 365 (assuming you use Silverlight as the application that consumes the WCF service, because you cannot call external services from a regular sandboxed solution). Finally, you could swap out the WCF service call and leverage the SharePoint server object model or the client object model to interact with the external list. This would entail one thoroughfare for the data connection to SharePoint, but two different ways of surfacing that data within SharePoint: one for managing the data and another to render tabular and chart-based views of the data.

SUMMARY

This chapter illustrated how to create a modest BI solution that comprised a no-code and code approach, using SQL Azure as the main cloud-based database. The no-code approach was BCS-based and surfaced SQL Azure data in an external list, and the code-based approach used the ASP.NET Chart control within a web part that leveraged WCF to query SQL Azure. Together, these different applications demonstrated two ways to create a BI application using SQL Azure and SharePoint.

While the solution itself was not overly complex, one important takeaway should be the way in which you used the SQL Azure database within your solution: you used an external list and a service-based query to interact with the data. The external list connected to a SQL Azure database is an easy way to tie cloud-based data to SharePoint, and it provides you with a way to manage

data into and out of SQL Azure using a SharePoint list. SharePoint also provides ways to discretely control the security and permissions of the list. The service-based proxy you built to communicate with SQL Azure can be used across multiple application types and, in some cases, can be used in SharePoint Online. For example, with the sandboxed solution constraint in SharePoint Online, you can use Silverlight and services deployed to Windows Azure as an integration technique.

There are many more and different ways in which you can interact with cloud-based data. Some of these ways are discussed in this book, and others you will discover as you explore further the area of cloud-based BI. There is also a great deal of potential with cloud-based data; the oft-stated value-add of the Windows Azure usage model notwithstanding, you can use the cloud to scale data, use Windows Azure to move more code off of the server, and provide a way to create portable BI applications across SharePoint Server and SharePoint Online.

In the next chapter, you'll move beyond SQL Azure data and dive deeper into how you can use Windows Azure to host custom WCF services that you can consume in SharePoint.

ADDITIONAL REFERENCES

The following list provides some additional references you might find useful:

- **SQL Azure Introduction** — www.microsoft.com/en-us/sqlazure/database.aspx
- **Migrating Databases to SQL Azure** — <http://msdn.microsoft.com/en-us/library/ee730904.aspx>
- **Windows Azure Free Trial Offer** — www.microsoft.com/windowsazure/free-trial/sharepoint-integration/
- **Overview of SQL Server Reporting Services (SSRS)** — <http://technet.microsoft.com/en-us/library/cc526677.aspx>
- **Overview of BCS in MSDN** — <http://msdn.microsoft.com/en-us/magazine/ee819133.aspx>
- **Overview of WCF Data Services** — <http://msdn.microsoft.com/en-us/library/cc668794.aspx>

3

Building a Windows Azure–Based Service

WHAT'S IN THIS CHAPTER?

- Getting started with Windows Azure and SharePoint
- Understanding Windows Azure table storage
- Creating a Windows Communication Foundation Service in Windows Azure
- Connecting to Windows Azure with Business Connectivity Services

Windows Azure is a good platform for running your services and data in the cloud, but how do you leverage these services and data from within SharePoint? In this chapter, you will learn about the Windows Azure Table service, which enables Windows Azure to host very large and scalable flat datasets in the cloud. (Azure storage services also include the Blob service and the Queue service, but these services are outside the scope of this chapter.) Windows Azure can also host your Windows Communication Foundation (WCF) services, and you will learn how to create a WCF service in Windows Azure that reads and writes data in a Windows Azure table. Then you will learn how to create a Business Connectivity Services (BCS) model to consume the Windows Azure service without writing any code.

The scenario this chapter uses leverages county tax data from all the states from 2007. You will see how to move this open-source government data into your Windows Azure table and consume it in your SharePoint applications. Although the example uses data from somewhere else, you could apply the same techniques for your own data, including moving some of your SharePoint lists to the cloud.

Before digging into how everything works, it is important to start at the beginning. Getting started can be one of the most challenging aspects to any new technology; and when you put

two robust technologies together Windows Azure and SharePoint Server 2010, the challenges often multiply. The next section describes how to set up your development environment to build Windows Azure solutions with SharePoint and, more importantly, how to avoid some issues.

GETTING STARTED

You need two things to develop Windows Azure and SharePoint solutions: SharePoint 2010 and the Windows Azure SDK and tools. Also, because you will most likely be developing solutions on a single machine using Hyper-V, you will need to work around some of the technical issues with running the Windows Azure compute emulator and SharePoint on the same machine. You will see one way to solve this problem using networking tunneling.

Installing SharePoint 2010

Getting started with SharePoint is easy because a prebuilt SharePoint 2010 Information Worker Virtual Machine (IW VM) is located at www.microsoft.com/download/en/details.aspx?id=21099. Download and run the IW VM in Hyper-V. You will need a 64-bit Windows Server 2008 R2 machine with at least 8 GB of memory (16 GB is even better). Another option is to run the SharePoint Easy Setup Script, located at <http://channel9.msdn.com/Shows/SharePointSideshow/Building-a-SharePoint-Development-Machine-Using-the-Easy-Setup-Script>. The Easy Setup Script will automatically build a native install or boot to a Virtual Hard Drive (VHD) SharePoint 2010 developer machine. Both are good options; however, if you want to do Windows Azure development, you need to use Hyper-V or have two machines, due to incompatibilities between SharePoint and the Windows Azure SDK. You will see how to work around this limitation in the “Creating an SSH Tunnel” section.

Installing Windows Azure SDK and Tools

Once you have SharePoint up and running, download and install the Windows Azure Tools for Microsoft Visual Studio 1.4 from <http://go.microsoft.com/fwlink/?LinkID=128752>. You only need to download and install the `VSCloudService.exe`, which contains both the SDK and Visual Studio tools. Do *not* install this SDK on the same machine you have SharePoint running on. If you do, you will most likely break one or all of the following in some way: Business Connectivity Services (BCS), User Profile Service, or Managed Metadata. This is a known issue, which likely will be fixed in the near future. For now, you need to install them on separate machines. Or, in this case, install the Windows Azure SDK on the Hyper-V host machine that is running the SharePoint VHD.

Creating an SSH Tunnel

Once you have the Windows Azure SDK installed on the Hyper-V host, this presents another problem. The Windows Azure tools use both a local compute emulator and a storage emulator. Typically, the compute emulator runs at 127.0.0.1:8080. The challenge is accessing this endpoint address from your SharePoint VHD. It is not possible to access this address externally from the machine that is running the compute emulator. This is done as a security mechanism to protect your machine from attack, but it makes it very difficult for the SharePoint and Windows Azure developer. There are a couple of possible solutions to this problem.

The first approach is to simply publish your Windows Azure services to the cloud before connecting to them with SharePoint. This means that SharePoint never actually connects to your Windows Azure developer environment. From a connectivity aspect, this is the easiest approach, but it makes it very difficult to debug and rapidly iterate over your code.

The second approach uses a Secure Shell (SSH) tunnel. Although the SSH tunnel is a little more complicated to understand and set up, it provides the same development experience as having everything installed on a single machine. I would like to credit Emmanuel Huna with documenting this approach on his blog at http://blog.ehuna.org/2009/10/accessing_the_windows_azure_10.html. His post is not actually targeted at the SharePoint and Windows Azure developer, but it works very nicely for this purpose. The blog post is very clear to follow, but the following steps summarize the basic concept of the SSH tunnel approach:

1. Install the SSH server called freeSSHd, from www.freesshd.com, on the Hyper-V host, which is Windows Server 2008 R2. This is the machine where you are running the Windows Azure SDK.
2. Configure freeSSHd to open a tunnel on the host machine.
3. Install the SSH client called PuTTY, from www.chiark.greenend.org.uk/~sgtatham/putty, on the SharePoint VHD and configure it to connect to the host machine.

Once you start the host SSH server and connect the SSH client to the host, you will be able to access the Windows Azure compute emulator at 127.0.0.1:8080 from the SharePoint server running in Hyper-V. This is a little complicated if you have not worked with SSH tunnels, but Emmanuel's blog post is very clear, and the diagram in Figure 3-1 should help you understand how all the various pieces fit together.

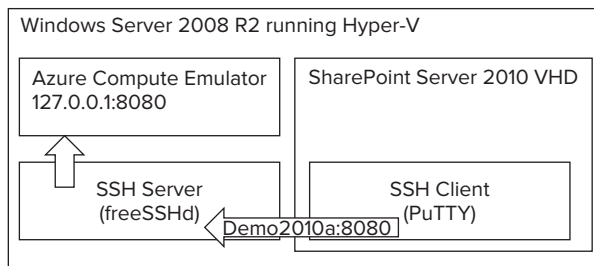


FIGURE 3-1

Initializing the Windows Azure Compute and Storage Emulators

After installing the Windows Azure SDK, you need to start the Windows Azure compute and storage emulators. Before starting the storage emulator, however, you need to initialize it. The Windows Azure storage emulator uses SQL Server to emulate Windows Azure storage tables, blobs, and queues. The initialization process creates the SQL database and tables to emulate these. When you start the storage emulator for the first time, the application automatically runs the initialization process. Although this generally works well, depending on the complexity of your SQL Server setup, you may have to run the tool manually from the command line. To do so, open the Windows Azure

SDK command prompt as an administrator and run the following command (note the period at the end, which specifies using the default SQL Server instance):

```
DSInit /sqlInstance:.
```

After running the `DSInit` command, you can start the storage emulator from the Windows Azure icon running in the task tray. For more details, see the MSDN help page at <http://msdn.microsoft.com/en-us/library/gg433132.aspx>.

At this point, you should have everything installed and connected to begin developing Windows Azure and SharePoint solutions.

USING WINDOWS AZURE TABLE STORAGE

In Chapter 2, “Using SQL Azure for Business Intelligence,” you learned about SQL Azure databases. Windows Azure storage adds tables, queues, blobs, and drives. In this chapter, you will learn about Windows Azure table storage. Windows Azure tables are a collection of entities. Think of a table entity as being similar to a row in a database. Tables in Windows Azure are not restricted to a schema — that is, each entity (or row) can have a different shape or set of properties. An entity can have up to 255 properties, including the three reserved properties: `PartitionKey`, `RowKey`, and `Timestamp`. These three properties are automatically added to every entity. Tables can be split, or partitioned, to help Windows Azure scale them to very large sizes. The `PartitionKey` is the unique ID for a given partition, and the `RowKey` is the unique ID for a given row in a partition. This means that the `PartitionKey` plus the `RowKey` uniquely identify any row in a table. The `Timestamp` is the modified time of the entity.

Entities support only a small number of data types:

- `byte[]`
- `bool`
- `DateTime`
- `double`
- `Guid`
- `Int32` or `int`
- `Int64` or `long`
- `String`

`String` is the default property type for a property.

This chapter focuses on the Windows Azure service side of the solution. You will learn more about Windows Azure table storage in Chapter 8, “Financial Modeling with Excel Services and Windows Azure,” including how to create and populate Windows Azure tables. You will also leverage some of the samples located in the Windows Azure SDK, at <http://msdn.microsoft.com/en-us/library/gg432966.aspx>.

Creating the Windows Azure Web Role

In this project, you will create a WCF service that runs in a Windows Azure web role. You will add to this Visual Studio solution throughout this chapter. The purpose of this service is to read from and write data to a Windows Azure table, which you will create later in this chapter, and then the service will be called from SharePoint's BCS.

1. Open Visual Studio and create a new Windows Azure project called **CountyTax**, as shown in Figure 3-2. Remember that this example uses county tax data from across the U.S.A.

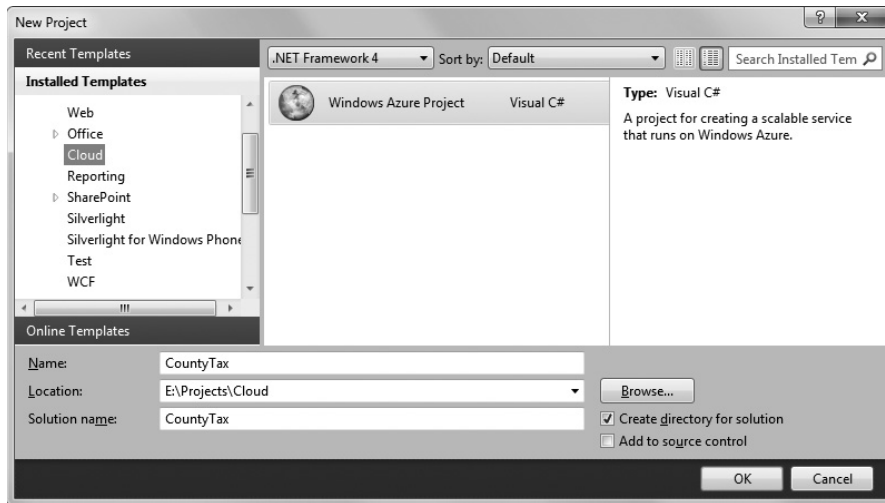


FIGURE 3-2

2. Add a WCF service web role called **CountyTaxService**, as shown in Figure 3-3. The WCF service web role is just a special version of the standard ASP.NET web role, with some added code to jump-start your WCF service development.

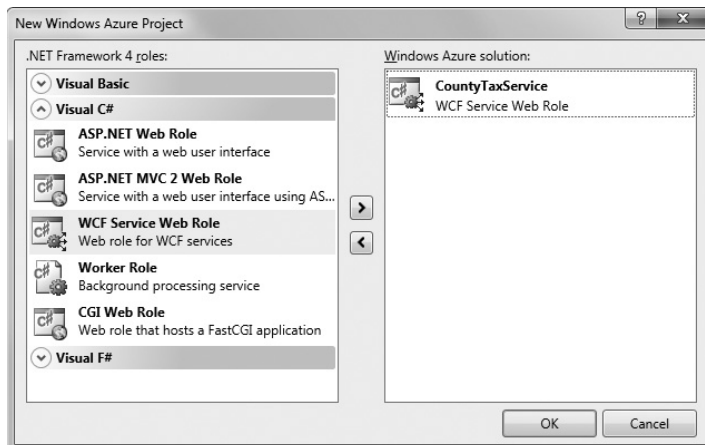


FIGURE 3-3

3. As shown in Figure 3-4, your Windows Azure solution is empty. Press F5 to ensure that everything is set up correctly and functioning. You can also delete the `IService1.cs` and `Service1.svc` service files, as you will not need them. You will also need references to the `System.Data.Services.Client` and `Microsoft.VisualBasic` namespaces.

As mentioned, you will add to this Visual Studio solution throughout the chapter. You'll start by adding the county tax data. The next section describes how to create the Entity Data Model (EDM), and how to create a Windows Azure table and bulk import some data.

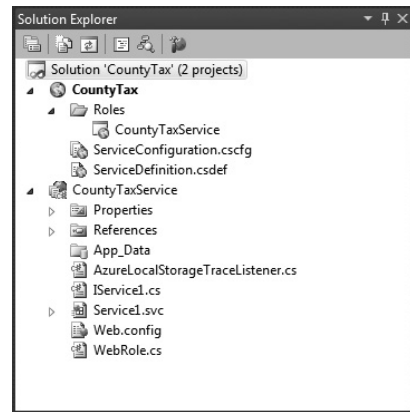


FIGURE 3-4

Creating the Entity Data Model

Azure tables are based on entities from the ADO.NET Entity Framework. You can define an entity based on the shape of the data — that is, the properties the entity contains. Although there are many data sets to explore and consume, this example uses county tax data from the Data.gov site. The county tax data is located at <http://explore.data.gov/Population/Tax-Year-2007-County-Income-Data/wvps-imhx>, as shown in Figure 3-5.

The site allows you to view, filter, visualize, export, discuss, and embed the data. In this case, you want to export the data as a comma-delimited `.csv` file. Click the Export button and choose CSV from the list of supported export formats.

After downloading the `.csv` file, you need to do a little cleanup of the headers. By default, the site creates the headers with the same text shown on the site. While normally this is good, it is better to change these to match the entity property names that you will be using. Open the `.csv` file using Notepad and change the first line to the following:

```
StateCode,CountyCode,State,CountyName,TotalReturns,TotalExemptions,
AdjustedGrossIncome,Wages,DividendIncomes,InterestIncome
```

Now that you have the data that you want to use as the basis for your table, it is time to create an entity to represent this data schema. Add a class to the `CountyTaxService` project called `CountyTaxEntity`. Replace the generated code with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.WindowsAzure.StorageClient;

namespace CountyTaxService
{
    // Entity for Windows Azure table
    public class CountyTaxEntity : TableServiceEntity
    {
        public CountyTaxEntity()
        {
        }

        // State Code
        public string StateCode { get; set; }
    }
}
```



```

// County Code
public string CountyCode { get; set; }
// State Abbreviation
public string State { get; set; }
// County Name
public string CountyName { get; set; }
// Total Number of Tax Returns
public string TotalReturns { get; set; }
// Total Number of Exemptions
public string TotalExemptions { get; set; }
// Adjusted Gross Income (In Thousands)
public string AdjustedGrossIncome { get; set; }
// Wages and Salaries Incomes (In Thousands)
public string Wages { get; set; }
// Dividend Incomes (In Thousands)
public string DividendIncomes { get; set; }
// Interest Income (In Thousands)
public string InterestIncome { get; set; }

}
}

```

code snippet 076576 Ch03_Code.zip/CountyTaxEntity.cs

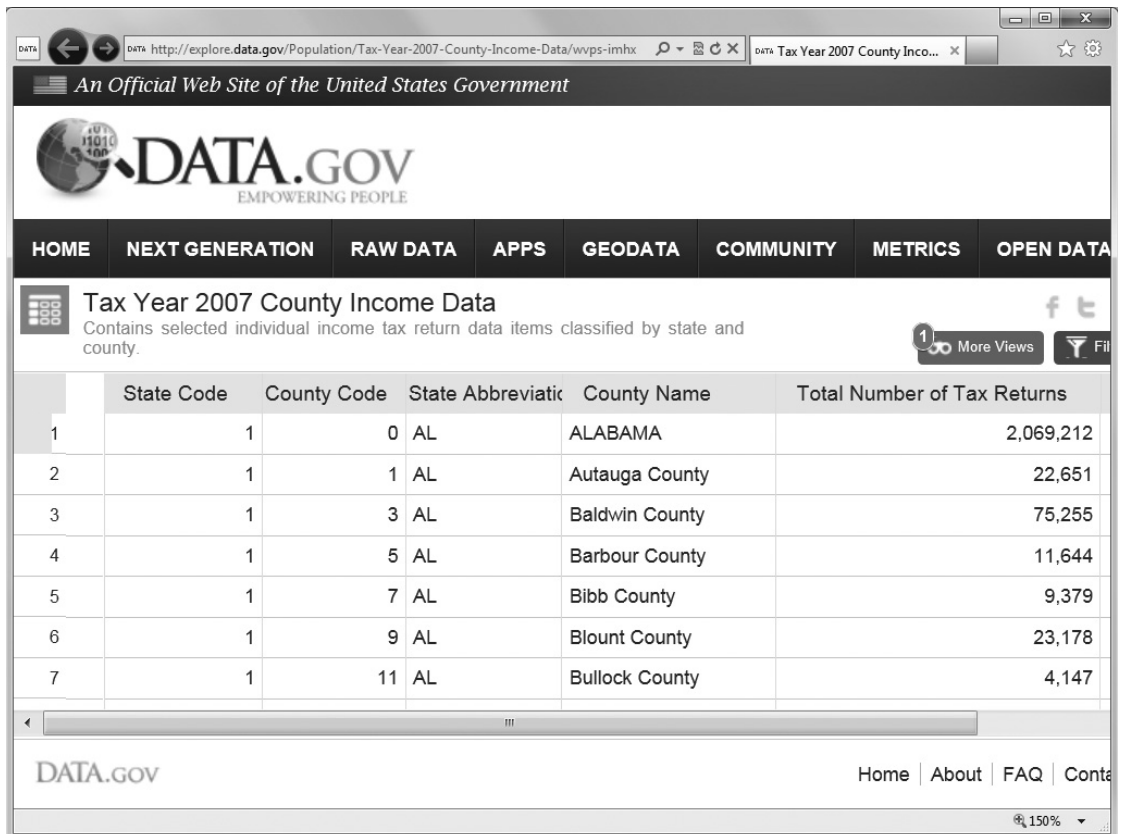


FIGURE 3-5

The preceding code defines an entity called `CountyTaxEntity` with string properties for the same fields contained in the comma-delimited dataset you downloaded from Data.gov. Note that this entity derives from `TableServiceEntity`. The `TableServiceEntity` includes the other required properties for a Windows Azure table: `PartitionKey`, `RowKey`, and `TimeStamp`.

Populating the Windows Azure Table

Before you can use the application, you need a way to create and populate the Windows Azure table with the .csv data file you downloaded. In this section, you will learn how to create an upload page and code to bulk add the data.

1. Start by adding a new web form to the `CountyTaxService` project called **Import.aspx**. Replace the generated code in the `Import.aspx` file with the following code:



Available for
download on
Wrox.com

```
<%@ Page Title="Import" Language="C#"
    MasterPageFile="~/Site.Master"
    AutoEventWireup="true"
    CodeBehind="Import.aspx.cs"
    Inherits="CountyTaxService.Import" %>

<asp:Content ID="HeaderContent"
    ContentPlaceHolderID="HeadContent"
    runat="server">
</asp:Content>
<asp:Content ID="BodyContent"
    ContentPlaceHolderID="MainContent"
    runat="server">

    <p>
        Upload a comma-delimited .csv file containing your contacts.
    </p>
    <br />
    <br />
    <table border="0" cellpadding="0" cellspacing="0">
        <tr>
            <td valign="top">
                <asp:FileUpload ID="Uploader" runat="server"
                    Style="margin-top: 0px"
                    Height="24px"
                    Width="472px" />&nbsp;
            </td>
            <td valign="top">
                <asp:Button ID="cmdUpload" runat="server"
                    Text="Import" OnClick="cmdUpload_Click" />
            </td>
        </tr>
    </table>
    <br />
    <asp:Label ID="lblInfo" runat="server"
        EnableViewState="False"
        Font-Bold="True">
</asp:Label>
```

```
<br />
<br />
</asp:Content>
```

code snippet 076576 Ch03_Code.zip/Import.aspx

This code will create a page from which you can browse for the .csv data file and import it into your Windows Azure table (see Figure 3-6).

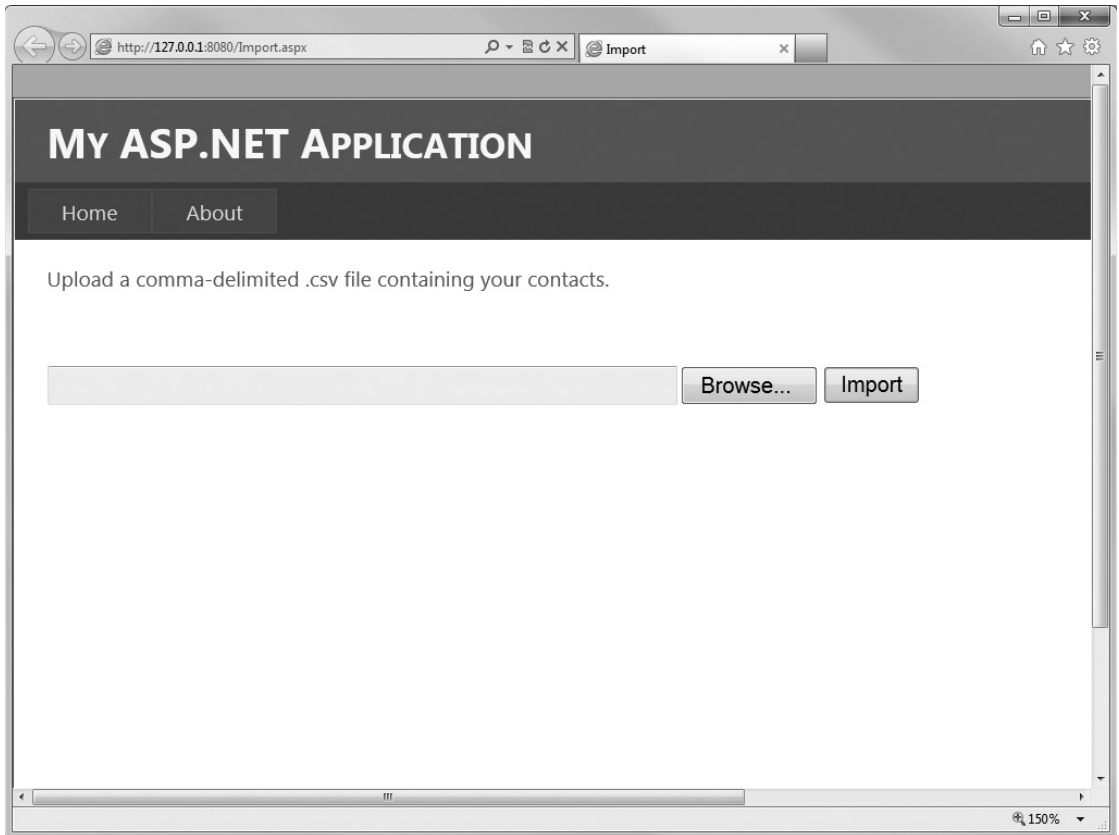


FIGURE 3-6

Actually, to get your page to look exactly like this, you need to add the `Site.Master` and `Site.css` pages to your site. You can find these in the included source code, or you can copy them from a standard web role project created with Visual Studio.

The important code happens on the click event of the `Import` button.

2. Add the following code to the `Import.aspx.cs` code-behind file. This code will read the file stream and start the import process.



Available for
download on
Wrox.com

```
protected void cmdUpload_Click(object sender, EventArgs e)
{
    // Check for file.
    if (this.Uploader.PostedFile.FileName == string.Empty)
    {
        this.lblInfo.Text = "No file specified.";
        this.lblInfo.ForeColor = System.Drawing.Color.Red;
    }
    else
    {
        try
        {
            // Import csv data into a table.
            this.ImportData(this.Uploader.PostedFile.InputStream);
        }
        catch (Exception err)
        {
            this.lblInfo.Text = err.Message;
            this.lblInfo.ForeColor = System.Drawing.Color.Red;
        }
    }
}
```

code snippet 076576 Ch03_Code.zip/Import.aspx.cs

3. The Import button's click event handler calls the `ImportData` method. The `ImportData` method reads the comma-delimited file stream and parses out each row and value into a `System.Data.DataTable` object. Add the following code to the `Import.aspx.cs` class:



Available for
download on
Wrox.com

```
// Process csv file and import data into the application.
private void ImportData(Stream csvStream)
{
    this.lblInfo.Text = "";

    // Create data table to hold the data in memory.
    DataTable dt = new DataTable();

    // Parse the csv file and add the data to the data table.
    using (var csvFile = new TextFieldParser(csvStream))
    {
        csvFile.TextFieldType = FieldType.Delimited;
        csvFile.SetDelimiters(",");
        csvFile.HasFieldsEnclosedInQuotes = true;

        // Read the first row of data
        // (which should contain column names).
        string[] fields = csvFile.ReadFields();

        // Add columns to data table, using first (header) row.
        DataColumn col = null;
        List<int> fieldIndices = new List<int>();

        if (!csvFile.EndOfData)
        {
```



```

// The FirstName field is required,
// since it's used for the partition key and row key.
if (!fields.Contains("FirstName"))
{
    this.lblInfo.Text =
        "The .csv file must contain a FirstName field, " +
        "named in the first row of data.";
    this.lblInfo.ForeColor = System.Drawing.Color.Red;
}

// Create array of property names from CountyTaxEntity.
List<string> propertyNames = new List<string>();
foreach (PropertyInfo info in
    typeof(CountyTaxEntity).GetProperties())
{
    propertyNames.Add(info.Name);
}

// Add a field to the data table if it
// matches one defined by CountyTaxEntity.
for (int i = 0; i < fields.Length; i++)
{
    if (propertyNames.Contains(fields[i]))
    {
        col = new DataColumn(fields[i]);
        dt.Columns.Add(col);

        // Track the field's index,
        // so we know which ones to add data for below.
        // This way any fields other than those named
        // by CountyTaxEntity will be ignored.
        fieldIndices.Add(i);
    }
}

// Add data from each row to data table
// where it matches column name.
DataRow row = null;
while (!csvFile.EndOfData)
{
    // Get the current row from the csv file.
    string[] currentRow = csvFile.ReadFields();

    // Create a new row in the data table.
    row = dt.NewRow();

    // Copy the data from the csv to the data table.
    foreach (var index in fieldIndices)
    {
        row[index] = currentRow[index];
    }

    // Add the row.

```

```

        dt.Rows.Add(row);
    }
}

// Insert values from the data table
// into a Windows Azure table.
try
{
    DataLayer.BulkInsertCountyTax(dt);

    // Redirect to main page.
    Response.Redirect("Default.aspx");
}
catch (ApplicationException e)
{
    this.lblInfo.Text =
        "Error importing csv file: " + e.Message;
    this.lblInfo.ForeColor =
        System.Drawing.Color.Red;
}
}

```

code snippet 076576 Ch03_Code.zip/Import.aspx.cs

There is nothing specific to Windows Azure in the `ImportData` code. It iterates through each row and creates a corresponding `DataRow` object in the `DataTable`. See the code comments to understand exactly what is going on in each code block. Note that after it builds a `DataTable` object in memory, it passes it to the `DataLayer` class, which you will add next. The `DataLayer` class contains a static method called `BulkInsertCountyTax`, which has one parameter for the `DataTable`.

To recap, before adding the `BulkInsertCountyTax` method, at this point you have converted the comma-delimited data file to an in-memory `DataTable`. Next, you will convert the in-memory `DataTable` to a Windows Azure table.

4. Add a new class project item to the `CountyTaxService` project called `DataLayer.cs`. The `DataLayer` class will contain all the code to create, read, write, update, delete, and list data from the Windows Azure table. You need to add a number of references to support the Windows Azure tables. Add references to `Microsoft.WindowsAzure.StorageClient`, `System.Data.Services.Client`, and `System.Data.DataSetExtensions`. Add the following using statements to the top of the class:

```

using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
using System.Data.Services.Client;

```

code snippet 076576 Ch03_Code.zip/DataLayer.cs

The next step will create the actual bulk import code. The first couple of lines delete and re-create the table, if it exists. This is simply done for the demo and enables the demo to be rerun repeatedly. Next, the `DataTable` is sorted by the `StateCode`. This is done because



Available for
download on
Wrox.com

the table is partitioned by `StateCode`. Partitioning is one way that Windows Azure scales tables, by splitting them based on the `PartitionKey`. When doing a bulk update, you can only update against one partition per update. This means that you need to ensure that all the items in the batch match. The technique used here is to keep track of the `StateCode`; and when it changes, do the batch update before moving on to the next `StateCode`. Note there is a limit of 100 items per batch update. The code checks for either condition, the `StateCode` changing or the item count, before it does an update.

5. The `BulkImportCountyTax` method creates a `TableServiceContext` object called `context`. The `context` object holds the updates in memory until the `SavesChanges` method is called, as previously explained. The code iterates through the `DataTable` and calls the `InsertCountyTaxInternal` method, which you will implement next. The code passes the `context` object and each field in the `DataRow` to the method, which creates a `CountyTaxEntity` and adds it to the `context`. Add the following code to the `DataLayer` class to create the `BulkInsertCountyTax` method:



Available for
download on
Wrox.com

```
// Bulk insert county tax entities from a DataTable object.
public static void BulkInsertCountyTax(DataTable dt)
{
    //DEMO: Delete the table first on Bulk Import
    tableClient.DeleteTableIfExists(tableName);
    //DEMO: Create the table if it does not exist.
    tableClient.CreateTableIfNotExist(tableName);

    // Ensure that the data table will be filtered case-insensitively.
    dt.CaseSensitive = false;

    // Bulk saves can only be made on same partition
    // Need to sort by state code, which is the partition key
    dt.DefaultView.Sort = "StateCode ASC";
    string lastStateCode = "";

    // Get data context.
    TableServiceContext context = tableClient.GetDataServiceContext();

    // counter to track batch items
    int i = 0;

    // Create and add each entity.
    foreach (DataRow row in dt.Rows)
    {
        // initialize the partition check
        if (lastStateCode == "") lastStateCode =
            row.Field<string>("StateCode");

        // Batch supports only 100 transactions at a time,
        // so if we hit 100 records for this partition,
        // submit the transaction and keep going.
        // or submit if the State code changes
        // (which is the Table Partition Key)
        if (i == 100 || lastStateCode !=
            row.Field<string>("StateCode"))
        {

```

```

        // Save changes, using the Batch option.
        context.SaveChanges(
            System.Data.Services.Client.SaveChangesOptions.Batch);

        // Reset the counter.
        i = 0;
    }

    // Insert the new entity for this row.
    InsertCountyTaxInternal(
        context,
        row.Field<string>("StateCode"),
        dt.Columns.Contains("CountyCode") ?
            row.Field<string>("CountyCode") : string.Empty,
        dt.Columns.Contains("State") ?
            row.Field<string>("State") : string.Empty,
        dt.Columns.Contains("CountyName") ?
            row.Field<string>("CountyName") : string.Empty,
        dt.Columns.Contains("TotalReturns") ?
            row.Field<string>("TotalReturns") : string.Empty,
        dt.Columns.Contains("TotalExemptions") ?
            row.Field<string>("TotalExemptions") : string.Empty,
        dt.Columns.Contains("AdjustedGrossIncome") ?
            row.Field<string>("AdjustedGrossIncome") : string.Empty,
        dt.Columns.Contains("Wages") ?
            row.Field<string>("Wages") : string.Empty,
        dt.Columns.Contains("DividendIncomes") ?
            row.Field<string>("DividendIncomes") : string.Empty,
        dt.Columns.Contains("InterestIncome") ?
            row.Field<string>("InterestIncome") : string.Empty);

    // Increment the counter.
    i++;

    lastStateCode = row.Field<string>("StateCode");
}

// Save changes, using the Batch option.
context.SaveChanges(SaveChangesOptions.Batch);
}

```

code snippet 076576 Ch03_Code.zip/DataLayer.cs

6. The `InsertCountyTaxInternal` method is a private method to add a `CountyTaxEntity` object to a `TableServiceContext`. This is the method that creates the entity in the Windows Azure table. The code first creates a new `CountyTaxEntity` object. Next, set the `StateCode` as the `PartitionKey`. The other required field is the `RowKey`, which needs to be unique, so it is set as a `Guid` just to keep things simple for this example. After that, it is just a matter of setting each field of the `CountyTaxEntity` object with the matching property passed into the method. To save the entity onto the context, call the `AddObject` method of the context

object, passing the table name and the `CountyTaxEntity` object. Add the following code to the `DataLayer` class:



```
// Insert a new county tax.
private static string InsertCountyTaxInternal(TableServiceContext context,
    string StateCode,
    string CountyCode,
    string State,
    string CountyName,
    string TotalReturns,
    string TotalExemptions,
    string AdjustedGrossIncome,
    string Wages,
    string DividendIncomes,
    string InterestIncome)
{
    // Create the new entity.
    CountyTaxEntity entity = new CountyTaxEntity();

    // Partition key is the State Abbreviation.
    entity.PartitionKey = State.ToUpper();

    // Row key is a GUID
    entity.RowKey = Guid.NewGuid().ToString();

    // Populate the other properties.
    entity.StateCode = StateCode;
    entity.CountyCode = CountyCode;
    entity.State = State;
    entity.CountyName = CountyName;
    entity.TotalReturns = TotalReturns;
    entity.TotalExemptions = TotalExemptions;
    entity.AdjustedGrossIncome = AdjustedGrossIncome;
    entity.Wages = Wages;
    entity.DividendIncomes = DividendIncomes;
    entity.InterestIncome = InterestIncome;
    // Add the entity.
    context.AddObject(tableName, entity);

    return entity.RowKey;
}
```

code snippet 076576 Ch03_Code.zip/DataLayer.cs

7. There is one last thing to do before you run this code. You need to get a connection to the Windows Azure storage using your storage account information. In this case, because you are running this in the local storage emulator, you don't need a real Windows Azure storage account just yet. You want this code to run once when the service starts up. The `WebRole.cs` file is created for you when you created the project. The `WebRole` class contains an `OnStart` method that runs when the service starts. Replace the following code in the `WebRole.cs` file:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
\

namespace CountyTaxService
{
    public class WebRole : RoleEntryPoint
    {
        public override bool OnStart()
        {
            // To enable the AzureLocalStorageTraceListner,
            // uncomment relevant section in the web.config
            DiagnosticMonitorConfiguration diagnosticConfig =
                DiagnosticMonitor.GetDefaultInitialConfiguration();
            diagnosticConfig.Directories.ScheduledTransferPeriod =
                TimeSpan.FromMinutes(1);
            diagnosticConfig.Directories.DataSources.Add(
                AzureLocalStorageTraceListener.GetLogDirectory());

            // For information on handling configuration changes
            // see the MSDN topic at
            // http://go.microsoft.com/fwlink/?LinkId=166357.

            // Get connection string and table name
            // from the role's configuration settings.
            string connectionString =
                RoleEnvironment.GetConfigurationSettingValue(
                    "StorageConnectionString");
            string tableName =
                RoleEnvironment.GetConfigurationSettingValue(
                    "TableName");

            CloudStorageAccount storageAccount =
                CloudStorageAccount.Parse(connectionString);

            CloudTableClient tableClient =
                storageAccount.CreateCloudTableClient();

            // Create the table if it does not exist.
            tableClient.CreateTableIfNotExist(tableName);

            return base.OnStart();
        }
    }
}
```

code snippet 076576 Ch03_Code.zip/WebRole.cs

8. The table connection information is stored in a couple of Azure service configuration files, `ServiceConfiguration.cscfg` and `ServiceDefinition.csdef`. In the

ServiceConfiguration.cscfg file, set the connection string to UseDevelopmentStorage=true. Add the following code to the ServiceConfiguration.cscfg file in the CountyTax project:



```
<?xml version="1.0" encoding="utf-8"?>
<ServiceConfiguration serviceName="CountyTax"
  xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration"
    osFamily="1" osVersion="*">
  <Role name="CountyTaxService">
    <Instances count="1" />
    <ConfigurationSettings>

      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
        value="UseDevelopmentStorage=true" />

      <Setting name="StorageConnectionString"
        value="UseDevelopmentStorage=true" />

      <Setting name="TableName"
        value="CountyTax" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

code snippet 076576 Ch03_Code.zip/ServiceConfiguration.cscfg

9. Before you can use custom properties in the ServiceConfiguration.cscfg file, you need to define them in the ServiceDefinition.csdef file. Add the following code to define the StorageConnectionString and TableName settings:



```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="CountyTax"
  xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="CountyTaxService">
    <Sites>
      <Site name="Web">
        <Bindings>
          <Binding name="Endpoint1" endpointName="Endpoint1" />
        </Bindings>
      </Site>
    </Sites>
    <Endpoints>
      <InputEndpoint name="Endpoint1" protocol="http" port="8080" />
    </Endpoints>
    <Imports>
      <Import moduleName="Diagnostics" />
    </Imports>
    <ConfigurationSettings>
      <Setting name="StorageConnectionString" />
      <Setting name="TableName" />
    </ConfigurationSettings>
    <LocalResources>
      <LocalStorage name="CountyTaxService.svclog"
        sizeInMB="1000" cleanOnRoleRecycle="false" />
    </LocalResources>
```

```
</WebRole>
</ServiceDefinition>
```

code snippet 076576 Ch03_Code.zip/ServiceDefinition.csdef

10. Set the CountyTax project as the startup project and the `Import.aspx` as the startup page, and then press F5 to run the Windows Azure web role. When the Import page loads, click the Browse button, select the `.csv` file that you created earlier in the chapter, and then click the Import button to load the `.csv` file into the Windows Azure table.

You can view the storage emulator using Visual Studio's Server Explorer, as shown in Figure 3-7.

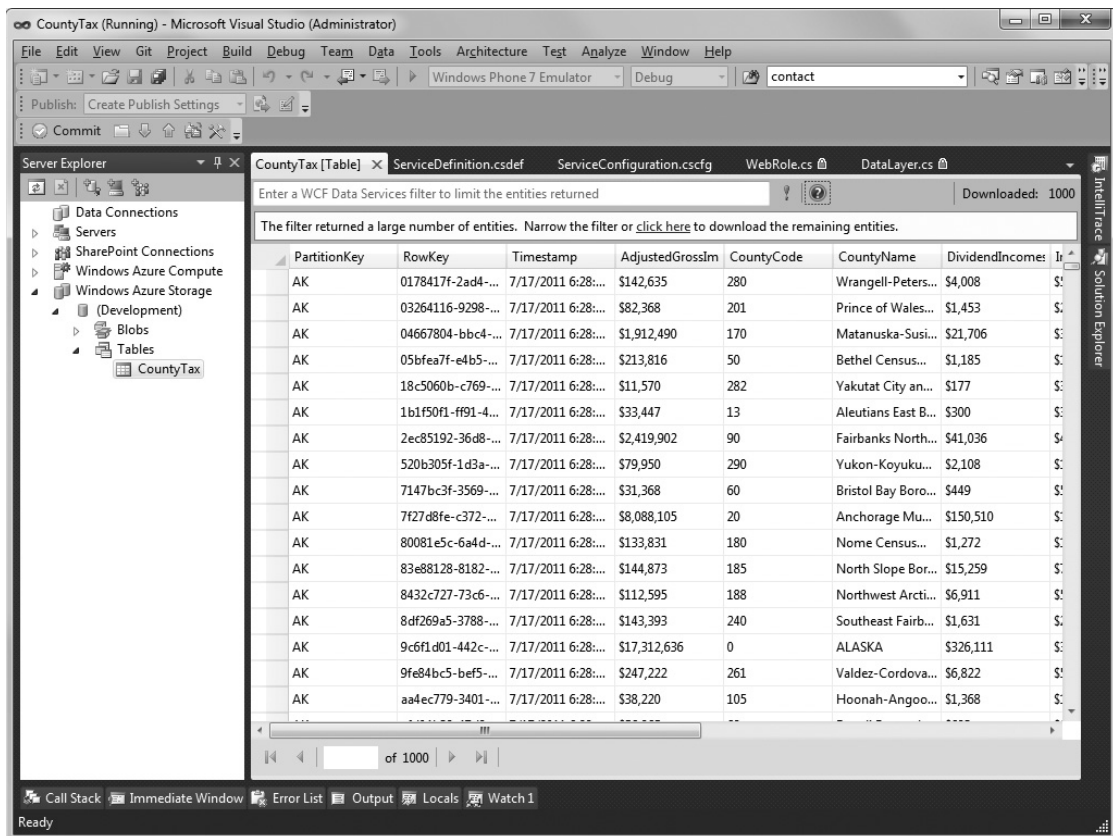


FIGURE 3-7

You now have your data imported into a newly created Windows Azure table. The next step is to create a Windows Azure WCF service endpoint for the SharePoint BCS to connect to.

CREATING A WCF SERVICE ENDPOINT

The second part of this example creates a Windows Azure service endpoint for the SharePoint BCS client to interact with. In this example, you have already created a Windows Azure WCF service project called `CountyTaxService`. Up to this point, you have used it to host the `Import.aspx` web page and the `DataLayer` code to import the table data from the `.csv` file. Now it is time to add the actual service.

Building a Windows Azure Service

To add the service, add a new WCF service project item called `CountyTaxService` to the `CountyTaxService` project. This will create the following three files:

- `CountyTaxService.svc` — The actual endpoint for the service, which in this case would be accessed using `http://127.0.0.1:8080`
- `CountyTaxService.svc.cs` — The code-behind file for the `CountyTaxService.svc` file. This is where the actual service code is placed.
- `ICountyTaxService.cs` — The service-contract interface.

You'll implement the service-contract interface first. You know that the service is going to be used to connect SharePoint BCS services with your Windows Azure table, so you can create the interface with this in mind. Add the following code to the `ICountyTaxService.cs` file:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace CountyTaxService
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change
    // the interface name "ICountyTaxService" in both code and config file
    // together.
    [ServiceContract]
    public interface ICountyTaxService
    {
        [OperationContract]
        CountyTaxEntity ReadItem(string RowKey);

        [OperationContract]
        List<CountyTaxEntity> ReadList();

        [OperationContract]
        string Create(CountyTaxEntity countyTaxEntity);
    }
}
```

```
[OperationContract]
string Update(CountyTaxEntity countyTaxEntity);

[OperationContract]
bool Delete(string RowKey);
}
}
```

code snippet 076576 Ch03_Code.zip/ICountyTaxService.cs

For the BCS connection, SharePoint just needs a few methods to handle the CRUD operations. The ReadItem, ReadList, Create, Update, and Delete methods are defined in the interface, which you can implement in the CountyTaxService.svc.cs file. This class implements the ICountyTaxService interface. The code in this class is very minimal as well, because all the actual code is implemented in the DataLayer class.



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace CountyTaxService
{
    public class CountyTaxService : ICountyTaxService
    {

        public CountyTaxEntity ReadItem(string RowKey)
        {
            CountyTaxEntity ct =
                DataLayer.GetCountyTaxByRowKey(RowKey);

            return ct;
        }

        public List<CountyTaxEntity> ReadList()
        {
            List<CountyTaxEntity> CountyTaxes =
                DataLayer.GetAllCountyTax();

            return CountyTaxes;
        }

        public string Create(CountyTaxEntity countyTaxEntity)
        {
            CountyTaxEntity ct = countyTaxEntity;

            string rowkey = DataLayer.InsertCountyTax(
                ct.StateCode,
                ct.CountyCode,
                ct.State,
                ct.CountyName,
```

```

        ct.TotalReturns,
        ct.TotalExemptions,
        ct.AdjustedGrossIncome,
        ct.Wages,
        ct.DividendIncomes,
        ct.InterestIncome);

    return rowkey;
}

public string Update(CountyTaxEntity countyTaxEntity)
{
    CountyTaxEntity ct = countyTaxEntity;

    string rowkey = DataLayer.UpdateCountyTax(
        ct.RowKey,
        ct.StateCode,
        ct.CountyCode,
        ct.State,
        ct.CountyName,
        ct.TotalReturns,
        ct.TotalExemptions,
        ct.AdjustedGrossIncome,
        ct.Wages,
        ct.DividendIncomes,
        ct.InterestIncome);

    return rowkey;
}

public bool Delete(string RowKey)
{
    bool succeeded =
        DataLayer.DeleteCountyTaxByRowKey(RowKey);

    return succeeded;
}
}
}

```

code snippet 076576 Ch03_Code.zip/CountyTaxService.svc.cs

Next, you will implement the code in the `DataLayer` class, which uses a Windows Azure storage client to read, write, update, and delete the table data.

Accessing Table Storage Data

Although the Windows Azure Table service has a REST interface to access the data, the Windows Azure team has created a client proxy to handle all the REST details for you. This makes writing the CRUD code very easy. As you can see in the following code, most of the work is handled using the `CreateQuery` method of the `context` object. The `CreateQuery` method uses a LINQ query to access the data. Note that the following code shows two different ways to use the LINQ syntax.

Both examples are valid; it is really just a matter of preference. Replace the following code in the `DataLayer` class:



Available for
download on
Wrox.com

```
private static string connectionString;
private static string tableName;
private static CloudStorageAccount storageAccount;
private static CloudTableClient tableClient;

static DataLayer()
{
    // Get connection string and table name from settings.
    connectionString =
        RoleEnvironment.GetConfigurationSettingValue(
            "StorageConnectionString");
    tableName =
        RoleEnvironment.GetConfigurationSettingValue(
            "TableName");

    // Reference storage account from connection string.
    storageAccount = CloudStorageAccount.Parse(connectionString);

    // Create Table service client.
    tableClient = storageAccount.CreateCloudTableClient();
}

// Get county tax filtered by state.
public static List<CountyTaxEntity> GetCountyTaxByState(string state)
{
    IQueryable<CountyTaxEntity> query =
        tableClient.GetDataServiceContext().
            CreateQuery<CountyTaxEntity>(tableName);
    if (state != null)
    {
        query = query.Where(c => c.PartitionKey == state.ToUpper());
    }
    return query.AsTableServiceQuery().ToList();
}

// Get county tax filtered by state.
public static List<CountyTaxEntity> GetAllCountyTax()
{
    IQueryable<CountyTaxEntity> query =
        tableClient.GetDataServiceContext().
            CreateQuery<CountyTaxEntity>(tableName);

    return query.AsTableServiceQuery().ToList();
}

// Get CountyTax using RowKey.
public static CountyTaxEntity GetCountyTaxByRowKey(string RowKey)
{
    // Get data context.
    TableServiceContext context = tableClient.GetDataServiceContext();

    return GetCountyTaxByRowKey(RowKey, context);
}
```

```

    }

    // Get CountyTax using RowKey.
    public static CountyTaxEntity GetCountyTaxByRowKey(
        string RowKey, TableServiceContext context)
    {

        CountyTaxEntity countyTax =
            (from ct in context.CreateQuery<CountyTaxEntity>(tableName)
             where
                 ct.RowKey == RowKey
             select ct).Single();

        return countyTax;
    }

    // Create new context and get CountyTax.
    public static CountyTaxEntity GetCountyTax(
        string State, string CountyCode)
    {
        // Get data context.
        TableServiceContext context = tableClient.GetDataServiceContext();

        return GetCountyTax(State, CountyCode, context);
    }

    // Get CountyTax using existing context.
    public static CountyTaxEntity GetCountyTax(
        string State, string CountyCode, TableServiceContext context)
    {
        CountyTaxEntity countyTax =
            (from ct in context.CreateQuery<CountyTaxEntity>(tableName)
             where
                 ct.PartitionKey == State &&
                 ct.CountyCode == CountyCode
             select ct).Single();

        return countyTax;
    }

    // Update contact data.
    public static string UpdateCountyTax(
        string RowKey,
        string StateCode,
        string CountyCode,
        string State,
        string CountyName,
        string TotalReturns,
        string TotalExemptions,
        string AdjustedGrossIncome,
        string Wages,
        string DividendIncomes,
        string InterestIncome)
    {

```

```
// Get data context.
TableServiceContext context = tableClient.GetDataServiceContext();

// Set updated values
CountyTaxEntity entity = GetCountyTax(State, CountyCode, context);

// Populate the other properties.
entity.StateCode = StateCode;
entity.CountyCode = CountyCode;
entity.State = State;
entity.CountyName = CountyName;
entity.TotalReturns = TotalReturns;
entity.TotalExemptions = TotalExemptions;
entity.AdjustedGrossIncome = AdjustedGrossIncome;
entity.Wages = Wages;
entity.DividendIncomes = DividendIncomes;
entity.InterestIncome = InterestIncome;

// Update the object.
context.UpdateObject(entity);

// Write changes to the Table service.
context.SaveChanges();

return entity.RowKey;
}

// Insert a new contact.
public static string InsertCountyTax(
    string StateCode,
    string CountyCode,
    string State,
    string CountyName,
    string TotalReturns,
    string TotalExemptions,
    string AdjustedGrossIncome,
    string Wages,
    string DividendIncomes,
    string InterestIncome)
{
    // Get data context.
    TableServiceContext context = tableClient.GetDataServiceContext();

    // Insert the new entity.
    string rowkey = InsertCountyTaxInternal(context,
        StateCode,
        CountyCode,
        State,
        CountyName,
        TotalReturns,
        TotalExemptions,
        AdjustedGrossIncome,
        Wages,
```

```

        DividendIncomes,
        InterestIncome);

    // Save changes to the service.
    context.SaveChanges();

    return rowkey;
}

// Insert a new county tax.
private static string InsertCountyTaxInternal(
    TableServiceContext context,
    string StateCode,
    string CountyCode,
    string State,
    string CountyName,
    string TotalReturns,
    string TotalExemptions,
    string AdjustedGrossIncome,
    string Wages,
    string DividendIncomes,
    string InterestIncome)
{
    // Create the new entity.
    CountyTaxEntity entity = new CountyTaxEntity();

    // Partition key is the State Abbreviation.
    entity.PartitionKey = State.ToUpper();

    // Row key is a GUID
    entity.RowKey = Guid.NewGuid().ToString();

    // Populate the other properties.
    entity.StateCode = StateCode;
    entity.CountyCode = CountyCode;
    entity.State = State;
    entity.CountyName = CountyName;
    entity.TotalReturns = TotalReturns;
    entity.TotalExemptions = TotalExemptions;
    entity.AdjustedGrossIncome = AdjustedGrossIncome;
    entity.Wages = Wages;
    entity.DividendIncomes = DividendIncomes;
    entity.InterestIncome = InterestIncome;
    // Add the entity.
    context.AddObject(tableName, entity);

    return entity.RowKey;
}

// Delete a county tax.
public static bool DeleteCountyTaxByRowKey(string RowKey)
{
    // Get data county tax.
    TableServiceContext context =
        tableClient.GetDataServiceContext();

```

```

// Retrieve county tax.
CountyTaxEntity entity =
    GetCountyTaxByRowKey(RowKey);

// Delete the entity.
context.DeleteObject(entity);

// Save changes to the service.
context.SaveChanges();

return true;
}

```

code snippet 076576 Ch03_Code.zip/DataLayer.cs

In Visual Studio, set the start page to `CountyTaxService.svc`. Press F5 to start your Windows Azure service running on the local compute emulator. This will launch `CountyTaxService.svc` in the browser, as shown in Figure 3-8, and attach the Visual Studio debugger.

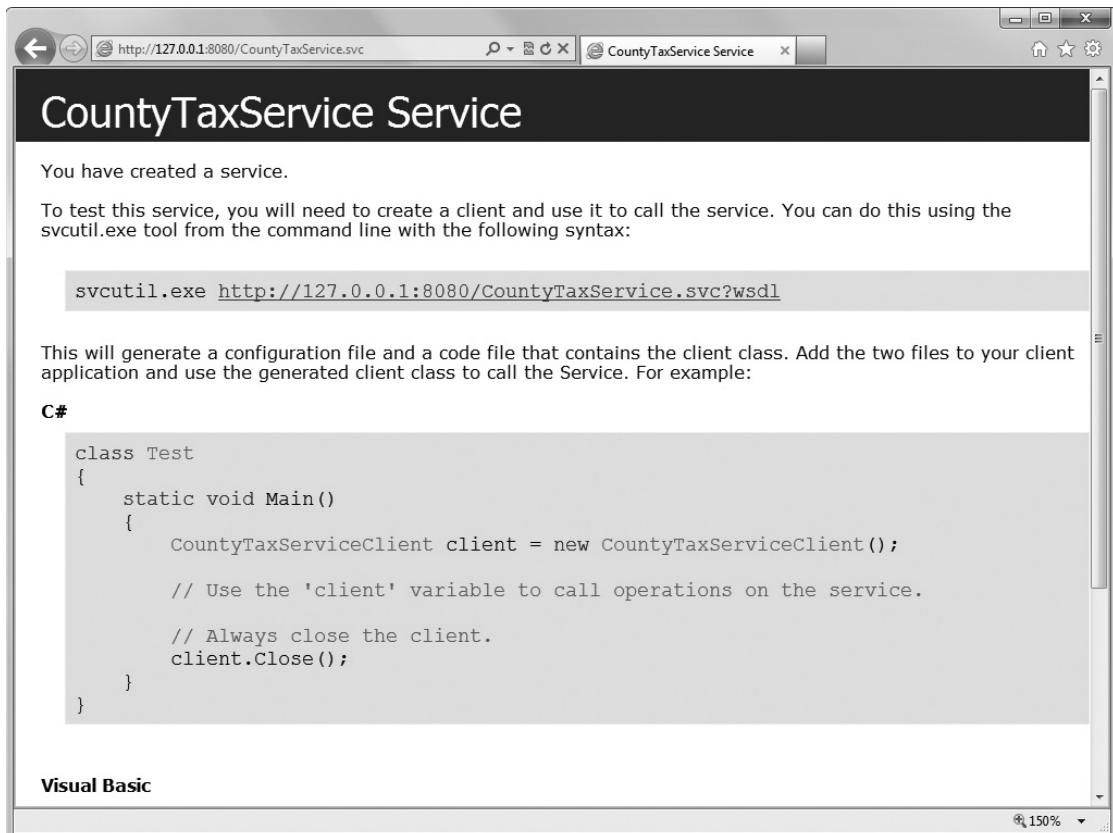


FIGURE 3-8

You have now created all the Windows Azure parts of the solution: a Windows Azure table to store the data, as well as a Windows Azure WCF service to access the data. The last step is to create a SharePoint BCS external content type using SharePoint 2010 Designer. This will enable you to connect to the Windows Azure table as though it were a native SharePoint list.

CREATING A SHAREPOINT BCS EXTERNAL CONTENT TYPE

SharePoint has the ability to connect to external data in systems such as SQL Server using BCS. BCS creates a mapping to the external data, called an *external content type* (ECT). You can create an ECT in one of two ways: programmatically in Visual Studio or declaratively in SharePoint Designer (SPD). In this example, you will use SPD to create the external content type declaratively without writing any code.

Consuming a Windows Azure Service Using BCS

Once you have the Windows Azure service created, consuming that service using SPD is easy. In fact, at this point there is nothing special about connecting to the Windows Azure service that is different from connecting to any other service. So, if you are familiar with BCS and SPD, this should be straightforward.

1. Open SPD and connect to the SharePoint site in which you want to create the ECT.
2. Click the External Content Types item in the left navigation pane of SPD. In the right pane, you will see a list of existing ECTs. Create a new ECT by clicking the External Content Type button in the New ribbon group.
3. Set the Name and Display Name properties to **AzureCountyTax**. You can leave all the other properties at their default values, as shown in Figure 3-9.

External Content Type Information	
Key information about this external content type.	
Name	AzureCountyTax
Display Name	AzureCountyTax
Namespace	http://intranet.contoso.com
Version	1.0.0.0
Identifiers	RowKey(String)
Office Item Type	Generic List
Offline Sync for external list	Enabled
External System	http://demo2010a:8080/CountyTaxService.svc?wsdl

FIGURE 3-9

4. Now you need to map the CRUD operations of the ECT to the WCF service you created in the previous section. Click the hyperlink labeled Click Here to discover external data sources and define operations. The Operations Designer page will open. The Operations Designer is where you manage the ECT mapping.
5. Click the Add Connection button and choose WCF Service from the External Data Source Type Selection dialog.
6. In the WCF Connection dialog, set the Service Metadata URL to `http://demo2010a:8080/CountyTaxService.svc?wsdl`. Remember, this is the endpoint of your Windows Azure

service. In this case, it is the local server name where SharePoint is running, which is forwarded to the host machine at `http:127.0.0.1:8080` through the SSH tunnel.

- 7. Set the Service Endpoint URL to `http://demo2010:8080/CountyTaxService.svc`, and the Metadata Connection mode to WSDL.
- 8. Set the Proxy Namespace to `BCSServiceProxy`. Your settings should look like Figure 3-10.
- 9. Click OK to connect to the Windows Azure service. You can verify that the connection was created when you see the Web Method operations in the Data Source Explorer tab, similar to Figure 3-11.
- 10. Map each BCS operation to the corresponding service operation from the context menu of each service operation. As shown in Figure 3-12, the following BCS operations are available: Read Item, Read List, Create, Update, and Delete. These match the names of the operations you created in the WCF service previously, making it easy to pick the correct one.

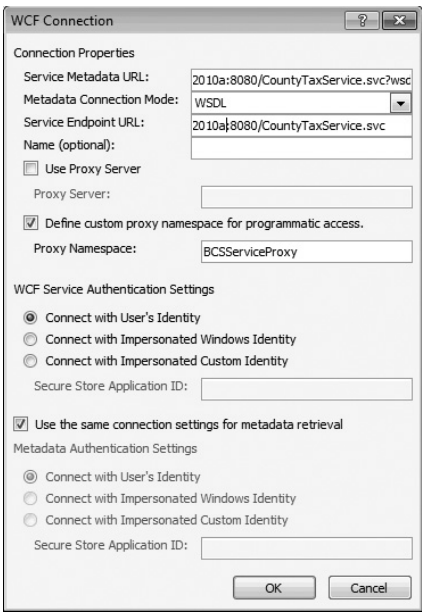


FIGURE 3-10

The following steps describe mapping the Update operation.

- 11. Right-click on Update and select New Update Operation from the context menu. The Operation Mapping wizard opens. The first page is the Operation Properties page (see Figure 3-13). You can click Next without making any changes.
- 12. The second step in the wizard is the Input Parameters Configuration page. This is where you map each input and output parameter between the BCS operations and the WCF service operations. This example is for the update operation, so select all the data source



FIGURE 3-11

elements. The only change you need to make is to select the RowKey element and check the Map to Identifier checkbox. Set the Identifier drop-down to RowKey. Ensure your settings match those in Figure 3-14, and then click Finish to create the operation.

Once you have finished mapping all the operations, you will see a page in SPD similar to Figure 3-15.

13. The last step is to create an external list from the ECT you just created. SPD makes this very easy to do with a single button click. Click the Create Lists & Forms button on the ribbon. In the Create List and Form for AzureCountyTax dialog, choose Create New External List and name the new list **AzureCountyTax**. This will create a list like the one shown in Figure 3-16.

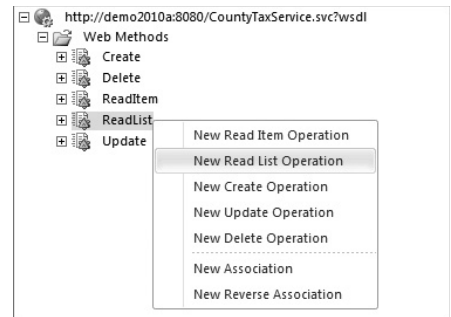


FIGURE 3-12

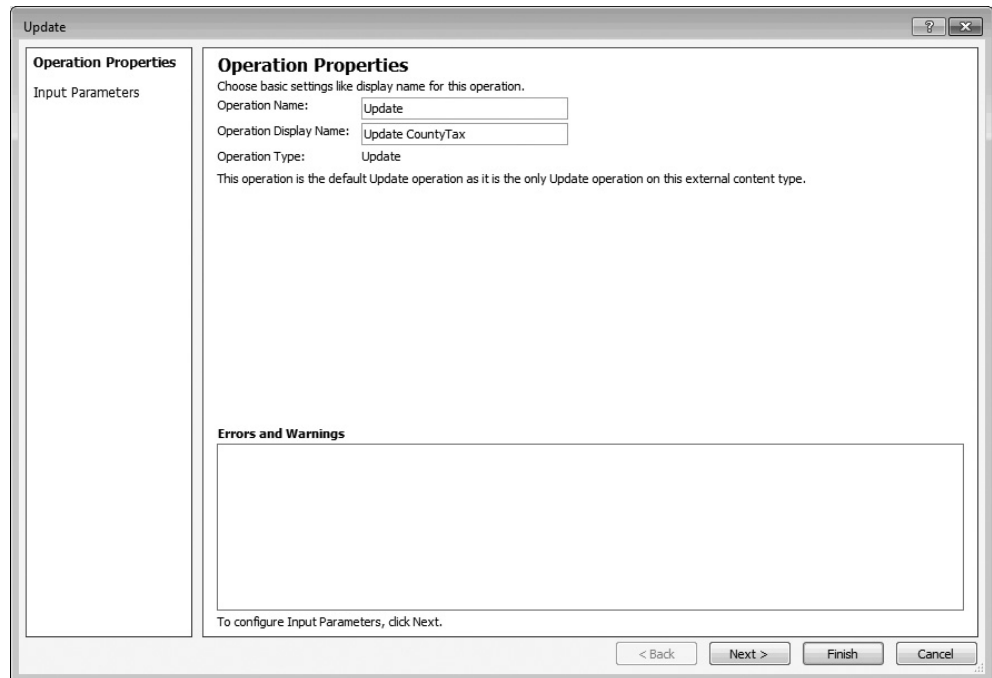


FIGURE 3-13

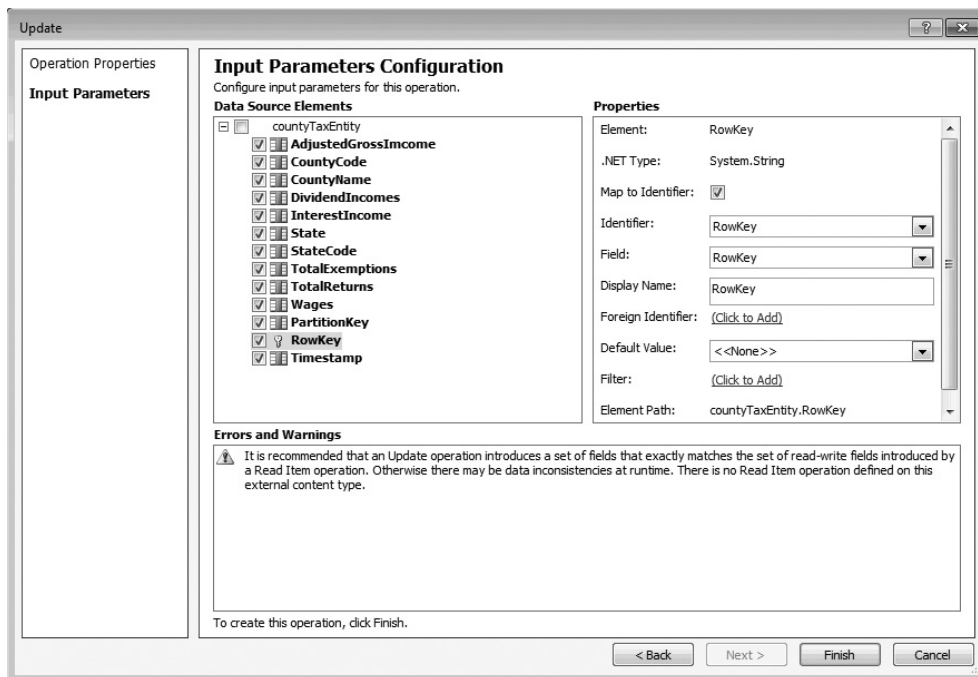


FIGURE 3-14

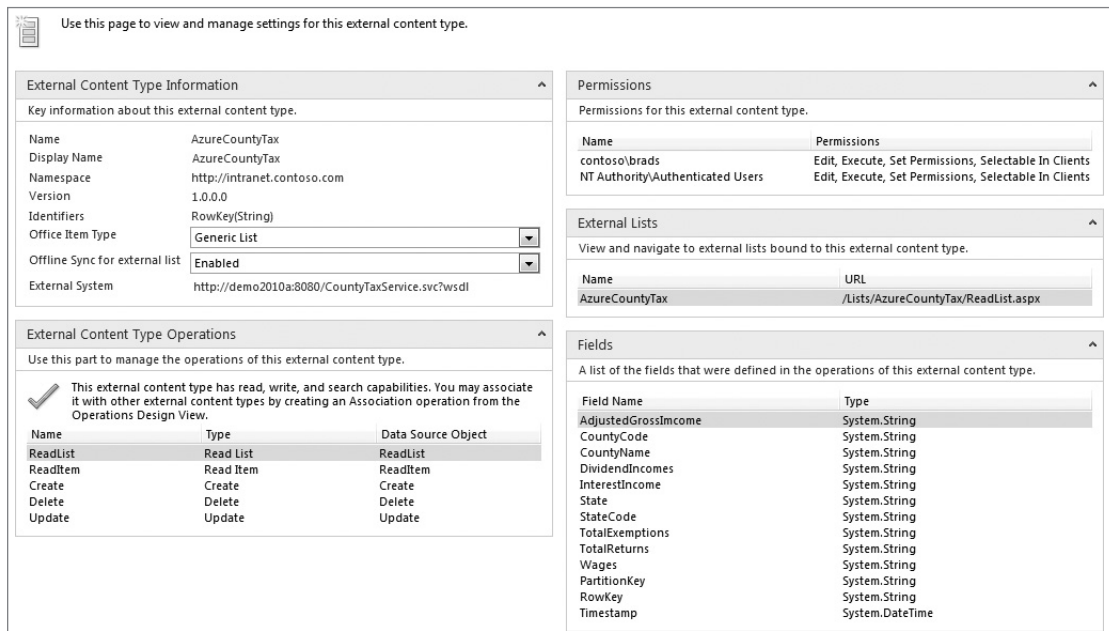


FIGURE 3-15

	AdjustedGrossIncome	CountyCode	CountyName	DividendIncomes	InterestIncome	State	StateCode	TotalExem
FAST Search Demo	\$1,002,480	145	Newton County	\$18,983	\$31,844	MO	29	52193
Content Management								
Customer Relationship Management	\$1,003,950	33	Lake County	\$18,515	\$41,934	CA	6	53383
Strategy Consulting								
Libraries	\$1,004,561	5	Ashland County	\$16,329	\$25,193	OH	39	48589
Site Pages								
Shared Documents								
DocDLLibrary	\$1,004,594	69	Franklin County	\$7,215	\$23,002	NC	37	52499
Declare Records Demo								
Lists	\$1,006,228	103	Effingham County	\$6,253	\$15,283	GA	13	47029
Calendar								
Tasks								
Projects	\$1,008,913	177	Stephenson County	\$22,532	\$41,985	IL	17	45536
Employees								
ClientOMContacts	\$1,012,011	169	Winona County	\$28,367	\$32,329	MN	27	42335
Airports								
CountyTax								

FIGURE 3-16

All the operations that you expect to work will work in this external list. For example, creating, updating, and deleting records works just like it does for any native SharePoint list.

SUMMARY

You have seen how you can leverage Windows Azure services in your SharePoint solutions using BCS. This pattern enables SharePoint developers to create hybrid applications that can offline work to Windows Azure. Using BCS as the bridge between Windows Azure and SharePoint ensures that the SharePoint and Windows Azure are loosely coupled though the declarative operation mapping of BCS so that if either side of the solution changes, you do not need to write, test, and deploy code to handle the changes. This makes for a very flexible solution. Finally, using BCS ensures that the Windows Azure data appears integrated in SharePoint as a first-class citizen. This configuration makes it very easy for SharePoint and Office power users and developers to take advantage of the power of the Windows Azure platform from an environment in Office and SharePoint that they are familiar with.

ADDITIONAL REFERENCES

Following are two additional references that you might find useful:

- **Azure Developer Center** — msdn.microsoft.com/WindowsAzure
- **Windows Azure SDK and Windows Azure Tools for Microsoft Visual Studio** — www.microsoft.com/downloads/en/details.aspx?FamilyID=7a1089b6-4050-4307-86c4-9dadaa5ed018&displaylang=en

4

Creating an Aggregated Solution Using Cloud-Based Data

WHAT'S IN THIS CHAPTER?

- Understanding Yahoo Pipes mashups
- Creating advanced pipes using Yahoo Query Language
- Decoupling SharePoint web parts from the data source

One of the biggest challenges (and biggest opportunities) for SharePoint developers is leveraging all the data APIs of the Internet. The Internet is a vast source of data. The problem is easily finding, querying, and manipulating the data you find into the data you need. And once you figure out how to do this for one data source, often you need to use a different process or API for other data sources. Furthermore, data sources on the Internet are very dynamic compared to traditional in-house data sources; therefore, your application needs to be flexible enough to handle changing data sources. This means that every time the data source changes, you need to edit, compile, and deploy your SharePoint applications. The solution is to decouple the Internet data sources from your SharePoint applications. You want to be able to combine the data sources in a declarative tool outside of your application. These types of applications are called *composite applications* or *mashups*. Many tools to create mashups are available, and one of the best and oldest is called *Yahoo Pipes*.

OVERVIEW OF MASHUPS

Mashups enable developers to combine (mash up) data from two or more different data sources to create a new service. The most common examples are mashups that include mapping data. For example, you can mash up a Bing map with crime data from the Windows Azure Marketplace, and then you can add in current home prices or houses for sale. Enterprise mashups employ the same concept inside of the firewall for corporate data.

One of the most prevalent sources of data on the Internet is in the form of Really Simple Syndication (RSS) feeds. RSS feeds are a common way to publish frequently changing data, such as news and blogs. Owners of the content have a standard way to syndicate their updates and data. Consumers can easily subscribe to this data using software called *feed readers* or *aggregators*. These tools enable you to read many feeds and focus on what is new.



FIGURE 4-1

Finding RSS feeds on the Internet is easy when you know what to look for. Most sites publish their data as an RSS feed, and surface the link to the feed using the RSS icon, shown in Figure 4-1.



One way to find RSS feeds is to use a site dedicated to aggregating RSS feeds, such as Feedzilla (www.feedzilla.com/rss.asp).

CREATING MASHUPS USING YAHOO PIPES

Once you have the feeds that you want to consume, you can mash up the data into your SharePoint application. There are a couple of different approaches to do this. The first, and probably the more common, approach is to call each RSS feed or data service from your application in code. Basically, you transform the data into the correct form and shape that your application needs to display all the data together.

The other approach, which is the foundation of this chapter, is to combine and format the data into the correct shape outside of your application. There are declarative tools that enable you to visually shape and combine data from multiple sources. The one you will learn about here is called Yahoo Pipes (<http://pipes.yahoo.com>).



Once you have mastered Yahoo Pipes, consider using a more advanced tool, called the Yahoo Query Language (YQL). YQL helps you normalize Internet data into a common format and query syntax. You can use YQL by itself, but it also works as an extension to Yahoo Pipes. Later in this chapter you will see how to use YQL to extend your SharePoint solutions and pipes.

Yahoo Pipes is a *composition tool*, commonly referred to as a *mashup tool*, that enables developers and power users to combine data from around the Internet. Yahoo Pipes also makes it easy to transform and manipulate the data using a visual editor.

In this section, you will learn how to use existing pipes created by other users and how to create your own pipes. Then you will see how to use this pipe data in your SharePoint web parts. This chapter does not attempt to cover all the functionality of Yahoo Pipes; rather, it is meant to demonstrate how you can pull data from multiple sources into your SharePoint solutions.

Using the Pipes Editor

The Yahoo Pipes editor is a web-based visual editor that enables you to wire up widgets to pipe the data through a process in which data enters on one end of the pipe and exits on the other end. Let's explore the editor by creating a very simple Hello World pipe that consumes tech news from Yahoo.

Start by browsing to the Yahoo Pipes page at <http://pipes.yahoo.com>. To use the Pipes editor, you need to be logged in. Click the Sign In link on the top-right corner of the page. You can log in using your Yahoo account or Facebook or Google accounts. Once you are logged in, click Create a Pipe from the top menu. This will open the Pipes editor, shown in Figure 4-2.

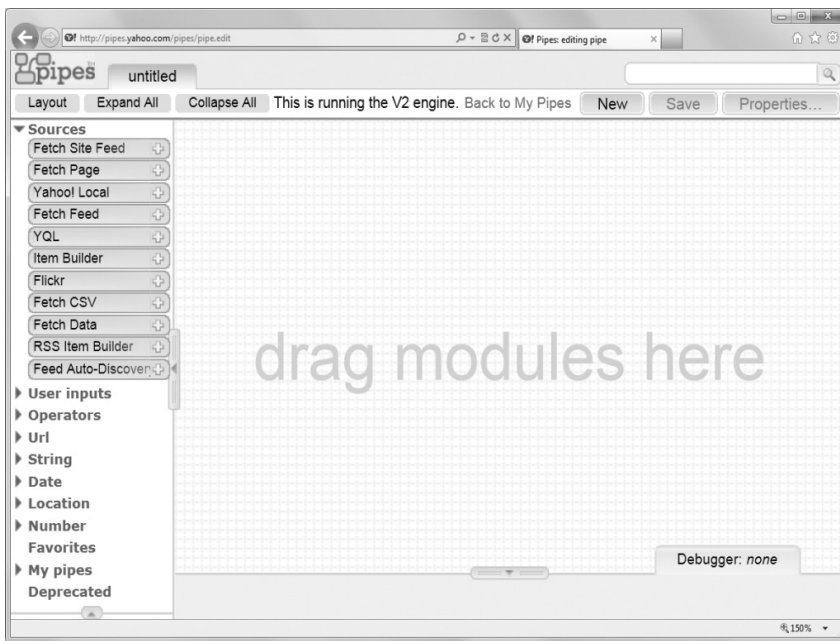


FIGURE 4-2

On the left side of the editor are the available data sources. There are a number of interesting sources, such as Fetch Feed, which will return an RSS feed, and Fetch Page, which will return the source of a page as a string. You can then use expressions to parse out just the elements you are interested in. You may also notice there is a YQL source. The YQL source enables you to consume data passed from a YQL query. You will learn more about YQL later in this chapter. For now, you will create a simple pipe that consumes a single feed but does no other manipulations.

The RSS feed that you will use is the Tech News feed from Yahoo News (<http://news.yahoo.com/rss/tech>). If you enter this path directly in the browser, you can see the raw RSS feed (edited here to two items for brevity):

```

<?xml version="1.0" encoding="utf-8"?>
<rss xmlns:media="http://search.yahoo.com/mrss/"
    xmlns:ynews="http://news.yahoo.com/rss/"
    version="2.0">
  <channel>
    <title>Tech News Headlines - Yahoo! News</title>
    <link>http://news.yahoo.com/tech/</link>
    <description>Get the latest Tech news headlines from Yahoo! News. Find
      breaking Tech news, including analysis and opinion on top Tech
      stories.</description>
    <language>en-US</language>
    <copyright>Copyright (c) 2011 Yahoo! Inc. All rights reserved</copyright>
    <pubDate>2011-08-07T14:48:40-04:00</pubDate>
    <ttl>5</ttl>
    <image>
      <title>Tech News Headlines - Yahoo! News</title>
      <link>http://news.yahoo.com/tech/</link>
      <url>http://l.yimg.com/a/i/us/nws/th/main_142c.gif</url>
    </image>
    <item>
      <title>Your smartphone: a new frontier for hackers</title>
      <description>&lt;p&gt;&lt;a href="http://news.yahoo.com/smartphone-
frontier-hackers-120230424.html"&gt;&lt;img
src="http://l.yimg.com/bt/api/res/1.2/15sR9jZsOvNmrwuB.T.13Q--
/YXBwaWQ9eW5ld3M7Zmk9ZmlsbDToPTg2O3E9ODU7dz0xMzA-
/http://l.yimg.com/os/en_us/News/ap_webfeeds/20d4458balf62c11f50e6a7067002a21.j
pg" alt="photo" align="left" title="FILE - In this Jan. 5, 2011 file photo, a
person operates their iPhone in New York. Security experts say attacks on
smartphones are growing fast â€" and attackers are becoming smarter about
developing new techniques. (AP Photo/Frank Franklin II, File)" border="0"
/&gt;&lt;/a&gt; Hackers are out to stymie your smartphone.&lt;/p&gt;&lt;br
clear="all"/&gt;</description>
      <link>http://news.yahoo.com/smartphone-frontier-hackers-
120230424.html</link>
      <pubDate>2011-08-07T12:50:21Z</pubDate>
      <source>AP</source>
      <guid isPermaLink="false">smartphone-frontier-hackers-120230424</guid>
      <media:content
url="http://l.yimg.com/bt/api/res/1.2/15sR9jZsOvNmrwuB.T.13Q--
/YXBwaWQ9eW5ld3M7Zmk9ZmlsbDToPTg2O3E9ODU7dz0xMzA-
/http://l.yimg.com/os/en_us/News/ap_webfeeds/20d4458balf62c11f50e6a7067002a21.j
pg"
          type="image/jpeg"
          width="130"
          height="86"></media:content>
      <media:text type="html">&lt;p&gt;&lt;a
href="http://news.yahoo.com/smartphone-frontier-hackers-
120230424.html"&gt;&lt;img
src="http://l.yimg.com/bt/api/res/1.2/15sR9jZsOvNmrwuB.T.13Q--
/YXBwaWQ9eW5ld3M7Zmk9ZmlsbDToPTg2O3E9ODU7dz0xMzA-
/http://l.yimg.com/os/en_us/News/ap_webfeeds/20d4458balf62c11f50e6a7067002a21.j
pg" alt="photo" align="left" title="FILE - In this Jan. 5, 2011 file photo, a
person operates their iPhone in New York. Security experts say attacks on
smartphones are growing fast â€" and attackers are becoming smarter about
developing new techniques. (AP Photo/Frank Franklin II, File)" border="0"

```

```

/&gt;&lt;/a&gt; Hackers are out to stymie your smartphone.&lt;/p&gt;&lt;br
clear="all"/&gt;</media:text>
  <media:credit role="publishing company"></media:credit>
</item>
<item>
  <title>Google, Microsoft goes public with patent spat</title>
  <description>Tech heavyweights Microsoft and Google are acting like a
couple of feuding starlets in a public online spat over "wait for it"
patents.</description>
  <link>http://news.yahoo.com/google-microsoft-goes-public-patent-spat-
195043412.html</link>
  <pubDate>2011-08-05T21:14:14Z</pubDate>
  <source>AP</source>
  <guid isPermaLink="false">google-microsoft-goes-public-patent-spat-
195043412</guid>
</item>
</channel>
</rss>

```

Creating Pipes

To create a pipe using the Tech News feed, perform the following steps:

1. Drag the Fetch Feed item to the design surface from the source on the left.
2. On the Fetch Feed item you added, set the URL to `http://news.yahoo.com/rss/tech`.
3. To wire the Fetch Feed item to the Pipe Output item, drag the small circle under the Fetch Feed to the small circle above the Pipe Output item. This is how you wire together your pipe.
4. To run the pipe in the editor, click the Pipe Output item to select it. You will see the output in the debug panel at the bottom of the editor, as shown in Figure 4-3.
This has not changed the data in any way. The debug window shows the values of the RSS feed — for example, the title and description of the feed item.
5. Save the pipe by clicking the Save button in the designer. Name the pipe **Yahoo Tech News**. Once you have saved the pipe, it will appear in your list of pipes on the left under My Pipes.

Now that you have completed creating your first Yahoo pipe, it is time to run it. Click the Run Pipe link at the top of the designer to run your pipe. This will launch your pipe in the information view, as shown in Figure 4-4.

The information view contains a lot of information about your pipe. On the left side of the page is information about the pipe itself, such as who created it and which data sources, tags, and modules it uses. The center pane includes the title of the pipe and an HTML rendering of the running pipe. In this case, your pipe returns tech news items, which also include a thumbnail image link. The information page renders your pipe as a carousel of images and shows the details when you hover over an item. However, the point of creating a pipe is to use it in your own applications, so this page also displays a number of different ways that you can render your pipe, such as RSS or JSON. Click the RSS link to show your pipe as an RSS feed. Notice that the address to your pipe as an RSS feed is `http://pipes.yahoo.com/pipes/pipe.run?_id=810da380a19e1e1f6e37b4921895`

12fd&_render=rss. The RSS feed is nearly identical to the original feed. Because you didn't make any changes to the data in the Pipes editor, this is to be expected.

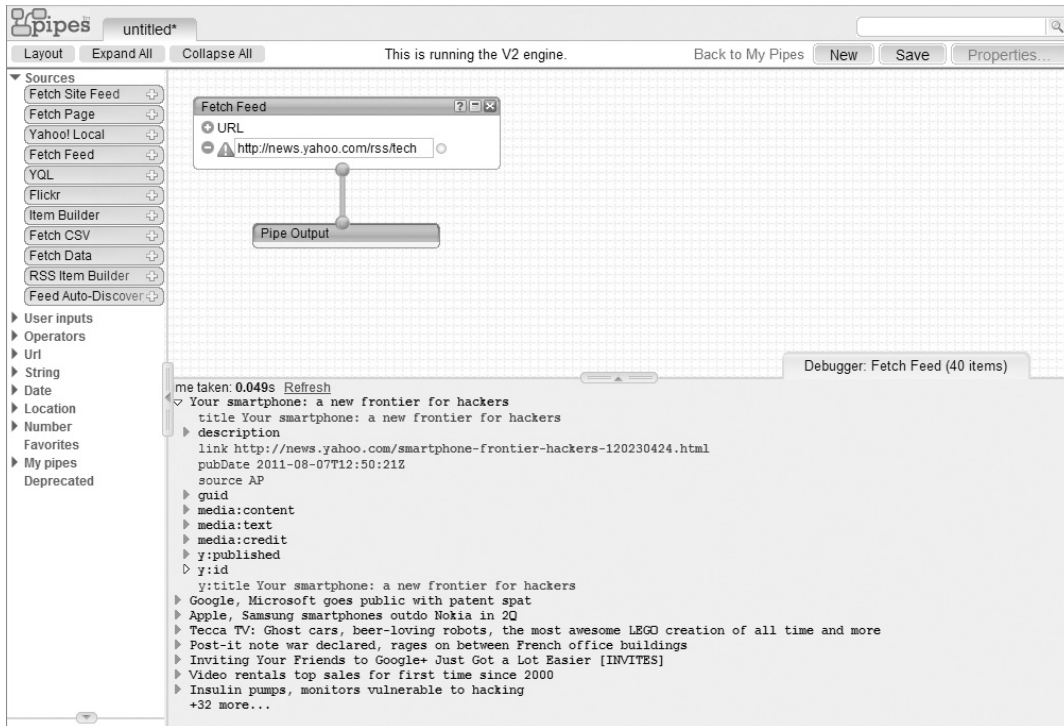


FIGURE 4-3

Change the last parameter to `json` to render the pipe as JSON. The address would look like `http://pipes.yahoo.com/pipes/pipe.run?_id=810da380a19e1e1f6e37b492189512fd&_render=json`. The following is a short snippet of your pipe rendered as JSON. Although this is not very readable, you can begin to see the power of Yahoo Pipes to manipulate data into whatever format you need in this snippet of the JSON response:

```

{"count":40,"value":{"title":"Yahoo Tech News","description":"Pipes
Output","link":"http://pipes.yahoo.com/pipes/pipe.info?_id=810da380a19e1e1f
6e37b492189512fd","pubDate":"Sun, 07 Aug 2011 12:20:37 -
0700","generator":"http://pipes.yahoo.com/pipes/","callback":"","items":[{"
title":"Your smartphone: a new frontier for hackers","description":"<p><a
rel=\"nofollow\" target=\"_blank\" href=\"http://news.yahoo.com/smartphone-
frontier-hackers-120230424.html\"><img
src=\"http://l.yimg.com/bt/api/res/1.2/15sR9jZsOvNmrwuB.T.13Q-
/YXBwaWQ9eW5ld3M7Zmk9ZmlsbD0PTg2O3E9ODU7dz0xMZA-
/http://l.yimg.com/os/en_us/News/ap_webfeeds/20d4458ba1f62c11f50e6a7067
002a21.jpg\" alt=\"photo\" align=\"left\" title=\"FILE - In this Jan. 5, 2011
file photo, a person operates their iPhone in New York. Security experts say
attacks on smartphones are growing fast &#x002014; and attackers are becoming
smarter about developing new techniques. (AP Photo\\Frank Franklin II, File)\"
  
```

```
border="\0\"/>\></a>Hackers are out to stymie your smartphone.</p><br
clear="\all\"/>\>", "link": "http://news.yahoo.com/smartphone-frontier-hackers-
120230424.html", "pubDate": "2011-08-
07T12:50:21Z", "source": "AP", "guid": {"isPermaLink": "false", "content": "smartphone
-frontier-hackers-
120230424"}, "media:content": {"height": "86", "type": "image/jpeg", "url": "http://
1.yimg.com/bt/api/res/1.2/15sR9jZsOvNmrwuB.T.13Q-
\YXBwaWQ9eW5ld3M7Zmk9ZmlsbDtpTg2O3E9ODU7dz0xMzA-
\http://1.yimg.com/os/en_us/News/ap_webfeeds/20d4458balf62c11f50e6a7067
002a21.jpg", "width": "130"}, "media:text":
```

The screenshot shows the Yahoo Pipes web interface. At the top, there's a navigation bar with links like Home, My Pipes, Browse, Discuss, Documentation, and a 'Create a pipe' button. A notification banner states: 'In the first week of August, all Pipes will be upgraded to the V2 engine. Report V2 engine issues here ...'. The main content area displays a pipe titled 'Yahoo Tech News' by Paul Stubbs. The pipe's description is 'Click to add description'. Below the description, there's a 'Pipe Web Address' field with the URL 'http://pipes.yahoo.com/pipes/pipe.info?id=810da380a19e1e1f6e37b492189512fd' and buttons for 'Edit Source', 'Delete', 'Publish', and 'Clone'. To the right of the pipe, there are options to 'Get as a Badge', 'MY Yahoo!', 'Google+', 'Get as RSS', 'Get as JSON', and 'More options'. The pipe is currently set to 'Image' view, showing a list of 40 items. The first item is titled 'Ariane rocket launches two satellites' and includes a thumbnail image of the rocket launch. The sidebar on the left contains sections for 'Properties' (Not published, 0 clones, Engine: v2), 'Bookmark / Share' (with social media icons), 'Tags (0)' (with an 'add new tag' button), 'Sources (2)' (listing yahoo.com and news.yahoo.com), and 'Modules (1)' (listing fetch). At the bottom of the sidebar is an 'Edit Source' button.

FIGURE 4-4

Since Yahoo Pipes can output the results of the data in a standard JSON format, you can use any tool or technique to manipulate the data.

Consuming Pipes

It is time to create a SharePoint web part to consume the Yahoo pipe you just created. In this example, you will create a simple sandboxed visual web part that uses jQuery to render the feed as a

simple list. This is intentionally a very simple web part, but it clearly demonstrates how easy it is to consume pipes from SharePoint. Begin in Visual Studio on your SharePoint development machine:

1. Create a new empty SharePoint project in Visual Studio called **SharePointPipes**.
2. Add a new visual web part (sandboxed) project item called **YahooPipes**.
3. Edit the `YahooPipes.ascx` file. By default, you will see the following boilerplate ASP.NET code in the page. You do not need to change this generated code; instead, you will add your code to the bottom of this file in the next step:

```
<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>
<%@ Assembly Name="Microsoft.Web.CommandUI, Version=14.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
    Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Register TagPrefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages"
    Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="YahooPipes.ascx.cs"
    Inherits="SharePointPipes.YahooPipes.YahooPipes" %>
```

4. Load jQuery by adding the following code at the bottom of the `YahooPipes.ascx` page:



Available for
download on
Wrox.com

```
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
    // Load jQuery
    google.load("jquery", "1.6.2");
</script>
```

code snippet 076576 Ch04_Code.zip/YahooPipes.ascx

This code uses the Google APIs to load the 1.6.2 version of jQuery. Note that there are many different ways to load jQuery. This example is just one of the ways.

5. Add a `div` element placeholder called **pipeFeed**. This `div` element is where the list of pipe feed items will be inserted.



Available for
download on
Wrox.com

```
<div id="pipeFeed">
</div>
```

code snippet 076576 Ch04_Code.zip/YahooPipes.ascx

This aspect of the example is also intentionally kept very simple to make it easy to understand what is going on. A real-world application would use some more advanced techniques here, such as jQuery templates. A jQuery template is an extension to jQuery that enables you to use XAML-style data binding. I encourage you to visit the jQuery template's site, at <http://api.jquery.com/jquery.template>, to learn more.

6. Add the following jQuery code to load and parse the Yahoo pipe feed. This code runs when the page is loaded and ready. It's a little dense, but the inline comments make it easy to understand:



```
<script type="text/javascript">
    // Run when document is loaded
    $(document).ready(function () {
        // Load the Yahoo Pipe as JSON

$.getJSON('http://pipes.yahoo.com/pipes/pipe.run?_id=810da380a19e1ef6e37b492189512fd&_
re
nder=json&_callback=?',
    function (data) {
        // The JSON is returned from the service in the data variable

        // Iterate through each item in the items array
$.each(data.value.items, function (i, item) {
    // Insert a hyperlink for each item
    $('<a>').attr('href', item.link).text(item.title).appendTo('#pipeFeed');
});

        // Add all the item links to a list
        $('#pipeFeed a').wrapAll('<ul>').wrap('<li>');

    });
});
</script>
```

code snippet 076576 Ch04_Code.zip/YahooPipes.ascx

The code first uses the jQuery function `getJSON`. You will update the URL in the `getJSON` call with the URL of your own pipe. This function does all the heavy lifting. When the results return, a function is called and the JSON response is passed in the `data` variable. Next, you iterate over the `items` array. One way to visualize the structure of the JSON data is to use the F12 debugger built into Internet Explorer. If you set a breakpoint and inspect the `data` variable, you would see something like what is shown in Figure 4-5.

The F12 debugger is able to display the JSON data object as a tree, which makes it very easy to browse and discover the correct variable names and the hierarchy of the data. This is going to be helpful when you insert the data onto the HTML page or when data binding using jQuery templates.

The next line actually inserts the Title and Hyperlink in the `pipeFeed` div as a hyperlink element. This is just straight jQuery code. The only things to note are the `link` and `title` properties of the item object. This is where understanding the JSON structure using the F12 debugger becomes helpful.

```
 $('<a>').attr('href', item.link).text(item.title).appendTo('#pipeFeed');
```

Finally, the last line just wraps all the link elements in an unordered list.

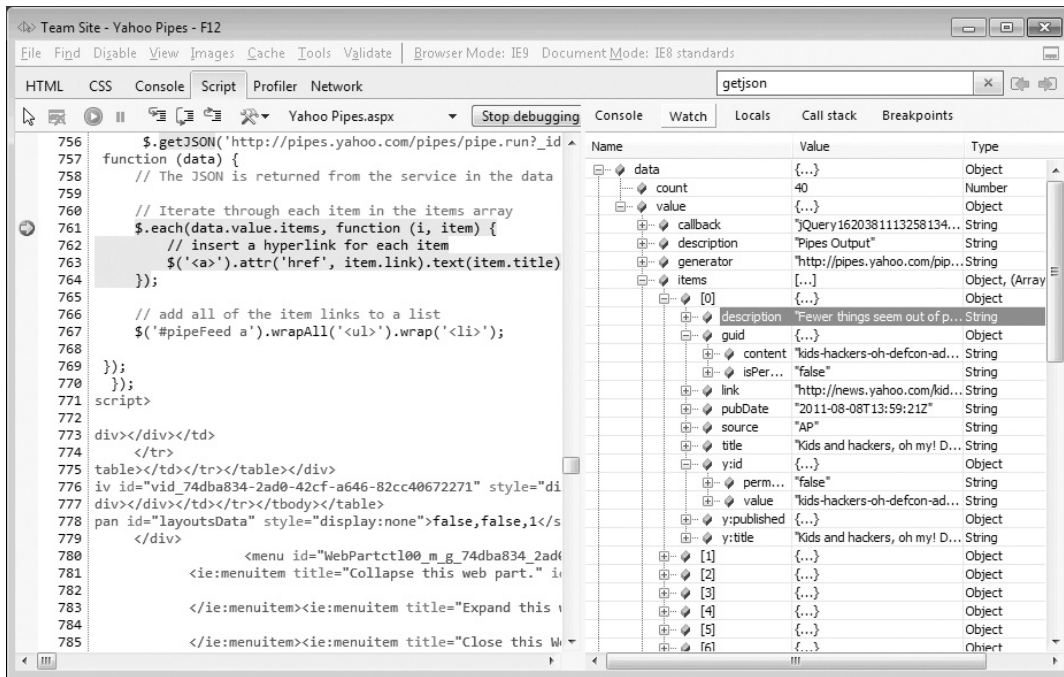


FIGURE 4-5

7. Press F5 to run the project. The solution will build and deploy the web part to the site configured for the project (in this example, `intranet.Contoso.com`).
8. Create a new site page called **Yahoo Pipes Viewer**. This is a test page that you will use to host the YahooPipes web part that you created.
9. Insert the YahooPipes web part to the Yahoo Pipes Viewer site page, and then save the page.

You should now see the results displayed from the Yahoo pipe you created earlier in this chapter (see Figure 4-6).

As shown in this example, Yahoo Pipes is capable of abstracting the data manipulation for your applications. Just as building declarative workflows makes it easy for power users to create workflow logic without requiring a developer, Yahoo Pipes provides the same advantage for your data sources. This example was very simple. Next you will look at some examples that demonstrate the kinds of things you can do with Yahoo Pipes.

Joining Multiple Feed Sources

You have created the basic SharePoint web part that displays data from a Yahoo pipe. Suppose that the business owner now requests that you add an additional feed source to display gaming news as well. If you had created the web part using traditional techniques, you would most likely need to crack open the source code of the web part, make the code changes, and then test and redeploy the web part. However, because you created the web part with a mashup tool, you can simply open the Pipes editor and make the changes.

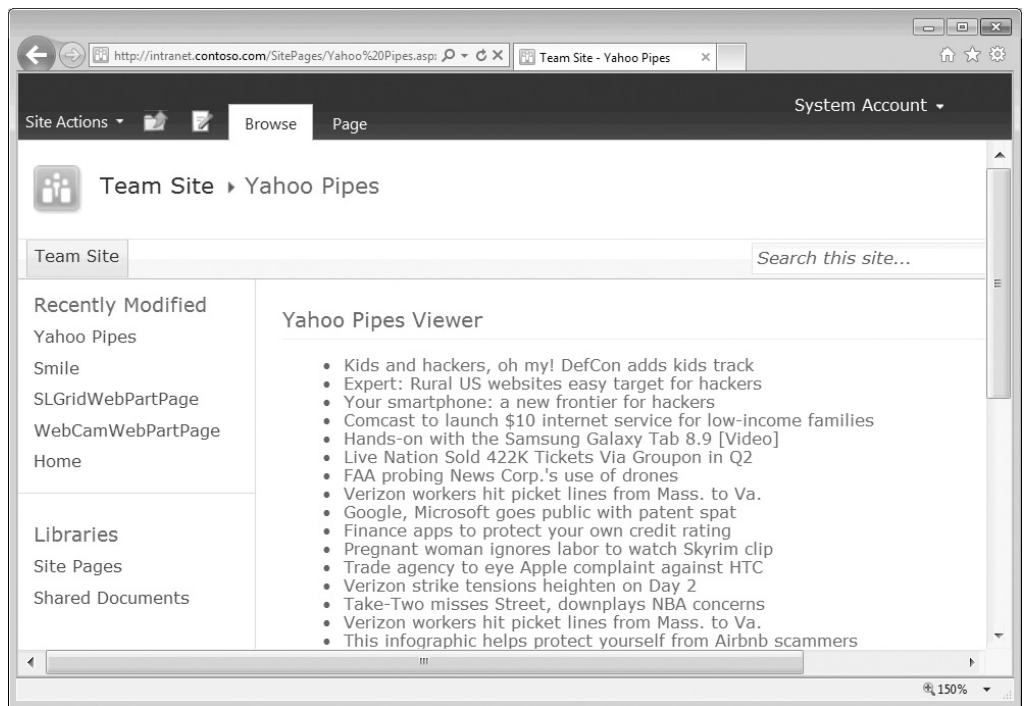


FIGURE 4-6

1. Open the Yahoo Pipes site and sign in.
2. Click the My Pipes link on the main menu to view your existing pipes.
3. Hover over the pipe, open in the editor and click Edit Source from the hover menu. There are a couple of ways to add a second URL to the current Fetch Feed module on the design surface, but in this example you are going to add another Fetch Feed module.
4. Drag the Fetch Feed module from the sources pane on the left side of the Yahoo Pipes designer to the design surface.
5. Set the URL to <http://news.yahoo.com/rss/gaming/> to pull the Gaming News RSS feed.

At this point, you have two Fetch Feeds on the page: the original pull from Tech News and the new one pulling from Gaming News. Only the Tech News feed is wired up to the pipe output. The pipe output only accepts one input, so you need another way to merge the feeds. To do this, you need to use the Union module from the Operators list.

6. Drag the Union module from the Operators list on the left to the design surface between the two feeds and the pipe output. The Union module can merge up to five data sources together.
7. Hover over the existing link between the Fetch Feed module and the Pipe Output module, and click to cut the link. Drag the module handle from the two Fetch Feed modules to the Union module to add the new links. Then connect the Union module to the Pipe Output module (see Figure 4-7).

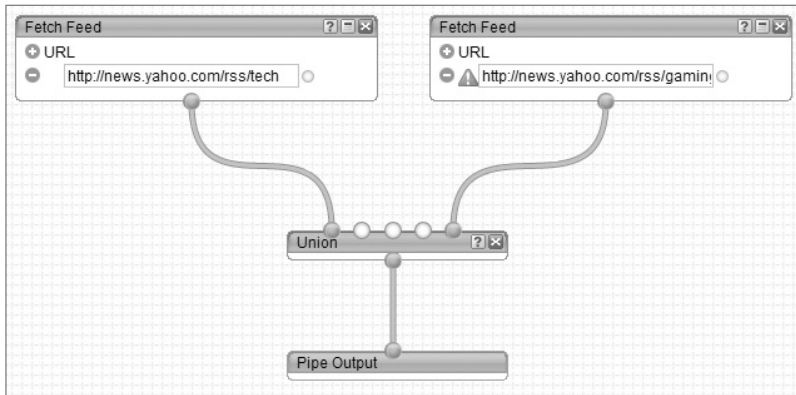


FIGURE 4-7

8. Click on the Pipe Output to see the results in the debug window. In this case, the Pipe Output reports that there are 70 items and displays the first eight.

You can also click on any module in the pipe design to see the results of the output up to that step. For example, if you click the first Fetch Feed module, which contains Tech News results, 40 items are shown. If you click the other Fetch Feed module, for the Gaming News, there are 30 items. Also, as expected, clicking the Union module reports 70 items.

There are no code changes to make in SharePoint or Visual Studio. You have decoupled the data source from the web part. After making changes to the pipe, you simply need to go the SharePoint page you created earlier and refresh the browser. You will now see all 70 items in the browser (see Figure 4-8).

Sorting and Filtering Feed Sources

Yahoo Pipes can easily sort items. Imagine the design as water (or data) flowing through pipes from top to bottom. To sort, all you need to do is drag the Sort module from the Operators section onto the design surface and wire up the in and out points. Therefore, if you want to sort the feed items before you merge them, put the Sort module between the Fetch Feed and Union modules. If you want to sort after the merge, put the Sort module between the Union and Pipe Output modules, as shown in Figure 4-9.

The Sort module requires you to specify the field to sort on. In this case, choose `item.title` from the drop-down list to sort on the title. You can specify to sort in ascending or descending order, and the Sort module enables you to sort by multiple fields — just click the plus (+) icon to add additional sort fields. Save the pipe and then open the SharePoint page to view the updated pipe data in your web part. As shown in Figure 4-10, the web part contains a sorted mix of tech and gaming news.

Filtering for a set of items is just as easy. Just as you did with sorting, you must drag the Filter module from the Operators list onto the design surface. Decide at which point in the pipe you want the filter to apply. In this example, the filter applies after the data is merged and before it is sorted, as shown in Figure 4-11.

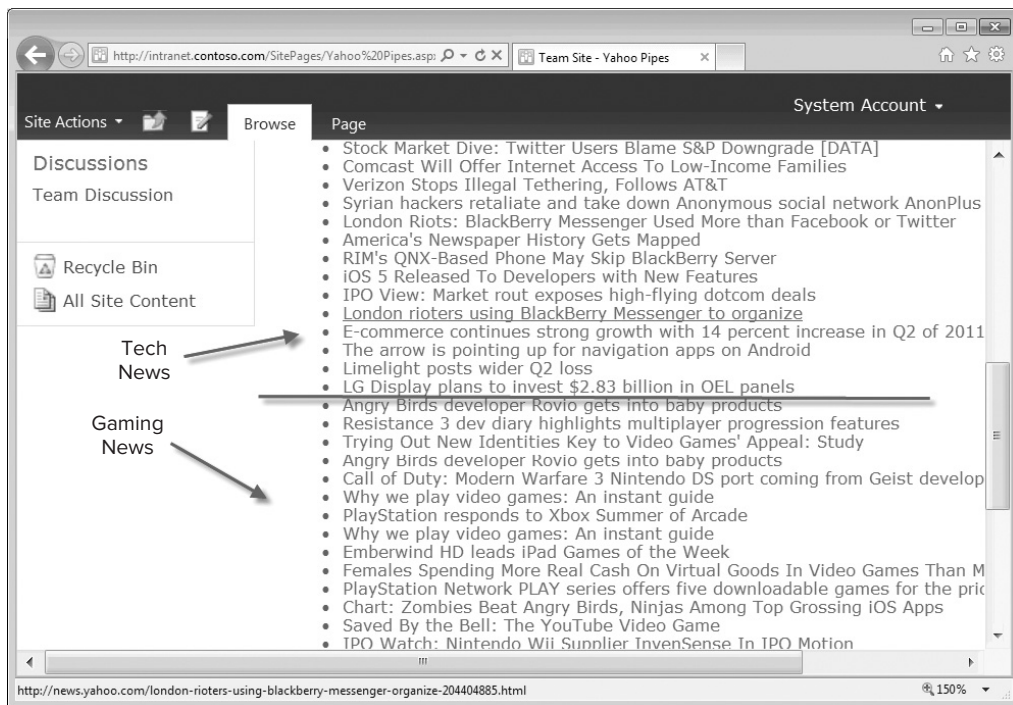


FIGURE 4-8

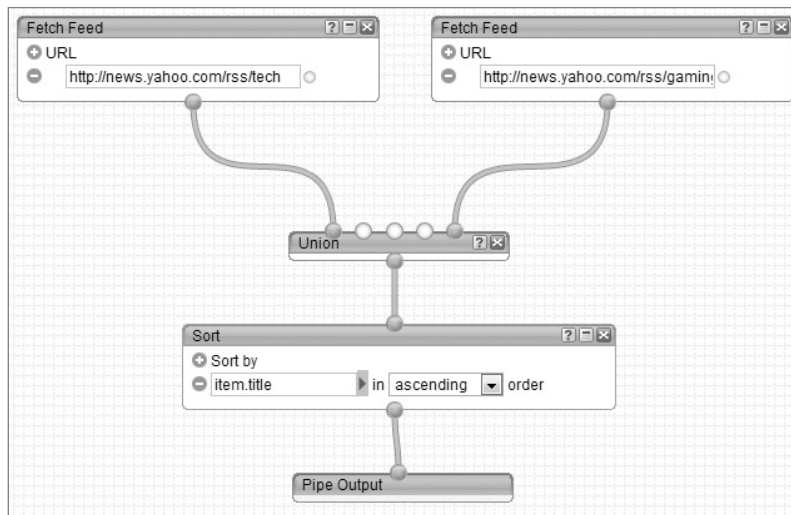


FIGURE 4-9

Yahoo Pipes Viewer

- A Plea to the Video Game Industry to Stop Stacking Release Dates
- A new Rambo video game is on the way
- Angry Birds developer Rovio gets into baby products
- Best iPad Games
- Call of Duty: Modern Warfare 3 Nintendo DS port coming from Geist develop
- Chart: Zombies Beat Angry Birds, Ninjas Among Top Grossing iOS Apps
- Cut the Rope: Experiments tops iPhone Games of the Week
- Emberwind HD leads iPad Games of the Week
- Females Spending More Real Cash On Virtual Goods In Video Games Than M
- Five iPhone games coming to a console near you
- IPO Watch: Nintendo Wii Supplier InvenSense In IPO Motion
- Lolcats get Xbox Live Security through the day
- Nintendo 3DS Japanese sales drop 50 percent after price cut
- Nintendo President apologizes for 3DS price drop
- No Wii U price or release date announcement until 2012
- Perked Up

FIGURE 4-10

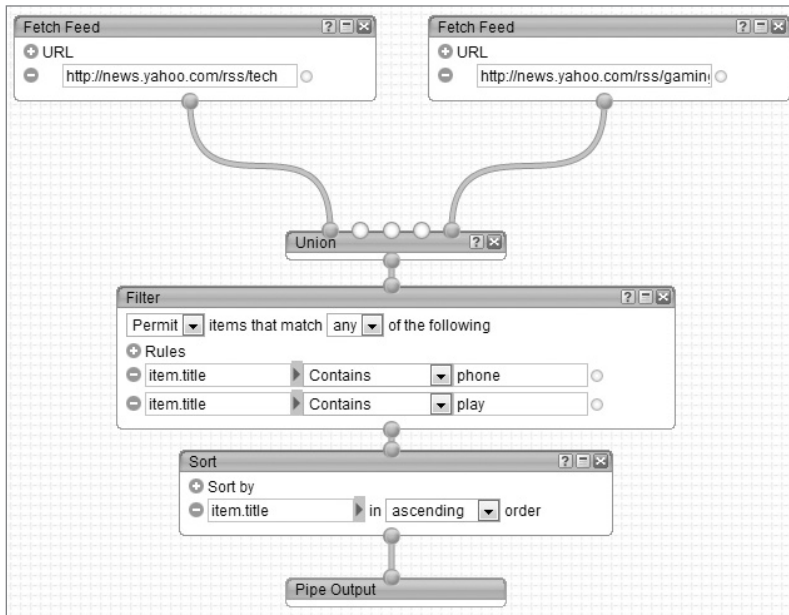


FIGURE 4-11

The Filter module allows you to apply a permit or block filter to items passing through the module. You can also add multiple fields. In this example, the filter permits items that have “phone” or “play” in the title. Save the pipe and return to your SharePoint test page. Refresh the page to see the new filtered and sorted list of Tech and Gaming News feed items, as shown in Figure 4-12.

This example just scratches the surface of what you can do with Yahoo Pipes and SharePoint. There is still much more you can learn to create more complex pipes. No matter how complex the pipes become, the basic process of adding modules and wiring them together remains the same.

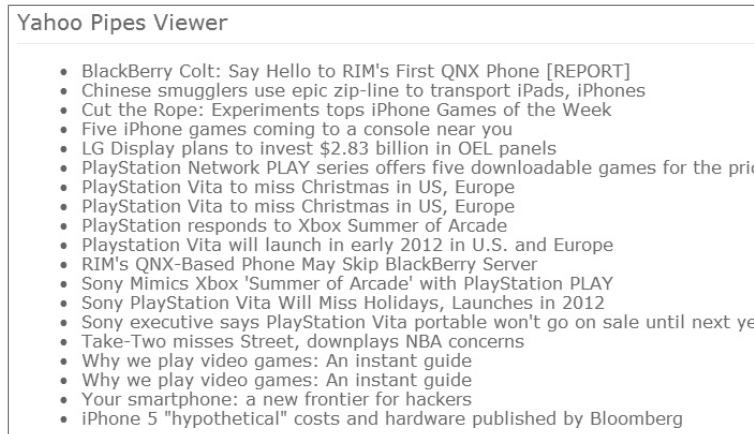


FIGURE 4-12

Using Existing Pipes

You saw earlier how to create a very simple pipe that consumes a couple of feeds and sorts and filters them. How do you get to the next level of complexity with your pipes? Although many good blogs, forums, tutorials, and videos are available, in my experience the best approach is to learn from others. This is what has made HTML and JavaScript so popular — you can visit any page on the Internet and view its source. Yahoo Pipes provides this same capability; you can browse any published pipe from anyone. You can clone these pipes or simply use them as-is. You could even use another pipe as a module in your pipe.

The list of current pipes is available at <http://pipes.yahoo.com/pipes>. Click the Browse link at the top of the page to browse the catalog of pipes. By default, the view is sorted by most popular, but there are many ways to find the right pipe. You can search by free form text or filter them by tag. You can filter by format, such as csv, georss, media, and more, by data source, or by modules used. Once you find the pipe you are looking for, click View Results from the hover menu to run the pipe, or click View Source to see the pipe in the designer.

The following example demonstrates how to leverage an existing pipe in the pipe you created earlier. In your existing pipe, you merged together two RSS feeds, Tech News and Gaming News. This worked well, but looking at the results merged together, it is difficult to tell which news item came from which feed. It would be nice if you could add a tag to the title to indicate which feed was the source.

1. Start by searching for a pipe using “fetch feed prepend” as the search phrase. Four results are returned, as shown in Figure 4-13.
2. Click the first result, “Fetch Feed, Prepend [string] to title, Trunc N (default 3).”

The details page shows information about the pipe, such as author, publish date, and pipe engine version. The number of times the pipe was cloned is an indication of its popularity. The modules used are listed as well — in this case, fetch, regex, textinput, truncate, strconcat, numberinput, and urlinput.

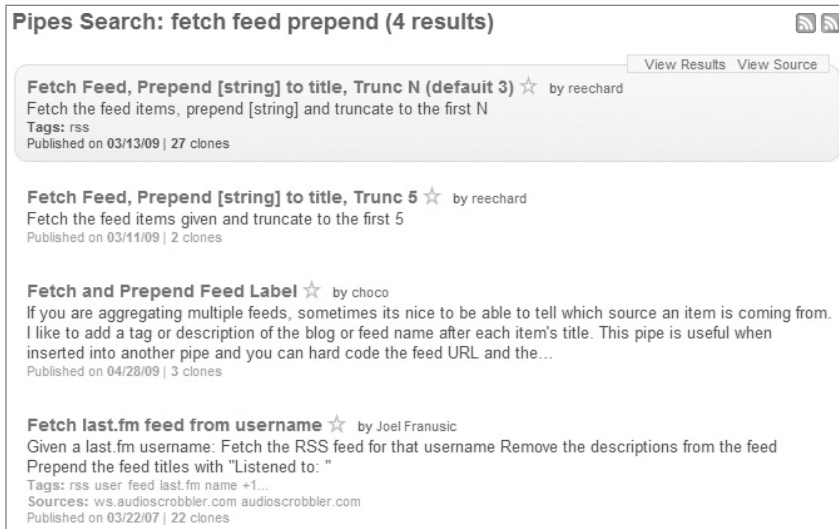


FIGURE 4-13

In the center of the details page is a running instance of the pipe. Unlike when you ran the pipe you created earlier, this particular pipe does not display results; it has a number of input parameters that must be filled in before it can run. The first parameter is the number of items to display. By default, this is set to 3. The second parameter is the text to prepend to the title of every item. The last parameter is the actual URL for the RSS feed source.

3. Set the “[text] to prepend ...” parameter to **gaming** and the “feed url” parameter to **http://news.yahoo.com/rss/gaming**, and then click the Run Pipe button. After the pipe runs, three items are returned from the gaming list, with the text “[gaming]” prepended to each item (see Figure 4-14).

There are a couple of things you could do at this point. For example, you could review the source of the pipe to learn what makes it tick. This is a great way to learn tips and tricks from others.

4. Sign in with your Yahoo, Facebook, or Google ID, and then click the View Source link to open this pipe in the designer. You will see something similar to Figure 4-15. Note that I rearranged the modules to make it a little easier to follow the data flow of the pipe. At the top are the three input parameters of the modules: `numberinput1`, `feedurl`, and `texttoprependbeforeeveryitem`. These parameters are wired up to properties of another module and have a gray colored line. As a pipe design gets more complex, the wires can be difficult to follow. One nice touch of the designer is that when you hover over a line, the modules that may cover the line become a little transparent, enabling you to see where the line connects.

You can see this on the right side of Figure 4-15, where the line connects to the String Builder module. The String Builder module concatenates the `texttoprependbeforeeveryitem` parameter with a left and right bracket character. The Truncate module takes the feed and limits the number of items equal to the `numberinput1` parameter. Each item in the feed is finally prepended with the text using the Regex module.

Fetch Feed, Prepend [string] to title, Trunc N (default 3)
 Fetch the feed items, prepend [string] and truncate to the first N
 Pipe Web Address: http://pipes.yahoo.com/pipes/pipe.info?_id=c8bd5b8a6af7626af5a2e901fc186f2
 ☆ View Source Clone

Configure this Pipe

numberinput1

[text] to prepend before every item's title

feed url

Use this Pipe

Get as a Badge MY Yahoo! Google™ Get as RSS Get as JSON More options ►

Image **List** 3 items

[gaming] World of Warcraft bleeding players despite new content
 Down about 600,000 players in about seven months during the last report, Blizzard reported another loss in the second quarter. Subtracting another 300,000 players from the player base, the subscriber base for the World of Warcraft massive multiplayer online game has dropped by nearly one million users in about nine months. This decline may have been slowed by the recent move to convert the game to a free-to-play model. Players can play for an unlimited amount of time for free until a character hits level...

[gaming] Trying Out New Identities Key to Video Games' Appeal: Study
 MONDAY, Aug. 8 (HealthDay News) — One reason why people worldwide spend 3 billion hours per week playing video games may be because the games allow them to "try on" characteristics they might like to have, a new study suggests.

[gaming] Angry Birds developer Rovio to offer baby products
 Not content with having a presence on just about every smartphone and tablet in the world, Angry Birds creator Rovio is now moving into the world of baby products.

[Report abusive Pipe](#)

FIGURE 4-14

After you understand what the pipe is doing, you may want to pull out some of the techniques used to perform a particular task. You would then just add those same modules to your pipe. You might also decide that you like what the pipe does except for a few things, in which case you could clone the pipe, which creates a copy of the pipe that will show up in your list under My Pipes. After you clone the pipe, you are free to make any changes to your cloned copy, as if you created it from scratch.

You also can leverage a cloned pipe as a module, as follows.

5. After cloning the “Fetch Feed, Prepend [string] to title, Trunc N (default 3)” pipe, open the pipe you created earlier in this chapter. In this example, you will replace the two Fetch Feed modules for tech news and gaming news with the “Fetch Feed, Prepend [string] to title, Trunc N (default 3)” module that you just cloned, and which appears under your My Pipes section.
6. Drag two instances of the “Fetch Feed, Prepend [string] to title, Trunc N (default 3)” module onto the design surface above the Union module.
7. Cut the existing links between the Fetch Feed modules and the Union modules.
8. Wire each instance of the `FetchFeedPrependstringtotitleTruncNdefault3Copy` to the Union module. Set the `numberinput1` parameters to 6 for both instances. Set the “[text] to

prepend ...” parameter” to **Tech** and **Gaming** for each instance, respectively. Finally, set the “feed url” parameters to <http://news.yahoo.com/rss/tech> and <http://news.yahoo.com/rss/gaming>, respectively.

The final pipe should look similar to Figure 4-16.

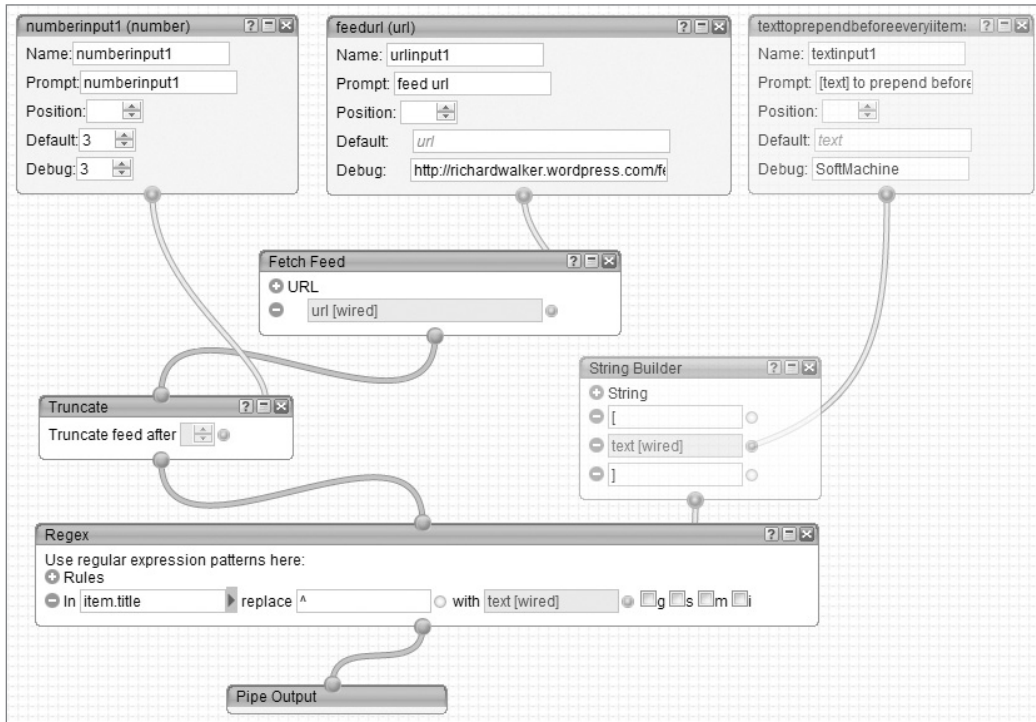


FIGURE 4-15

9. Save your updated pipe, and then run it from the Yahoo Pipes site to verify that it is working correctly. Once you are satisfied that it is returning the results you are expecting, switch over to SharePoint and open your web part page. Figure 4-17 shows some example results.

This example has demonstrated how Yahoo Pipes provides an easy way to mash up Internet data using a visual designer that runs in the browser. You were able to easily surface this data inside of a SharePoint web part using jQuery. It is also rather simple to leverage other pipes created by the community. Decoupling the data feeds from the application enables rapid changes to occur without compiling or deploying new code to SharePoint; and because this code runs as a sandboxed web part, you can run these types of web parts on Office 365.

Of course, sometimes you need to consume more complex data sources, such as REST services, or maybe you need a script to modify the data. For these types of scenarios, Yahoo has created the Yahoo Query Language (YQL). The next section describes how to use YQL to extend your Yahoo pipes.

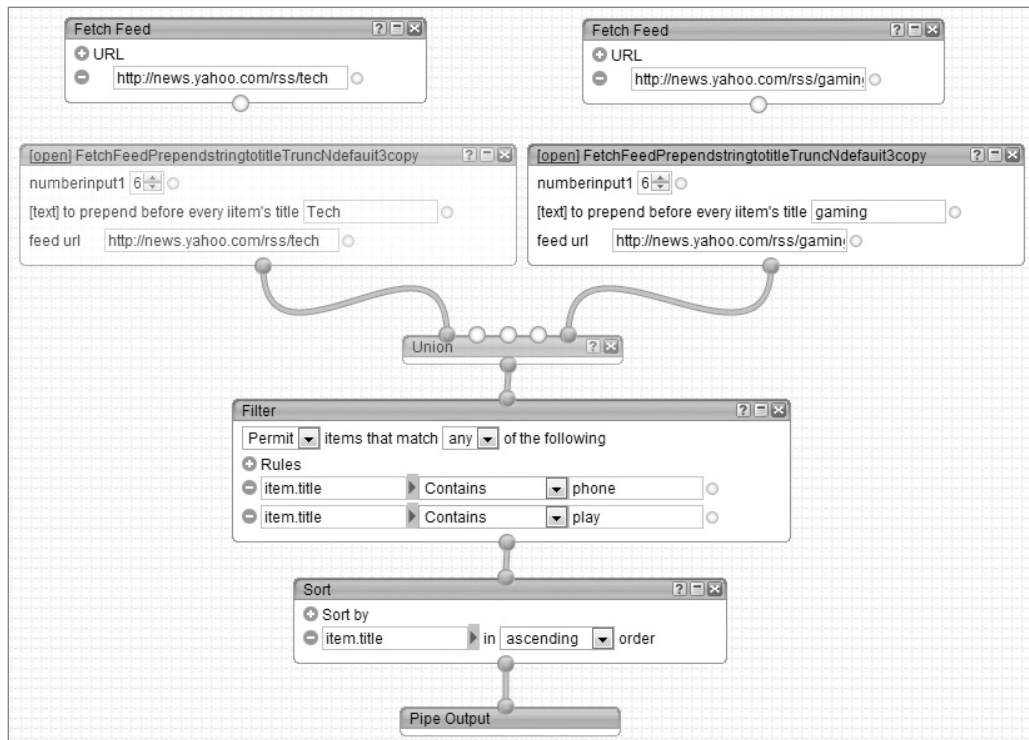


FIGURE 4-16

Yahoo Pipes Viewer	
•	[Tech] Chinese smugglers use epic zip-line to transport iPads, iPhones
•	[Tech] Your smartphone: a new frontier for hackers
•	[gaming] PlayStation responds to Xbox Summer of Arcade
•	[gaming] Why we play video games: An instant guide

FIGURE 4-17

EXTENDING YOUR PIPES WITH YAHOO QUERY LANGUAGE

YQL enables you to query Internet data sources using a SQL-like syntax. Just like you can with a database SQL syntax, you can query, join, and filter data from the Internet. The problem is that most Internet sites surface their data in the form of web services or REST services. Each of these services may have a different API, which you would need to locate and understand how it works. YQL attempts to normalize all these sources using a single query language.

One of my favorite sources for finding APIs that can be used in a SharePoint application is the site ProgrammableWeb (www.programmableweb.com/apis). As of this writing, there are 3,627 different web APIs registered on the site. Clearly, trying to learn all these APIs would be an impossible task.

One way to think about YQL is that, at a high level, it is conceptually similar to the SharePoint Business Connectivity Services (BCS) — not necessarily in the implementation, but in the problem they solve. BCS creates a mapping between some other external data sources and a SharePoint list. YQL also creates a mapping between external data on the Internet and a table, called an *open data table*. In both cases, the goal is to abstract the details of the data source from the consumer. In SharePoint, you can access the external list just like any other regular list. In YQL, you can access each open data table the same way, regardless of the data's source.



I highly encourage you to browse the community-created open data tables at www.datatables.org. You can use these tables as-is or use them as templates for learning how to build your own custom tables.

YQL is a somewhat of an advanced topic, and this chapter by no means attempts to cover all its features. The goal is to introduce you to YQL and how you can use it to take your Yahoo pipes to the next level.

Using the YQL Console

The YQL console is the test tool for writing YQL queries against open data tables. To get started, sign in to the YQL developer page at <http://developer.yahoo.com/yql>. This page has links to documentation, support, and other resources. Open the YQL console by clicking the Try the Console button. When the YQL console opens, it is preloaded with a YQL query to return all the open data tables. As shown in Figure 4-18, the console is divided into a number of panels.

The top-left panel is where you write your YQL query. You can choose XML or JSON as the output format. Clicking the Test button will run your query and show the results in the middle-left panel. The results panel has two display formats: Formatted, which shows only the raw XML, and Tree, which displays the data as a tree structure. The bottom panel shows the REST command to call the YQL query with a REST call.

You can copy the REST query into your browser window to test the query. For example, enter the following URL in your browser:

```
http://query.yahooapis.com/v1/public/yql?q=show%20tables&diagnostics=true
```

The results will look something like Figure 4-19. This is a list of the available open data tables. This is the same list that you see in the console on the bottom-right.

On the top-right of the console are a few sample queries to help get you started. There is a section for your recent queries and a list of query aliases. Think of the query alias as a shortcut to your query (like `bit.ly` is for hyperlinks).

Let's start by looking at one of the example queries. In the example queries list on the right side of the console window, click "get 10 flickr "cat" photos." You will see the following link in the statement window:

YOUR YQL STATEMENT [permalink](#) [Create Query Alias](#)

show tables

☒ XML ☐ JSON ☒ Diagnostics

FORMATTED **TREE** ☐ Wrap Text

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="153" yahoo:created="2011-08-10T05:24:35Z" yahoo:lang="en-US">
  <diagnostics>
    <publiclyCallable>true</publiclyCallable>
    <user-time>3</user-time>
    <service-time>0</service-time>
    <build-version>20450</build-version>
  </diagnostics>
  <results>
    <table>answers.getbycategory</table>
    <table>answers.getbyuser</table>
    <table>answers.getquestion</table>
    <table>answers.search</table>
    <table security="APP">appdb.application</table>
    <table>appdb.categories</table>
    <table>atom</table>
    <table>avatars.get</table>
    <table>csv</table>
    <table>data.uri</table>
    <table security="USER">fantasysports.draftresults</table>
    <table security="USER">fantasysports.games</table>
    <table security="USER">fantasysports.leagues</table>
    <table security="USER">fantasysports.leagues.scoreboard</table>
    <table security="USER">fantasysports.leagues.settings</table>
    <table security="USER">fantasysports.leagues.standings</table>
    <table security="USER">fantasysports.leagues.transactions</table>
    <table security="USER">fantasysports.players</table>
```

THE REST QUERY [How do I use this?](#) [hide](#)

<http://query.yahooapis.com/v1/public/yql?q=show%20tables&diagnostics=true>

QUERY ALIASES

RECENT QUERIES

EXAMPLE QUERIES

- get my social graph
- get my profile data
- get my friends
- get all my friends profiles
- get my friends nicknames
- get my last added friend
- get my friends sorted by nicknan

DATA TABLES (153)

Show Community Tables [What's this?](#)

Filter Tables

social

- social.connections
- social.connections.updates
- social.contacts
- social.contacts.connections
- social.contacts.sync
- social.contacts.updates
- social.entities
- social.notifications
- social.notifications.summary
- social.presence
- social.profile
- social.profile.image
- social.profile.status
- social.relationships
- social.relationships.updates
- social.updates

FIGURE 4-18

```
select * from flickr.photos.search where text="Cat" limit 10
```

This statement will select all fields from the flickr.photos.search open data table for which the text field contains “cat”; the results are limited to no more than 10 items. Click the Test button to run the query. You will see something similar to the following results:

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="10" yahoo:created="2011-08-10T16:16:52Z" yahoo:lang="en-US">
  <diagnostics>
    <publiclyCallable>true</publiclyCallable>
    <url execution-start-time="1312993012158"
      execution-stop-time="1312993012348" execution-
      time="190"><![CDATA[http://api.flickr.com/services/rest/?method
      =flickr.photos.search&text=Cat&page=1&per_page=10]]></url>
    <user-time>191</user-time>
    <service-time>190</service-time>
    <build-version>20450</build-version>
  </diagnostics>
  <results>
    <photo farm="7" id="6029067647" isfamily="0" isfriend="0"
      ispublic="1" owner="32764990@N04" secret="3a4a3e79d5"
      server="6085" title="IMG_1372"/>
```

```

<photo farm="7" id="6029072177" isfamily="0" isfriend="0"
  ispublic="1" owner="32764990@N04" secret="1710acbabe"
  server="6197" title="IMG_1379"/>
<photo farm="7" id="6029064697" isfamily="0" isfriend="0"
  ispublic="1" owner="32764990@N04" secret="d6dec55987"
  server="6146" title="IMG_1369"/>
<photo farm="7" id="6029624580" isfamily="0" isfriend="0"
  ispublic="1" owner="65650827@N05" secret="f4b0c6ec4e"
  server="6127" title="&#40763; #cat #neko #tyobita #photocat
  &#12397;&#12371; &#29483; &#12493;&#12467;
  &#29483;&#12496;&#12459;"/>
<photo farm="7" id="6029610464" isfamily="0" isfriend="0"
  ispublic="1" owner="10989907@N07" secret="d4be617a46"
  server="6071" title="New Mug, New Motto, Cute Cat"/>
<photo farm="7" id="6029604520" isfamily="0" isfriend="0"
  ispublic="1" owner="53611866@N06" secret="5c64a51a5b"
  server="6071" title="Stalker Log"/>
<photo farm="7" id="6029610264" isfamily="0" isfriend="0"
  ispublic="1" owner="27381883@N04" secret="f6ca357c81"
  server="6130" title="Cat"/>
<photo farm="7" id="6029050729" isfamily="0" isfriend="0"
  ispublic="1" owner="53611866@N06" secret="c7bd11eb32"
  server="6138" title="136 prayers for Carl to love the stalker -
  various languages (like shinto shrine)"/>
<photo farm="7" id="6029608272" isfamily="0" isfriend="0"
  ispublic="1" owner="32764990@N04" secret="c90aa6cfff"
  server="6077" title="IMG_1343"/>
<photo farm="7" id="6029036497" isfamily="0" isfriend="0"
  ispublic="1" owner="24614535@N04" secret="4b02d983cd"
  server="6203" title="slo fuzz in the morning"/>
</results>
</query>

```

The flickr.photos.search open data table is a mapping to the Flickr API that was created by Yahoo. The results show 10 items. Notice that there is a title and id but no link to the actual photo page. To remedy this, there are a couple of things you could do. You could run this query and then in your code parse the photo ids and build a valid path to the photo page. This is actually a bit complicated and requires knowledge of the Flickr API. The point of YQL is to abstract this direct manipulation of the API from the data source provider. Looking back at the list of example queries, note that there is a query called “get a flickr photo by photo ID.” Click that query to show it in the console. You will see the following query, which has a photo id already set to 2439864402:

```
select * from flickr.photos.info where photo_id= 2439864402
```

The “get a flickr photo by photo ID” query selects all fields from the flickr.photos.info open data table where the photo id is 2439864402. Click Test to run the query. The result shows a single item returned with all the photo details, including the URL to the Flickr photo page for the image. You can see the URL element value of <http://www.flickr.com/photos/genexe/2439864402> at the bottom of the following the XML results:



FIGURE 4-19

```

<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="1" yahoo:created="2011-08-10T16:24:20Z" yahoo:lang="en-US">
  <diagnostics>
    <publiclyCallable>true</publiclyCallable>
    <url execution-start-time="1312993460445"
      execution-stop-time="1312993460609" execution-
      time="164"><![CDATA[http://api.flickr.com/services/rest/?method
      =flickr.photos.getInfo&photo_id=2439864402]]></url>
    <user-time>166</user-time>
    <service-time>164</service-time>
    <build-version>20450</build-version>
  </diagnostics>
  <results>
    <photo dateuploaded="1209077470" farm="4" id="2439864402"
      isfavorite="0" license="0" media="photo"
      originalformat="jpg" originalsecret="2715a07a5c"
      rotation="0" safety_level="0" secret="ce56cba75c"
      server="3296" views="1329">
    <owner iconfarm="4" iconserver="3645"

```

```

    location="South San Francisco, USA" nsid="8614440@N03"
    realname="Alex Ho" username="generationexe"/>
<title>iJustine and her Flip</title>
<description>Justine Ezarik and Sam Pullara</description>
<visibility isfamily="0" isfriend="0" ispublic="1"/>
<dates lastupdate="1254171313" posted="1209077470"
    taken="2008-04-23 03:12:34" takengranularity="0"/>
<editability canaddmeta="0" cancomment="0"/>
<publiceditability canaddmeta="0" cancomment="1"/>
<usage canblog="0" candownload="1" canprint="0" canshare="1"/>
<comments>1</comments>
<notes/>
<tags>
  <tag author="8614440@N03" id="8591386-2439864402-5913"
    machine_tag="0" raw="web">web</tag>
  <tag author="8614440@N03" id="8591386-2439864402-26493"
    machine_tag="0" raw="2.0">20</tag>
  <tag author="8614440@N03" id="8591386-2439864402-4251"
    machine_tag="0" raw="expo">expo</tag>
  <tag author="8614440@N03" id="8591386-2439864402-11227"
    machine_tag="0" raw="web20">web20</tag>
  <tag author="8614440@N03"
    id="8591386-2439864402-10061458" machine_tag="0"
    raw="web20expo">web20expo</tag>
  <tag author="8614440@N03" id="8591386-2439864402-34067"
    machine_tag="0" raw="moscone">moscone</tag>
  <tag author="8614440@N03" id="8591386-2439864402-46"
    machine_tag="0" raw="sanfrancisco">sanfrancisco</tag>
  <tag author="8614440@N03" id="8591386-2439864402-37813"
    machine_tag="0" raw="2008">2008</tag>
  <tag author="8614440@N03" id="8591386-2439864402-42024"
    machine_tag="0" raw="crawl">crawl</tag>
  <tag author="8614440@N03" id="8591386-2439864402-17079"
    machine_tag="0" raw="bayarea">bayarea</tag>
  <tag author="8614440@N03" id="8591386-2439864402-19522"
    machine_tag="0" raw="conference">conference</tag>
  <tag author="8614440@N03" id="8591386-2439864402-32001"
    machine_tag="0" raw="justine">justine</tag>
  <tag author="8614440@N03"
    id="8591386-2439864402-8825800" machine_tag="0"
    raw="ezarik">ezarik</tag>
  <tag author="8614440@N03"
    id="8591386-2439864402-8024771" machine_tag="0"
    raw="ijustine">ijustine</tag>
  <tag author="8614440@N03" id="8591386-2439864402-4070"
    machine_tag="0" raw="sam">sam</tag>
  <tag author="8614440@N03" id="8591386-2439864402-242477"
    machine_tag="0" raw="pullara">pullara</tag>
  <tag author="8614440@N03" id="8591386-2439864402-2335"
    machine_tag="0" raw="yahoo">yahoo</tag>
  <tag author="8614440@N03" id="8591386-2439864402-13686"
    machine_tag="0" raw="flip">flip</tag>
  <tag author="8614440@N03" id="8591386-2439864402-2546"
    machine_tag="0" raw="video">video</tag>

```

```

    </tags>
    <urls>
      <url type="photopage">http://www.flickr.com/photos/genexe
        /2439864402/</url>
    </urls>
  </photo>
</results>
</query>

```

The problem is taking the output from the first query, which returns a list of photo ids that match your criteria, and passing the photo ids to the second query. Fortunately, YQL supports joining tables with sub-selects. You can find more details from the documentation at <http://developer.yahoo.com/yql/guide/joins.html>. Sub-selects enable you to do exactly what you want to do here, pass the results from one query to another. Copy the following query into the console and click the Test button to run it:

```

select * from flickr.photos.info where photo_id in (select id from
flickr.photos.search where text='Cat' limit 10)

```

This query will return the photo information for each item in the sub-select. You should see something similar to the following listing (results have been abbreviated to two items for brevity):

```

<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="2" yahoo:created="2011-08-10T16:38:28Z" yahoo:lang="en-US">
  <diagnostics>
    <publiclyCallable>true</publiclyCallable>
    <url execution-start-time="1312994308604"
      execution-stop-time="1312994308825" execution-
      time="221"><![CDATA[http://api.flickr.com/services/rest/?method
      =flickr.photos.search&text=Cat&page=1&per_page=10]]></url>
    <url execution-start-time="1312994308826"
      execution-stop-time="1312994308981" execution-
      time="155"><![CDATA[http://api.flickr.com/services/rest/?method
      =flickr.photos.getInfo&photo_id=6029134589]]></url>
    <url execution-start-time="1312994308826"
      execution-stop-time="1312994308987" execution-
      time="161"><![CDATA[http://api.flickr.com/services/rest/?method
      =flickr.photos.getInfo&photo_id=6029678606]]></url>
    <user-time>385</user-time>
    <service-time>537</service-time>
    <build-version>20450</build-version>
  </diagnostics>
  <results>
    <photo dateuploaded="1312993969" farm="7" id="6029678606"
      isfavorite="0" license="5" media="photo"
      originalformat="jpg" originalsecret="2657c38958"
      rotation="0" safety_level="0" secret="4e9e3f70d7"
      server="6195" views="0">
      <owner iconfarm="1" iconserver="217" location="Japan"
        nsid="60223652@N00" realname="Hisashi"
        username="hisashi_0822"/>
    <title>E06_2356</title>
  </results>
</query>

```



```

<description/>
<visibility isfamily="0" isfriend="0" ispublic="1"/>
<dates lastupdate="1312993978" posted="1312993969"
    taken="2011-08-10 15:56:08" takengrularity="0"/>
<editability canaddmeta="0" cancomment="0"/>
<publiceditability canaddmeta="0" cancomment="1"/>
<usage canblog="0" candownload="1" canprint="0" canshare="1"/>
<comments>0</comments>
<notes/>
<tags>
    <tag author="60223652@N00" id="6747496-6029678606-1344"
        machine_tag="0" raw="cat">cat</tag>
    <tag author="60223652@N00" id="6747496-6029678606-36478"
        machine_tag="0" raw="&#29483;">&#29483;</tag>
    <tag author="60223652@N00"
        id="6747496-6029678606-829979" machine_tag="0"
        raw="GB1">gb1</tag>
    <tag author="60223652@N00"
        id="6747496-6029678606-681418" machine_tag="0"
        raw="GB2">gb2</tag>
</tags>
<urls>
    <url type="photopage">http://www.flickr.com/photos/hisashiv
        /6029678606/</url>
</urls>
</photo>
<photo dateuploaded="1312994171" farm="7" id="6029134589"
    isfavorite="0" license="5" media="photo"
    originalformat="jpg" originalsecret="9eed172edf"
    rotation="0" safety_level="0" secret="588596b6d6"
    server="6074" views="0">
<owner iconfarm="1" iconserver="217" location="Japan"
    nsid="60223652@N00" realname="Hisashi"
    username="hisashi_0822"/>
<title>E06_2394</title>
<description/>
<visibility isfamily="0" isfriend="0" ispublic="1"/>
<dates lastupdate="1312994173" posted="1312994171"
    taken="2011-08-10 16:07:17" takengrularity="0"/>
<editability canaddmeta="0" cancomment="0"/>
<publiceditability canaddmeta="0" cancomment="1"/>
<usage canblog="0" candownload="1" canprint="0" canshare="1"/>
<comments>0</comments>
<notes/>
<tags>
    <tag author="60223652@N00" id="6747496-6029134589-1344"
        machine_tag="0" raw="cat">cat</tag>
    <tag author="60223652@N00" id="6747496-6029134589-36478"
        machine_tag="0" raw="&#29483;">&#29483;</tag>
    <tag author="60223652@N00"
        id="6747496-6029134589-3056055" machine_tag="0"
        raw="GG2">gg2</tag>
</tags>
<urls>
    <url type="photopage">http://www.flickr.com/photos/hisashiv/

```



```

6029134589/</url>
    </urls>
  </photo>
</results>
</query>

```

This is the raw results from the YQL query. At this point, you could run this query in your application and build the code required to make the REST call. However, an easier approach would be to leverage the Yahoo pipe that you already created.

Creating a YQL Pipe Web Part

In the previous section, you created a YQL query that calls the Flickr API to return 10 cat images. Suppose that you wanted to add these results to the news feed pipe you created earlier in the chapter. Yahoo Pipes makes this easy to do by providing the YQL module. Open your existing pipe in the designer and drag the YQL module from the Sources list to the design surface. Copy your YQL query from the YQL console to the YQL module instance:

```

select * from flickr.photos.info where photo_id in (select id from
flickr.photos.search where text='Cat' limit 10)

```

The YQL module will return the data in whatever structure the query returns the data. In this example, you are using RSS feeds, so you need a way to transform the YQL results into an RSS feed structure. Use the Create RSS module to transform the YQL results into RSS. The Create RSS module enables you to map the incoming fields into the RSS fields. Figure 4-20 shows the field mappings. The drop-down for each field shows the available fields from the YQL results.

Unfortunately, this does not display all the fields. It shows only the top-level fields in the drop-down. You need to type the field name into the text box. For example, the Link field in the Create RSS module maps to the `item.urls.url` field from the YQL results.

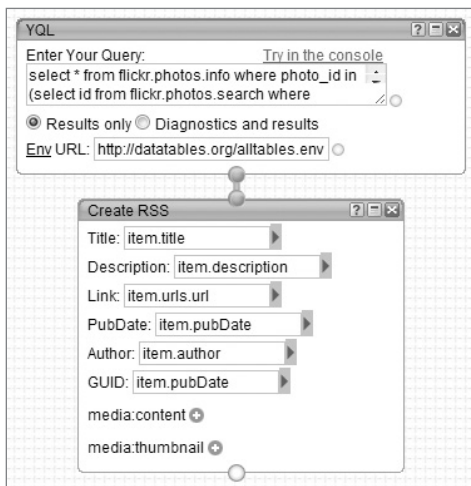


FIGURE 4-20

Finally, connect the Create RSS module to the Union module. Save the pipe and you are done. Figure 4-21 shows the completed pipe. It appears a little crowded in the screenshot, but actually using it in the designer is relatively straightforward.

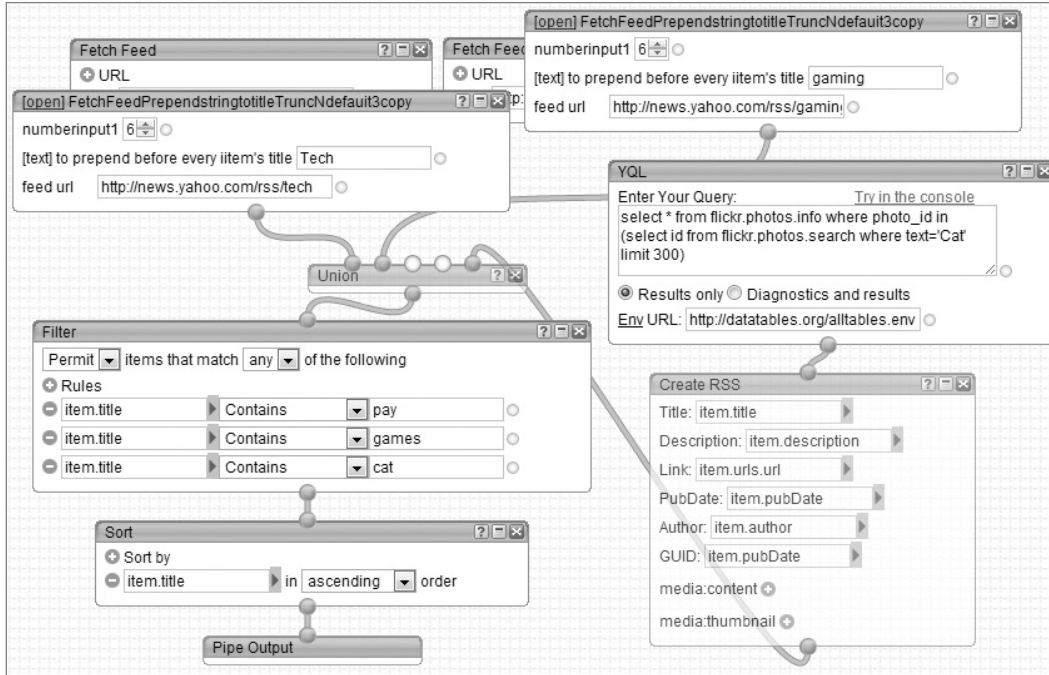


FIGURE 4-21

You now have a Yahoo pipe that merges tech and gaming news with cat photos from Flickr. Your existing SharePoint web part remains the same. Switch back to SharePoint and refresh the page. You will see links to tech and gaming news on Yahoo, and cat photos on Flickr. Figure 4-22 shows your Yahoo Pipes web part with the new items from all three data sources.

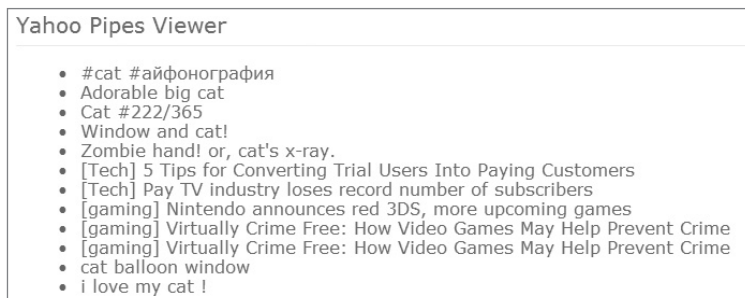


FIGURE 4-22

SUMMARY

You learned in this chapter how to decouple your application from Internet data sources and how to combine different data into a single source. Yahoo Pipes is a great tool for mashing up multiple data streams, and it enables power users to participate in the application process without requiring a developer to make changes to the SharePoint applications. You also learned how to use YQL to normalize the many different Internet data sources into a common format that can be queried consistently. Finally, you saw that combining all these techniques together into a single Yahoo pipe makes the entire end-to-end development process fast and reduces the amount of codes changes.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- **Pipes** — <http://pipes.yahoo.com>
- **Yahoo Developer Network** — <http://developer.yahoo.com/yql>
- **Community Open Data Tables for YQL** — www.datatables.org

5

Connecting LinkedIn and SharePoint Profile Data

WHAT'S IN THIS CHAPTER?

- Understanding the SharePoint social computing platform
- Understanding the LinkedIn social networking programming model
- Using JavaScript and Silverlight to connect LinkedIn and SharePoint profile data

SharePoint provides a rich social computing platform for users to share and collaborate on information. Users accustomed to working with social technologies on the Web expect to use those same techniques and technologies in the enterprise as well, and SharePoint provides support for many of these features. For example, social sites such as Facebook and LinkedIn include a personal dashboard page that enables users to see everything about their social worlds in one place. SharePoint includes a comparable social dashboard, called the *My Site*.

The key concept of social computing is the notion of a *profile*. Your profile is a collection of data about you — your likes, hobbies, pets, skills, and so on. Sharing this information and making connections with your friends (or *colleagues*, as they are called in SharePoint) is at the heart of the social movement. While it is fun to keep track of your friends and family on Facebook, collaborating on this same type of information in your enterprise becomes critical for teams to work together; and collaborating with external colleagues and professionals on LinkedIn can be critical to your professional career.

In this chapter you will learn how to program against some of the social features in SharePoint and LinkedIn and how you can bring together information from both platforms onto your SharePoint site.

OVERVIEW OF SHAREPOINT SOCIAL COMPUTING

SharePoint pulls together in one platform social features that will be familiar to users in your enterprise. This section looks at some of those features, such as My Site, user profiles, tagging, and blogging. Although SharePoint has an opportunity to improve any individual feature, the real power lies in the fact that you have one consistent platform to install, program, learn, and use.

My Site

My Site is SharePoint's social dashboard. It enables you to discover information about your colleagues. It is also your own personal portal space where you can share documents, blogs, wikis, and almost any other SharePoint list or library type. By default, your My Site has a Shared Documents library that enables you to share documents publicly with everyone in your enterprise. It also contains a Private Documents library, which you can use to store personal documents that are visible only to you. As in other SharePoint sites, you can control the permissions to any of these lists and libraries to grant or deny permissions to the resource. Figure 5-1 shows the My Site for Dan Jump, the CEO of Contoso.

My Site My Newsfeed My Content My Profile Find People

View My Profile as seen by: Everyone

10:58 AM

Check Out my LinkedIn Profile

Dan Jump
CEO
Executive
(425) 555-0179
Seattle, WA Redmond
danj@contoso.com
Edit My Profile More information

Dan Jump has been CEO of Contoso since 1985. Since becoming CEO of Contoso, Dan has successfully grown the company from 1 million dollars in annual sales to 2.3 billion. Dan is a graduate from MIT with a MBA with a concentration in foreign affairs. Dan was also an officer in the US Marines, doing a tour in Desert Storm. In his off time he enjoys his 4 grandchildren and his golden retriever, Zephyr.

Overview Organization Content Tags and Notes Colleagues Memberships

Ask Me About

Here are some topics Dan Jump can help you with. To ask a question, click on the relevant topic below.

- Define and Implement...
- Execution Management
- Leadership
- Organization Skills
- Project Management
- Promoting Change

Recent Activities

Dan Jump updated profile. 6/25/2011
Mobile phone: (425) 555-0179
Birthday: July 04

Dan Jump updated profile. 6/25/2011
Skills: C++, Public Speaking, Design

Dan Jump updated profile. 6/25/2011
Interests: Dogs

Dan Jump says "Check Out my LinkedIn Profile".

My Organization Chart

- Dan Jump CEO
 - Adam Barr General Manager of Professional Services
 - Frank Martinez COO
 - Jim Daly CFO
 - Sanjay Shah Chief of Technical Strategy

Organization Browser

In Common with You

In this space, other people who view your page will see things they have in common with you such as:

- First manager you both share
- Colleagues you both know
- Memberships you both share

FIGURE 5-1

A SharePoint My Site is a site collection. By default, you are the site collection administrator of your My Site, which means you can create any artifacts that you could in any regular site collection, such as lists and libraries. You can edit the pages of your My Site, adding and removing web parts and pages. You can also turn features on and off and create any number of subsites. Finally, because you are the site collection administrator, you can upload and run sandboxed solutions on your My Site.

SharePoint Profiles

Profiles are the foundation of all social networking. Your profile contains all the metadata about you — for example, your name, work phone, department, title, and profile picture. You can also enter a description about yourself and list other hobbies, skills, or work-related projects. This information is the key to collaborating and finding others within your organization. Suppose you are looking for someone who is a subject matter expert on expense reporting or who has worked on the expense application. If everyone maintains updated profile information or the information is synchronized for external systems, finding the right expert in your organization is easy to do using SharePoint. Figure 5-2 shows Dan's profile information in edit mode.


Basic Information		Show To
Account name:	contoso\danj	Everyone
Name:	Dan Jump	Everyone
Work phone:	(425) 555-0179	Everyone
Department:	Executive	Everyone
Title:	CEO	Everyone
About me:	<div> <p>Dan Jump has been CEO of Contoso since 1985. Since becoming CEO of Contoso, Dan has successfully grown the company from 1 million dollars in annual sales to 2.3 billion. Dan is a graduate from MIT with a MBA with a concentration in foreign affairs. Dan was also an officer in the US Marines, doing a tour in Desert Storm. In his off time he enjoys his 4 grandchildren and his golden retriever, Zephyr.</p> </div> <p>Provide a personal description expressing what you would like others to know about you.</p>	Everyone
Picture:	 <input type="button" value="Choose Picture"/> <input type="button" value="Remove"/>	Everyone
<p>Upload a picture to help others easily recognize you at meetings and events.</p>		
Ask Me About:	<u>Promoting Change</u> ; <u>Project Management</u> ; <u>Organization Skill</u>	Everyone

FIGURE 5-2

Often your company is already collecting profile information about you in other systems, such as Active Directory and back-end human resources systems. SharePoint enables this information to be synchronized with the SharePoint profile using the User Profile Synchronization service. In many cases this information is synchronized in both directions so that updates you make to your profile on your My Site flow back to your Active Directory profile.



To learn more about SharePoint 2010 user profile synchronization, visit <http://technet.microsoft.com/en-us/library/ee721049.aspx>.

Adding Custom User Profile Properties

SharePoint includes dozens of properties out of the box, and you can enable others and add your own custom properties as well. To add additional properties, perform the following steps:

1. Open the Central Administration website and select Manage Service Applications under the Application Management section.
2. Select the User Profile Service Application and then click Manage from the ribbon.

The User Profile Service management page, as shown in Figure 5-3, enables you to manage all aspects of the SharePoint profiles. You can also see statistics about the number of profiles in use.

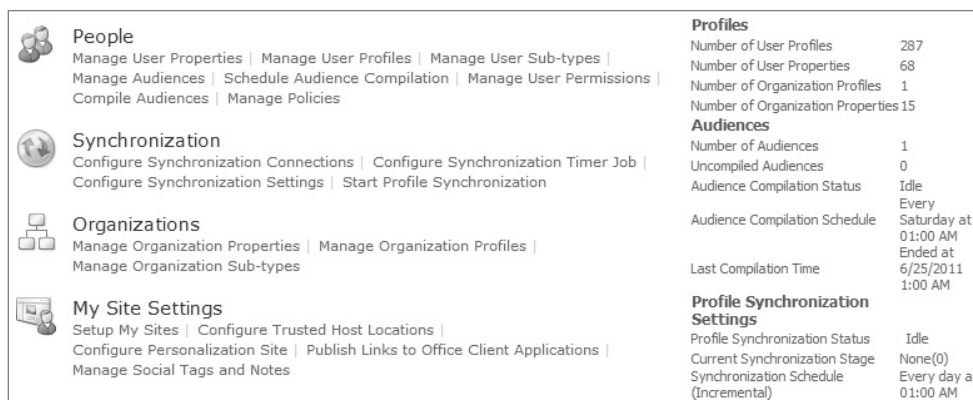


FIGURE 5-3

3. Under the People group, click Manage User Properties to open the property management page (see Figure 5-4).

This page lists all the properties and the order and grouping that they will appear in on the My Site page. You can add a custom property by clicking the New Property link at the top of the page. While it is beyond the scope of this chapter to go into all the details about how to manage, sync, and add properties, you should explore this area on your own.

Adding custom properties is a good way to directly map properties from another profile provider (such as Facebook or, as you will see in this chapter, LinkedIn) directly to a property in your SharePoint My Site. For example, a LinkedIn profile contains a property called Patents, which is where you list all the patents you hold. A SharePoint profile does not contain a patents field or a similar field onto which you could map the value, but you could add a custom field called Patents to hold this information.

Use this page to add, edit, organize, delete or map user profile properties. Profile properties can be mapped to Active Directory or LDAP compliant directory services. Profile properties can also be mapped to Application Entity Fields exposed by Business Data Connectivity.

[New Property](#)
[New Section](#)
[Manage Sub-types](#)
 Select a sub-type to filter the list of properties: **Default User Profile Subtype**
 Go to page **1** of 1

Property Name	Change Order	Property Type	Mapped Attribute	Multivalue	Alias
> Basic Information		Section			
Id	^ v	unique identifier			
SID	^ v	binary			
Active Directory Id	^ v	binary			
Account name	^ v	Person	<Specific to connection>		✓
First name	^ v	string (Single Value)			
Phonetic First Name	^ v	string (Single Value)			
Last name	^ v	string (Single Value)			
Phonetic Last Name	^ v	string (Single Value)			
Name	^ v	string (Single Value)			✓
Phonetic Display Name	^ v	string (Single Value)			
Work phone	^ v	string (Single Value)			

FIGURE 5-4

Social Tagging and Notes

Another key aspect to social computing is the capability to tag and comment on what others are doing. Tagging documents enables users to find and categorize information faster by searching or browsing by the tags. SharePoint provides the capability to add social tags to almost any item in SharePoint, including documents and pages. You can also attach comments or notes to any item. SharePoint also supports a specific type of tagging called *rating*. You can assign rating values to documents so that the search engine can weight the documents in the search results as higher or lower based on the rating score.

On your My Site, all the tags that you have used are organized into a *tag cloud*. A tag cloud displays tags and sizes them based on frequency of use. The more you use a tag value, the larger the font will be. This helps you quickly pick out the most commonly used and, presumably, important tags. Figure 5-5 shows Dan's tag cloud.

Overview	Organization	Content	Tags and Notes	Colleagues	Memberships
Refine by type: All Tags Notes			Activities for: Industry Trends		
Refine by tag: Sort: Alphabetically By Size			Tagged http://intranet.contoso.c... with Industry Trends. 3/9/2011 View Related Activities		
Camera Industry Trends			Tagged http://intranet.contoso.c... with Industry Trends. 3/9/2011 View Related Activities		
Market Analysis Product					
Review					

FIGURE 5-5

Social tagging, just like profile data, increases the chances of users finding the documents and people they are looking for. Users of the information can help shape the search results, pushing the right content, as determined by the expertise of many users, to the top.

Activity Feeds

SharePoint automatically tracks some of your actions — such as tagging, rating, adding notes, and updating your profile — and aggregates this information into an *activity feed*. Others can view your activity feed, and you can view the feeds of others. This enables colleagues to follow what each other is doing and which documents they think are important. You can also extend the type of activities that are captured using custom code, an advanced topic beyond the scope of this chapter. If you are interested in more details, there is a good sample at <http://code.msdn.microsoft.com/activityfeedsconsole>. Figure 5-6 shows a number of Dan's activities.


Recent Activities
Dan Jump updated profile. 6/25/2011 Mobile phone: (425) 555-0178  Birthday: July 04
Dan Jump updated profile. 6/25/2011 Skills: C++, Public Speaking, Design
Dan Jump updated profile. 6/25/2011 Interests: Dogs
Dan Jump says "Check Out my LinkedIn Profile".

FIGURE 5-6

Blogs

Blogs were one of the first tools for socializing on the Internet. SharePoint 2010 also supports blogging, and users can create a blog for their My Site as well. Typically, blogs are written by one author and displayed in the order they were posted. This makes it easy to browse the blog entries in order. Many people use blogs to write short articles or posts that explain how to do a particular task or process. SharePoint makes it as easy to create a blog post as it is to create a Word document: simply use the ribbon to format the text and insert diagrams and photos. SharePoint blogs are also syndicated in an RSS feed, making it easy for users to follow your posts in Outlook or another feed reader. Figure 5-7 shows the default blog page created by SharePoint.

Blogging and wikis, described in the next section, go hand in hand as tools that enable users to quickly and easily participate in the organization's knowledge base. The goal of social computing in the enterprise is to reduce the barriers to users sharing information.

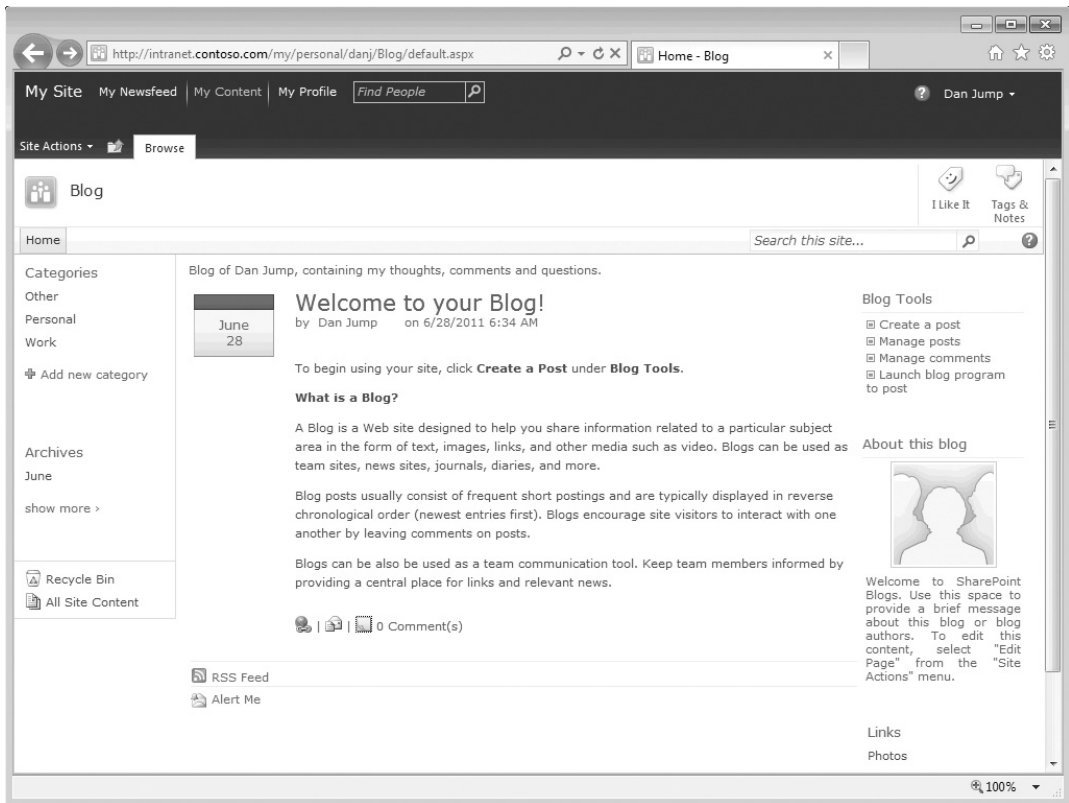


FIGURE 5-7

Wikis

Many of the pages in a SharePoint site are based on the wiki model. Wikis enable many users to edit the content of a page. User changes are automatically tracked by the system. The premise of the wiki model is that the transparency of the editors helps to police the system, ensuring that users make changes responsibly. Each user is empowered to update the information and easily create new pages on-the-fly, although changes can be rolled back or denied by the page owner. Wikis are very organic and grow based on what users determine to be the best structure for the information. A wiki is a good vehicle for collecting and sharing knowledge within the company. They often are used to document fast-changing status updates or processes. The sample wiki page in Figure 5-8 shows that wiki page editing is just rich text editing.

Traditionally, wikis have their own markup syntax; however, SharePoint can hide this from average users by providing a rich text editing experience using the ribbon. If you are a wiki power user, however, you can use standard wiki markup tags that you may have used on other wiki platforms.

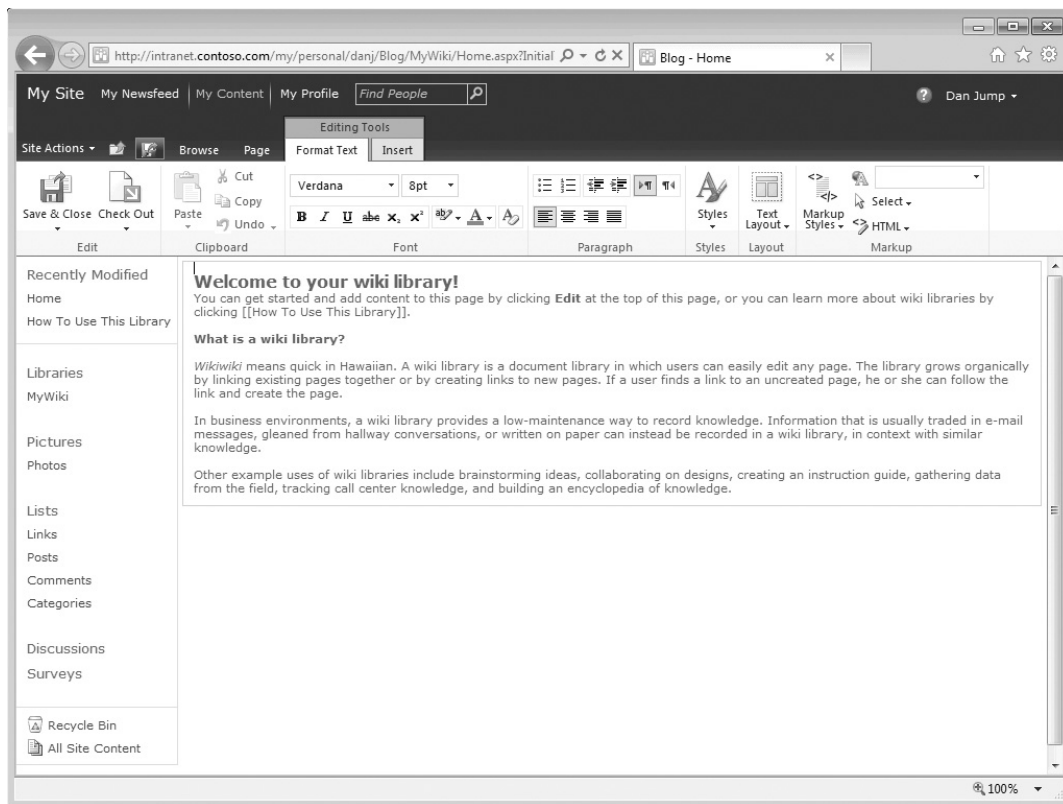


FIGURE 5-8

UNDERSTANDING THE LINKEDIN SOCIAL NETWORKING PROGRAMMING MODEL

LinkedIn is a social networking site for professionals. It provides a place to collaborate and exchange ideas with other professionals who share common interests with you. Although both LinkedIn and Facebook are centered on user profile data, Facebook traditionally focused on friends and family, whereas LinkedIn focused on professionals and building working relationships. Over the past few years, however, the lines have blurred between friends and colleagues, between fun and work. Both platforms have followed this trend, increasingly pushing into each other's space.



Everything you learn in this chapter could equally apply to SharePoint and Facebook, but this chapter focuses on SharePoint and LinkedIn. At first glance, this appears to be a more natural fit, as LinkedIn is providing many of the same features outside the firewall as SharePoint is inside the firewall.

Developers can use two APIs to create applications against LinkedIn data. The first place to start is the LinkedIn Developer Network, at <http://developer.linkedin.com>. The first API is a traditional REST-based API using OAUTH 1.0. The REST API is complete and covers all the exposed data and functionality of LinkedIn. A newer API is a JavaScript-based API (JSAPI) using OAUTH 2.0. The JSAPI is much easier to use and get started with, but it does not contain all the functionality that you may need. However, the JSAPI provides a number of ways for you to break out and call all the functions in the REST API. In an interesting twist, you will also learn how to leverage Silverlight, creating a Silverlight web part that calls functions in the JSAPI. This mixing of the three technologies is very easy to do, and it leverages the best features of each one.

LinkedIn Profiles

Like SharePoint, LinkedIn is centered on a user's profile. LinkedIn is all about who you are, who you know, and what you share. LinkedIn also has a dashboard-like home page similar to SharePoint's My Site. This page enables you to see information about what is going on in your network of friends. You can also see and manage all your profile properties from the Profile tab. Figure 5-9 shows the View Profile tab of Dan's Profile page, which includes his photo and recent status updates. Like SharePoint profiles, status updates in LinkedIn are considered micro-blogs, similar to Twitter. You can enter a brief message about your current status or anything else you want to blast out to all your friends.

Creating a LinkedIn Silverlight Web Part

One way to expose LinkedIn profile data in SharePoint is by using a Silverlight web part. Although it's possible to use OAUTH from Silverlight to interact with LinkedIn directly, doing so requires a good understanding of OAUTH. Instead, in this section you are going to leverage the JavaScript version of the LinkedIn API (JSAPI) and the HTML bridge functionality of Silverlight. This avoids needing to understand and program OAUTH, as the JSAPI handles all these details.

You will use the Silverlight Web Part VSIX project from the Visual Studio Gallery. Visit <http://bit.ly/SLWebPartVSIX> to see details about installing and using this Visual Studio extension. After you have it installed, you will create a custom Silverlight web part. This web part will enable you to put the JSAPI into the web part in addition to the Silverlight control.

1. Start by creating an empty Silverlight application and then add an empty SharePoint project to the solution.
2. Wire the Silverlight and SharePoint projects together by adding the Silverlight Custom Web Part project item template to the SharePoint project.
3. Because you are working with profile information, it would be nice to see this web part appear on your My Site. Update the Site URL property of the SharePoint project to point to your My Site.

In this example, I am playing the role of Dan Jump, the CEO of Contoso, so the Site URL would be <http://intranet.contoso.com/my/personal/danj/>. I also created a test page, `SPLinkedInProfileApp.aspx`, to host the LinkedIn web part; and to make development

easier, I set the Debug Start Action to `http://intranet.contoso.com/my/personal/danj/SitePages/SPLinkedInProfileAppWebPartPage.aspx` from the SharePoint project's property pages.

4. Press F5 in Visual Studio to launch the page.

At this point you have an empty Silverlight web part hosted on your My Site. Now let's look at how to add the JSAPI and authenticate against LinkedIn.

The screenshot displays a LinkedIn profile for Dan Jump. At the top, the LinkedIn logo and 'Account Type: Basic' are visible. Navigation tabs include Home, Profile, Contacts, Groups, Jobs, Inbox, Companies, News, and More. Below these are 'Edit Profile' and 'View Profile' buttons. The profile header shows a photo of Dan Jump, his name 'Dan Jump (YOU)', title 'CEO of Contoso', and location 'Redmond, Washington (Greater Seattle Area) | Computer Software'. A post from 8 minutes ago says 'Check out my LinkedIn Profile'. Below the post is a table of profile details:

Current	• CEO at Contoso
Past	• Officer at United States Marine Corps
Education	• Ironwood College
Connections	0 connections
Public Profile	http://www.linkedin.com/pub/dan-jump/36/51a/b43

At the bottom of the profile section are 'Share', 'PDF', and 'Print' buttons. The 'Summary' section contains a paragraph about Dan Jump's career at Contoso and his military service. The 'Specialties' section lists skills like Project Management and Leadership. The 'Experience' section lists his role as CEO at Contoso in the Computer Software industry.

FIGURE 5-9

Authenticating in LinkedIn

Authenticating against LinkedIn is very easy to do using the JSAPI. First, you need to register your application in LinkedIn, at <https://www.linkedin.com/secure/developer>, in order to generate the application API key and secret key. Once you have the keys, simply add the JSAPI `script` tag to your Silverlight Custom web part page. This page is an `.ascx` page that already contains the Silverlight object tag. Add the following `script` tag to the bottom of the page after the Silverlight control:



```
<script
  type="text/javascript"
  src="http://platform.linkedin.com/in.js">
  api_key: c148iSP2NUaMtY4fujGSzGfUyDxwZiVEf16QtekI9JoErc0g98fczpIAEYYbxxSb
  onLoad: onLinkedInLoad
  authorize: true
</script>
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

At runtime this will download the `in.js` file from the LinkedIn server and call the `onLinkedInLoad` function when the user is authenticated. If the user has already been authenticated, this function is called directly after the script is loaded. Conversely, if this is the first time the user has accessed the page, he or she will need to authenticate. The JSAPI looks for a special `div` tag placed on the page. The LinkedIn login button will be inserted into this `div` tag. Add the following `div` tag someplace on the page. Place it wherever you want the login button to appear. In this case, it is placed above the Silverlight control.



```
<!-- LinkedIn Login Button -->
<script type="IN/Login"></script>
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

Press F5 to run the page. You will see the LinkedIn login button appear just above the Silverlight control, as shown in Figure 5-10.

Clicking the login button will open a browser window for you to authenticate against LinkedIn and grant the application permission to access your account information. Figure 5-11 shows the LinkedIn authentication page.

After you authenticate, the authentication cookies will be stored in your browser and the `OnLinkedInLoaded` function will be called. On subsequent visits to the page, you will not see the login button, as the authentication cookies will automatically log you in and call the `OnLinkedInLoaded` function.

That's it. The JSAPI handles all the details of OAUTH, making the whole process very easy. One last thing to add to the page, to help with building JavaScript applications in general, is a reference to JQuery and any other script libraries that you typically use. Add the following `script` tags to the page to load the JQuery and JQuery UI libraries:

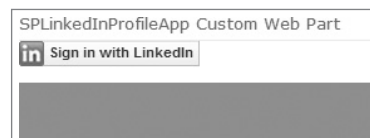


FIGURE 5-10

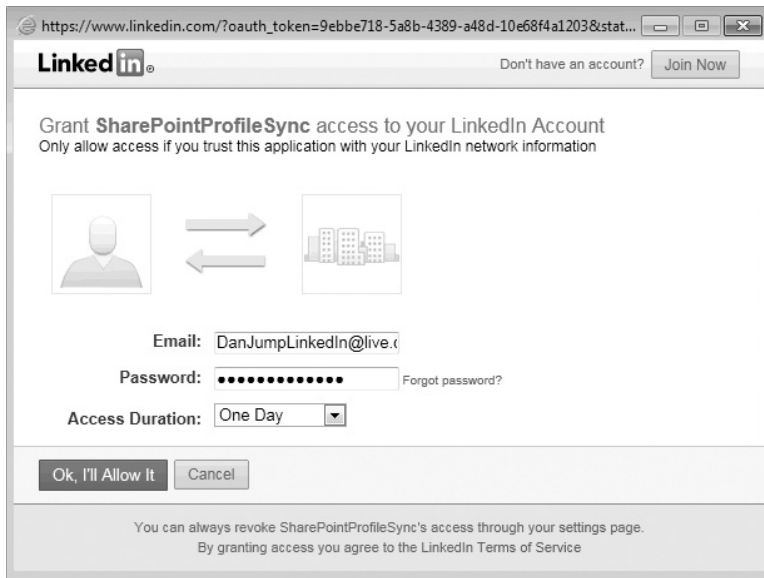


FIGURE 5-11



Available for
download on
Wrox.com

```
<script
  type="text/javascript"
  src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.6.1.js">
</script>
<script
  type="text/javascript"
  src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.8.14/jquery-ui.js">
</script>
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

After you have successfully authenticated, you have access to the JavaScript variable called `IN`. The `IN` variable is the entry point to the JSAPI.

Reading LinkedIn Profile Properties

After authenticating, you can retrieve the LinkedIn profile properties and pass them to Silverlight. Add the following XAML code to the Silverlight control to display the properties. You can copy the logo from the LinkedIn site and add to the Visual Studio project in the `images` folder to display the LinkedIn logo. It will then display the user's name, title, and profile picture.



Available for
download on
Wrox.com

```
<Grid
  x:Name="LayoutRoot"
  Background="CornflowerBlue"
  Height="480" Width="640">
  <!-- LinkedIn Profile Properties-->
  <Image
    Source="images/LinkedIn_Logo60px.png"
    Margin="8,8,0,0"
```



```

        Stretch="Fill"
        HorizontalAlignment="Left"
        VerticalAlignment="Top"
        Height="50"
        Width="60" />
<sdk:Label
    Name="FullName"
    Margin="89,72,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    FontSize="29.333"
    Width="388" />
<Image
    Name="ProfilePicture"
    Margin="8,72,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Height="75"
    Width="75" />
<sdk:Label
    Name="Title"
    Margin="89,104,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    FontSize="16"
    Width="388" />
</Grid>

```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml

You will need to create a function in Silverlight to be called from JavaScript once the call returns from LinkedIn. In the Silverlight code-behind file, `MainPage.xaml.cs`, add the following code. The key point here is adding the `ScriptableMember` attribute to the function. This is what makes this function callable from JavaScript. Also, be sure that its visibility is set to public.



Available for
download on
Wrox.com

```

[ScriptableMemberAttribute]
public void LoadProfile(string id,
                        string firstName,
                        string lastName,
                        string pictureUrl,
                        string headline)
{
    FullName.Content = firstName + " " + lastName;
    Title.Content = headline;

    //Get the profile image
    BitmapImage image = new BitmapImage();
    image.UriSource = new Uri(pictureUrl);

    ProfilePicture.Source = image;
}

```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

The previous example passes each value from JavaScript to the function in Silverlight. It is also possible to pass the full profile object without parsing out the values first. Here is what the function would look like:



Available for
download on
Wrox.com

```
[ScriptableMember]
public void LoadProfile(ScriptObject profile)
{
    string id = (string)profile.GetProperty("id");

    FullName.Content =
        (string)profile.GetProperty("firstName") +
        " " +
        (string)profile.GetProperty("lastName");

    Title.Content =
        (string)profile.GetProperty("headline");

    //Get the profile image
    BitmapImage image = new BitmapImage();
    image.UriSource =
        new Uri((string)profile.GetProperty("pictureUrl"));

    ProfilePicture.Source = image;
}
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

Both of these examples produce the same results. In Figure 5-12, you can see the user's name and title. LinkedIn returns a URL to the user's profile picture, so in Silverlight you need to create a bitmap image object to host the image.

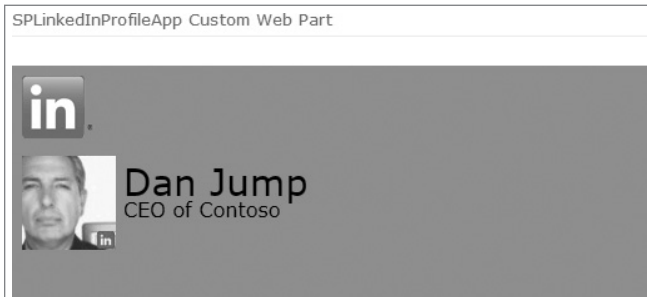


FIGURE 5-12

That was the Silverlight side of the code. Now let's take a look at the JavaScript side. Once the LinkedIn JSAPI loads, you can hook the authentication (auth) process and call the onLinkedInAuth function on the SPLinkedIn.ascx file when it completes.



Available for
download on
Wrox.com

```
//Waits for the user to log in to LinkedIn
function onLinkedInLoad() {
    // Hook the auth event
    IN.Event.on(IN, "auth", onLinkedInAuth);
}
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

In the `onLinkedInAuth` function, call LinkedIn to retrieve the user's profile information. When the profile object is returned, call the Silverlight function that you created earlier to display this on the Silverlight control. Before you can call functions in Silverlight from JavaScript, get a reference to the Silverlight control. Add the XML node, `<param name="onLoad" value="pluginLoaded" />`, to the Silverlight control in the `SPLinkedIn.ascx` file. This will cause the Silverlight control to call the `pluginLoaded` function once it has loaded. Call the `getHost` method of the Silverlight control to get a reference that you can call functions on from JavaScript:



Available for
download on
Wrox.com

```
//Get the Silverlight control
var slCtl = null;
function pluginLoaded(sender, args) {
    slCtl = sender.getHost();
}
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

You need to register the Silverlight object with the browser's object model. Call the `RegisterScriptableObject` method of the `HTMLPage` class to register it.



Available for
download on
Wrox.com

```
void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    HtmlPage.RegisterScriptableObject("MainPage", this);
}
```

code snippet 076576 Ch05_Code.zip/MainPage.xaml.cs

The last step is to actually call LinkedIn and then call Silverlight. The way to call LinkedIn functions is using the `IN.API` object. In this case, you want to return the profile, so you will use the `Profile` method, passing "me" as the parameter. This will return the profile of the currently logged-in user. You can chain together modifiers to return a specific set of fields — in this case, `id`, `firstName`, `lastName`, `pictureUrl`, and `headline`. Use the `result` function to call the Silverlight `LoadProfile` function. Note that the results are an array of objects, and because you are only expecting one, just take the first value in the array.



Available for
download on
Wrox.com

```
//Retrieve LinkedIn profile
function onLinkedInAuth() {
    var profile = "";
    IN.API.Profile("me")
        .fields(["id", "firstName", "lastName", "pictureUrl", "headline"])
        .result(function (result) {
            profile = result.values[0];
            slCtl.Content.MainPage.LoadProfile(
                profile.id,
                profile.firstName,
                profile.lastName,
                profile.pictureUrl,
                profile.headline);
        });
}
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

If you prefer to just pass the whole profile object to Silverlight, the code would look like the following function. The only difference is the signature of the call to the Silverlight `LoadProfile` function:



Available for
download on
Wrox.com

```
//Retrieve LinkedIn profile
function onLinkedInAuth() {
    var profile = "";
    IN.API.Profile("me")
        .fields(["id", "firstName", "lastName", "pictureUrl", "headline"])
        .result(function (result) {
            profile = result.values[0];
            slCtl.Content.MainPage.LoadProfile(
                profile);
        });
}
```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

Many properties are available in LinkedIn. More than 50 general profile fields are documented in detail at <http://developer.linkedin.com/docs/DOC-1061>. Many more are also defined in complex data structures, such as the publication property. The publication object contains 10 fields that describe the publication, such as authors, title, dates, and so on. As you program against LinkedIn, you will need to reference the documentation to ensure that you are using the correct fields.

Reading SharePoint Profile Properties

At this point, you have learned how to read the user profile from LinkedIn. This section looks at how to do the same using SharePoint. Add the following XAML code to your Silverlight control to display all the SharePoint profile fields and values in a list box. In this example, a button retrieves the values from SharePoint.



Available for
download on
Wrox.com

```
<!-- SharePoint Profile Fields -->
<Button
    Name="button1"
    Content="Get Profile"
    Click="button1_Click"
    Margin="12,170,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Height="23"
    Width="75" />
<ListBox
    Name="spProfileProperties"
    Height="205"
    Margin="12,200,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Width="616" />
<sdk:Label
    Name="label1"
    Content="SharePoint Profile"
    FontSize="16"
    Margin="93,169,0,0"
    HorizontalAlignment="Left"
```

```
VerticalAlignment="Top"
Width="272" />
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml

You program SharePoint user profiles using the `userprofiles.service.asmx` web service. Add a reference to the user profile service using the path `http://intranet.contoso.com/my/personal/danj/_vti_bin/userprofiles.service.asmx`. This is the path to the service endpoint under the My Site site collection for Dan Jump. You could also use the service at the root site. On the `button1` click handler, call the user profile service to retrieve Dan Jump's profile:



```
string ctx = ClientContext.Current.Url;

private void button1_Click(object sender, RoutedEventArgs e)
{
    //Create a binding and endpoint
    BasicHttpBinding binding = new BasicHttpBinding();
    EndpointAddress endpoint = new EndpointAddress(
        ctx + "/_vti_bin/userprofiles.service.asmx");

    //Create the SOAP client
    UserProfileServiceSoapClient client =
        new UserProfileServiceSoapClient(binding, endpoint);

    // Add the behavior to fix the SOAP messages
    client.Endpoint.Behaviors.Add(new AsmxBehavior());

    client.GetUserProfileByNameCompleted +=
        client_GetUserProfileByNameCompleted;

    // Get the user profile
    client.GetUserProfileByNameAsync("danj");
}
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

You can use the SharePoint `ClientContext` object to obtain the current root site and set the endpoint of the user profile web service. Most of the code is just normal web service code, with one caveat: You need to fix some of the data types returned from SharePoint that are not compatible with Silverlight. This is accomplished using the helper class called `AsmxBehavior`. Behaviors enable you to hook into the WCF pipeline to tweak the types before Silverlight sees them. Visit <http://blogs.microsoft.co.il/blogs/johnnyt/archive/2010/08/13/getting-sharepoint-s-user-properties-in-your-silverlight-4-application.aspx> to read more about using Silverlight with the SharePoint User Profile Services.

All calls in Silverlight are asynchronous, so the `client_GetUserProfileByNameCompleted` function is called when the web service call returns. The `GetUserProfileByName` function returns an `ObservableCollection` of `PropertyData` objects. Iterate over this collection to add each property and value to the list box. Note that some of the values may be empty or contain multiple values. This code checks for null values and returns either an empty string or the first item. For example, the `Interests` field is a multi-value field of interests. In this example, Dan has two interests, camping

and dogs. However, when you look at the results, you will see only Camping because you return the first item.



Available for
download on
Wrox.com

```
void client_GetUserProfileByNameCompleted(
    object sender,
    GetUserProfileByNameCompletedEventArgs e)
{
    if (e.Error == null)
    {
        ObservableCollection<PropertyData>
            profileProperties = e.Result;
        // Add each profile field and value to listbox
        foreach (UserProfileService.PropertyData propertyData
            in profileProperties)
        {
            string profilePropertyAndValue =
                String.Format("{0} {{1}}",
                    propertyData.Name,
                    (propertyData.Values.Count > 0 ?
                    propertyData.Values[0].Value : String.Empty));
        }
    }
    else
    {
        Debug.WriteLine(e.Error.Message);
    }
}
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

The result is a list box displaying all the user profile properties in SharePoint (see Figure 5-13).

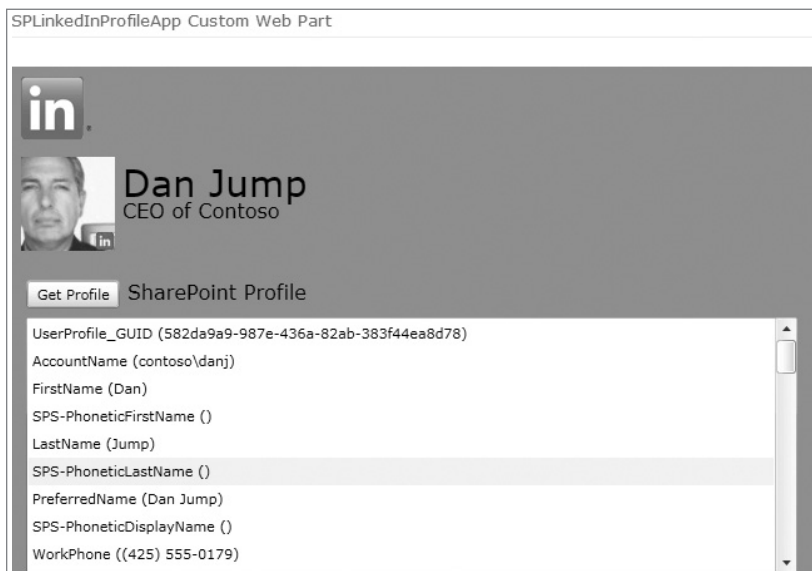


FIGURE 5-13

As shown in Table 5-1, SharePoint also provides many profile properties. In the SharePoint UI, all the properties are filled in with data. Looking at this list, you can see that many properties are empty. Some of these are filled in by other parts of the system or they may be valid empty fields. For example, the Manager field is empty because Dan is the CEO and therefore doesn't have a manager.

This table is also helpful to understand the values that are stored in the fields. Note that the field names are not very user- or code-friendly. In fact, the server API contains an enumeration class that has descriptions for each of these fields and maps them to friendlier names. This also makes it easier to code, as you don't need to remember the string name of all the fields. You will find a copy of this class in the sample project included for you to use when coding in Silverlight on the client.

TABLE 5-1: SharePoint Profile Properties

PROPERTY	VALUE
UserProfile_GUID	582da9a9-987e-436a-82ab-383f44ea8d78
AccountName	contoso\danj
FirstName	Dan
SPS-PhoneticFirstName	
LastName	Jump
SPS-PhoneticLastName	
PreferredName	Dan Jump
SPS-PhoneticDisplayName	
WorkPhone	(425) 555-0179
Department	Executive
Title	CEO
SPS-JobTitle	
Manager	
AboutMe	Dan Jump has been CEO of Contoso since 1985. Since becoming CEO of Contoso, Dan has successfully grown the company from 1 million dollars in annual sales to 2.3 billion. Dan is a graduate of MIT with an MBA, with a concentration in foreign affairs. Dan was also an officer in the U.S. Marines, doing a tour in Desert Storm. In his off time he enjoys his 4 grandchildren and his golden retriever, Zephyr.
PersonalSpace	/my/personal/danj/

continues

TABLE 5-1 *(continued)*

PROPERTY	VALUE
PictureURL	http://intranet.contoso.com:80/my/User%20Photos/Profile%20Pictures/contoso_danj_MThumb.jpg
UserName	danj
QuickLinks	
WebSite	
PublicSiteRedirect	
SPS-Dotted-line	
SPS-Peers	
SPS-Responsibility	Promoting Change
SPS-SipAddress	danj@contoso.com
SPS-MySiteUpgrade	
SPS-DontSuggestList	
SPS-ProxyAddresses	
SPS-HireDate	
SPS-DisplayOrder	
SPS-ClaimID	
SPS-ClaimProviderID	Windows
SPS-ClaimProviderType	Windows
SPS-LastColleagueAdded	
SPS-OWAUrl	
SPS-SavedAccountName	
SPS-ResourceAccountName	

PROPERTY	VALUE
SPS-ObjectExists	
SPS-MasterAccountName	
SPS-DistinguishedName	CN=Dan Jump, CN=Users, DC=contoso, DC=com
SPS-SourceObjectDN	
SPS-LastKeywordAdded	6/25/2011 12:00:00 AM
WorkEmail	danj@contoso.com
CellPhone	(425) 555-0178
Fax	(425) 555-0180
HomePhone	(425) 555-0181
Office	Seattle, WA
SPS-Location	Redmond
SPS-TimeZone	SPLinkedInProfileApp.UserProfileService.SPTimeZone
Assistant	CONTOSO\tonip
SPS-PastProjects	Process Re-engineering
SPS-Skills	C++
SPS-School	Ironwood College
SPS-Birthday	7/4/2000 12:00:00 AM
SPS-StatusNotes	Check Out my LinkedIn Profile
SPS-Interests	Camping
SPS-EmailOptin	0

Posting LinkedIn Status Messages

Another common feature of social networking sites such as SharePoint and LinkedIn is the capability to post short status messages. Status messages are displayed on the My Site of SharePoint and the home page of LinkedIn. The messages also appear in the activity feeds, enabling others to follow your status over time. Figure 5-14 shows Dan's activity on LinkedIn.

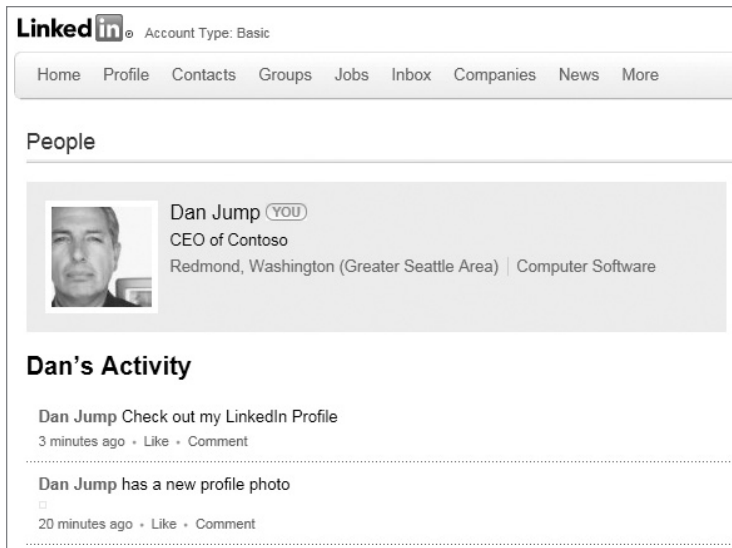


FIGURE 5-14

You saw earlier in this chapter how to call Silverlight functions from JavaScript. In this example, you learn how to call JavaScript functions from Silverlight in order to leverage the LinkedIn JSAPI to update the status in LinkedIn.

1. Update the Silverlight application to display the SharePoint status and add a button to write the status to LinkedIn. Add the follow XAML code to your Silverlight application:



```
<!-- Set Status-->
<Button
    Name="setStatusButton"
    Content="Set Status"
    Click="setStatusButton_Click"
    Margin="12,411,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Height="23"
    Width="75" />
<TextBox
    Name="MyStatus"
    Margin="96,409,0,0"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Height="48"
    Width="531" />
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml

This will add a button and a textbox to the page.

2. Update the preceding code to populate the list box with the profile properties and values with the following code:



Available for
download on
Wrox.com

```
// Add each profile field and value to listbox
foreach (UserProfileService.PropertyData propertyData
        in profileProperties)
{
    string profilePropertyAndValue =
        String.Format("{0} {1}",
            propertyData.Name,
            (propertyData.Values.Count > 0 ?
                propertyData.Values[0].Value : String.Empty));

    // Add item to List
    spProfileProperties.Items.Add(profilePropertyAndValue);

    // Get the status
    if (propertyData.Name == PropertyConstants.StatusNotes)
    {
        MyStatus.Text = (string)(propertyData.Values.Count > 0 ?
            propertyData.Values[0].Value : String.Empty);
    }
}
```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

The change in this code is to get the status value from the SharePoint profile properties. SharePoint stores the status updates in a profile property called SPS-StatusNotes. To make it easier to read and understand, you can use the enumeration value from the PropertyConstants called StatusNotes. Figure 5-15 shows Dan's status displayed on his My Site.

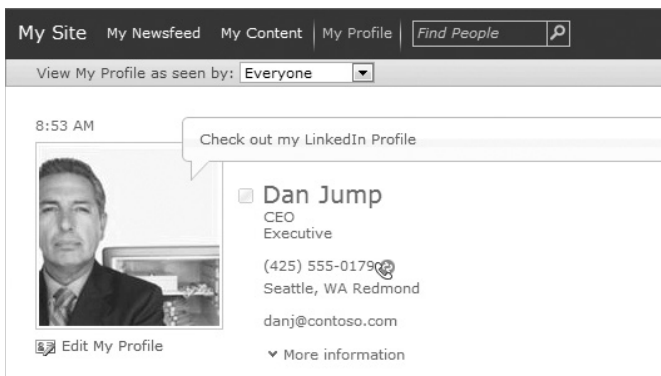


FIGURE 5-15

3. Add the following code to handle the click event of the button. This will call the `postLinkedInStatus` JavaScript function to make the update in LinkedIn to the user's profile status.

```
private void setStatusButton_Click(
    object sender, RoutedEventArgs e)
{
    // Call JavaScript function to update
    // the LinkedIn status from the SharePoint status
}
```



Available for
download on
Wrox.com

```

        HtmlPage.Window.Invoke(
            "postLinkedInStatus",
            MyStatus.Text);
    }

```

code snippet 076576 Ch05_Code.zip/ MainPage.xaml.cs

As mentioned earlier, the LinkedIn JSAPI does not cover all the functionality that the REST API covers. One of the biggest gaps is in updating your status. However, the JSAPI does have an escape hatch that enables you to access the REST API calls using the `Raw` method. The `Raw` method enables you to call any of the REST calls. The trick is that you have to code the body by hand. Next, you will see a very simple way to update the status without needing to understand how to code the message body. Add the following JavaScript function, which will be called from Silverlight to update the LinkedIn status:



Available for
download on
Wrox.com

```

function postLinkedInStatus(statusMessage) {

    IN.API.Raw("/people/~/current-status")
        .method("PUT")
        .body(JSON.stringify(statusMessage))

}

```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

The `Raw` method takes the path to the REST function call. You must also specify the type of REST call — GET, PUT, or POST. These are defined in the REST API docs at <http://developer.linkedin.com/docs/DOC-1007>. One area that can be a little complicated is passing the body to the REST call. The REST APIs can take the body as XML or JSON, but the `Raw` call only allows you to pass the body as JSON. This becomes more difficult as the body gets more complex, because all the docs show the XML body but not the JSON body. There is a help doc that explains how to convert from XML to JSON, but for new developers it's an error-prone and time-consuming process. For example, here is the XML body for updating the status:

```

<?xml version="1.0" encoding="UTF-8"?>
<current-status>is setting their status using the LinkedIn API.</current-status>

```

Another one of the rules for calling the REST API specifies that if there is only one parameter, as is the case here, you can just pass the value. In this example, you take the `statusMessage` passed from Silverlight and pass it to the `body` function, using the `JSON.stringify` function to clean up any special characters.

As with most programming, there are several ways to accomplish the same task. Another way to update the status method in LinkedIn is to use the `person-activities` function (you can view the full documentation at <http://developer.linkedin.com/docs/DOC-1009>):



Available for
download on
Wrox.com

```

function postLinkedInStatus(statusMessage) {

    activityURL = "/people/~person-activities";
    activityBODY = {
        "contentType": "linkedin-html",

```

```

        "body": "statusMessage"
    };

    IN.API.Raw(activityURL)
        .method("POST")
        .body(activityBODY.toString())
        .result(function (result) {
            alert('posted activity');
        })
        .error(function (error) {
            alert('error');
        });
}

```

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

This example is interesting because it demonstrates a more complex body that has been converted to JSON. Here is the XML format example from the documentation. There are two parameters: `content-type` and `body`. Notice that the `body` can contain an HTML-formatted message.

```

<activity locale="en_US">
<content-type>linkedin-html</content-type>
<body>&lt;a href=&quot;http://www.linkedin.com/profile?viewProfile=&amp;key=ABCDE
FG&quot;&gt;Richard Brautigan&lt;/a&gt; is reading about &lt;a
href=&quot;http://www.tigers.com&quot;&gt;Tigers&lt;/a&gt;http://www.tigers.com
&gt;Tigers&lt;/a&gt;.</body>
</activity>

```

Converting your status message to JSON becomes the following (use the conversion rules of the `Raw` function located at <http://developer.linkedin.com/docs/DOC-1287> to convert the XML to JSON):

```

{
    "contentType": "linkedin-html",
    "body": "statusMessage"
}

```



Available for
download on
Wrox.com

code snippet 076576 Ch05_Code.zip/SPLinkedIn.ascx

Finally, note that LinkedIn is throttled to limit status updates to five per day; and, like Twitter, there is a 140-character limit.

SUMMARY

SharePoint's social computing features enable you to easily collaborate with your colleagues across the boundaries of geography and internal organizational structures. Whether you are in the next department or the next state, SharePoint makes it easy to find, follow, and interact with experts and others who share your interests or skills. However, SharePoint is not an island; leveraging the synchronization capabilities built into SharePoint enables you to consume and share your profile data

with a number of internal resources. You can also leverage other social platforms in the cloud, such as Facebook and LinkedIn, to consume and share profile information to a larger Internet audience.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- **LinkedIn Developer Center** — <http://developer.linkedin.com>
- **Silverlight Web Part Extension** — <http://bit.ly/SLWebPartVSIX>
- **Social Computing in SharePoint 2010** — <http://technet.microsoft.com/en-us/sharepoint/ee263906>
- **SharePoint 2010 Social Computing Resource Center** — <http://msdn.microsoft.com/en-us/sharepoint/gg987020>

6

Using Twitter in Your SharePoint Solutions

WHAT'S IN THIS CHAPTER?

- Understanding what Twitter is and how you can use it programmatically
- Understanding OAuth applications using Twitter
- Leveraging Twitter to build an application that integrates with SharePoint

Many companies are looking at ways to integrate Web 2.0 sites and technologies into SharePoint. The popularity of Twitter makes it a prime candidate to begin to integrate this Web 2.0 experience into the collaborative experience of SharePoint. In this chapter, you'll learn about Twitter and the difference between authenticated and unauthenticated integration. More importantly, you'll get a taste of how you can get started using the Twitter application programming interfaces (APIs) to build your own Twitter-based application for SharePoint.

Before we take a look at how you integrate Twitter and SharePoint, let's first explore the basics of Twitter.

OVERVIEW OF TWITTER

You have very likely heard of Twitter before reading this chapter, as it is one of the most prominent and pervasively used social networking technologies of our recent times. If you have not heard of Twitter, it's essentially a website that offers a form of social networking — more precisely, micro-blogging services — that provides members who sign up for the service the capability to send and read messages from other members. These text-based messages are restricted to a 140-character limit and are displayed on the Twitter website, as well as routed to those who are

“following” you. (To follow someone means you have signed up to be informed every time they submit a *tweet*, or message, to Twitter.) Figure 6-1 shows the landing page of Twitter for a user called *redmondhockey*, along with a series of CBSNews tweets.

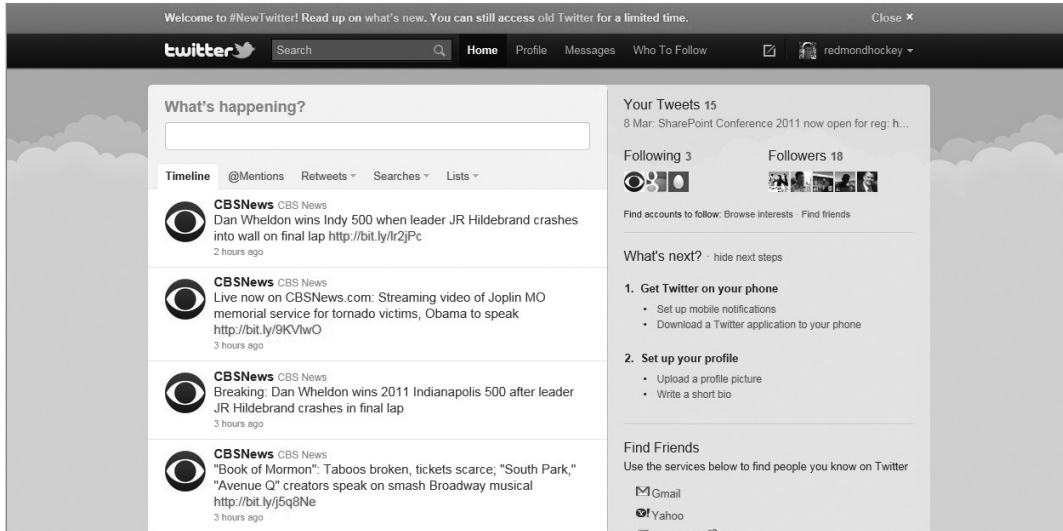


FIGURE 6-1

While it may seem pedestrian to have another social messaging framework that fits within the Web 2.0 world, there are a couple of things that set Twitter apart. The first is that it is widely used, not only as a way to transmit messages but also as a way to gauge the social conscience. For example, many marketing teams and companies are using Twitter as a way to both disseminate messages and analyze the reaction to those messages to understand what people are saying, thus thinking, about certain issues and topics. The second is that Twitter is an interesting social phenomenon because it's at the tipping point of being a central tool in news alerts and collection — both for the public and the primary news agencies. For example, when Osama Bin Laden's compound was raided in May of 2011, a software engineer was unknowingly broadcasting a series of tweets that recorded intermittent happenings around the event. This led to news tip-offs and the public having a Twitter-mediated window into what happened in Abbottabad, courtesy of this (at least initially) unsuspecting chap. Social networking and news aside, Twitter is also evolving into a platform that enables developers to build many different types of applications against it; many companies are building their own Twitter integrations that are either native to their applications or are complementary to platforms such as SharePoint.

With all this in mind, this chapter explores Twitter from a developer's perspective at a high level. You'll get a chance to walk through building a service-based application that queries Twitter for trend data — in our case, these trends will be broken out by geographical region. Specifically, you'll build a Silverlight-based web part that leverages the Trends REST API in Twitter to build an integrated application that can filter and display trends specific to certain geographical regions. To get started, let's discuss some of the fundamentals around developing applications for Twitter, paying close attention to how you factor authorization into your Twitter applications.

DEVELOPING APPLICATIONS USING TWITTER

Twitter has its own set of developer documentation and SDK, which you can find by visiting <http://apiwiki.twitter.com/w/page/22554679/Twitter-API-Documentation>. The SDKs are fairly meaty and for the most part up to date. You may find that certain areas are not documented and are left to your own devices; this is part of a quickly evolving platform. Also, the developer community around Twitter is significant and growing. There are an increasing number of blogs, third-party libraries, and other resources that will help you get started developing using Twitter.

Twitter provides a rich set of REST-based APIs that you can use to build applications. For those of you who are new to REST, it stands for Representational State Transfer and in essence uses web-based protocols such as HTTP and SOAP to enable lightweight interaction with services and data.

The REST APIs are comprehensive, offering functionality for numerous services and resources, including streaming (providing near real-time public status over a persistent connection), search, timelines, users, lists, list members and subscribers, direct messages, favorites, notifications, OAuth (Open Authorization), and much more. Each of these areas offers numerous opportunities for application development for different audiences — the data moving across Twitter is as deep as it is wide.



See the section “Additional References” at the end of the chapter for more helpful links on how to get started with Twitter development.

If you have experience developing REST-based applications, you’ll adapt very quickly to the Twitter framework. However, there is one area that is worth spending some time discussing: authorization. When you build applications that use Twitter, you can either authorize the application or user, or simply leverage the open APIs that don’t require authorization. Although you’ll build an application that does not require application authorization, it’s worth introducing how you might manage authorization with a Twitter application using OAuth.

Using OAuth in Your Twitter Applications

OAuth is an open-security protocol that supports API and application authorization for desktop and browser applications in a simple and standard way. The ultimate goal for OAuth is to provide a way to manage security tokens to enable applications to offload the authorization process to existing infrastructures, as opposed to building that infrastructure themselves. This is done through a REST-based, lightweight approach and the passing of SAML tokens (a type of XML-based security token).

Using Twitter OAuth means that you’d be redirecting your web application (either through the browser UI or programmatically through Windows Identity Foundation, which is beyond the scope of this chapter) to authenticate using Twitter’s existing login and authorization infrastructure. For example, with a simple web application, this would mean when you load the landing page, the web application redirects to a Twitter login page; you sign in and are passed back to your original web application. Thus, with regard to authorization and authentication, Twitter does the heavy lifting for you; you don’t have to maintain any identity management code in your application. This process

and implementation is done within the OAuth framework. You can find detailed information on OAuth at <http://oauth.net/>.

While you can build many types of applications, this chapter discusses two types of Twitter applications: those that require some level of authentication, and those that do not. For authentication, this chapter serves more as background. As mentioned earlier, you will actively leverage REST APIs that don't require authentication to build your solution in this chapter.

Building Applications That Require Authorization

At one time, you could authorize applications using Twitter with basic authorization — a form of authentication that simply involved passing the *username* and *password* of an application to Twitter for authorization. However, those days are now gone (i.e., Twitter no longer supports basic authentication), and now in place of basic authentication is OAuth. As described earlier, this means that your application must “redirect” to Twitter to handle the log-in process, as opposed to usernames and passwords being sent across the wire.

In this regard, the premise behind OAuth is that you can have Twitter (or another application or site) act on behalf of the person or application trying to gain access to your site or application. Furthermore, when accessing a site or application, you actually don't need the password or username of an authoritative account. Instead of using an explicit username and password, OAuth uses the SAML *tokens*, which provide limited access to a site or application. An analogy would be a card key to enter a building. Your all-access card key represents the username and password, and an OAuth token can grant you access either to specific parts of the building, say the first floor, or to all floors. The point is that you have more control over the level of authorization, and you don't expose key credential information in the process.

When you implement OAuth in your applications, you can use standard libraries across languages and platforms. Many of these libraries are open source, but all of them accomplish at least two main things. First, they obtain a token to access otherwise protected resources by asking the user to grant access (*request token*); second, they use that token to access protected resources (*access token*). At a high level, you can think of each OAuth authorization process following a similar path (assuming in this case that Twitter is our point of authorization):

1. The user loads a fictional website called ContosoRadBikes.com, but because it requires OAuth-based authorization from Twitter, it redirects to a special Twitter authorization page.
2. When the user is redirected to Twitter, ContosoRadBikes.com asks for a request token.
3. The user then types in his or her Twitter credentials. Because the user is a member of Twitter, and we're using Twitter as the source of OAuth authorization, once the user enters his or her credentials, they are trusted and can access the website.
4. After the user is successfully authorized on the login page, Twitter sends the request token to ContosoRadBikes.com.
5. After ContosoRadBikes.com receives the request token, it requests an access token from Twitter.
6. Twitter receives the request and then issues an access token. The user now has access to ContosoRadBikes.com.

This seems like a lot of back and forth, but the net result is a trusted relationship between you and ContosoRadBikes.com using Twitter as the point of authorization. And as you can see, the only place you entered any username and password information was through a Twitter-mediated login page, as opposed to it being embedded in the actual application and passed across HTTP requests.

Let's look at a more practical example with some code behind it. Suppose that you want to add an OAuth layer to a local ASP.NET site, thereby requiring any user of your site to have a token before he or she can access it. In addition, you want to use Twitter as the third party to verify the credentials of anyone who wants that access. How do you manage this relationship using OAuth — and more specifically, OAuth with Twitter?

First, you need to register an application with Twitter. You can do this by navigating to <https://dev.twitter.com/apps> and clicking “Register a new app” (see Figure 6-2).

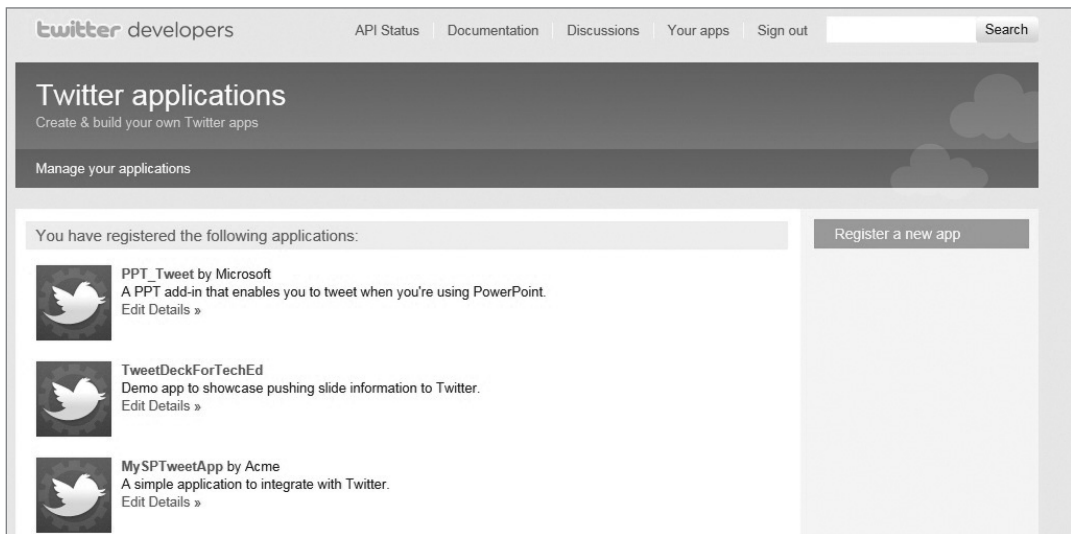


FIGURE 6-2

On the Register an Application page that appears (see Figure 6-3), you can enter information about your application, including application name, description, website, organization, application type, callback URL, default access type, and application icon. (Ensure that your default access type is read/write.) After you complete the registration, click Save Application.

When the registration of the application is saved, a number of settings are automatically generated for you. You can click Application Details to see the key elements used for OAuth: the consumer key and the consumer secret, as shown in Figure 6-4. The token URLs are standard ways to request the OAuth token. When you implement an application to use this consumer key and secret, Twitter will redirect users to a page that enables them to authenticate — this in essence represents the connection point between the application requesting authorization and Twitter (the token-granting authority). After the user authenticates, Twitter then issues the token to access the ASP.NET site.

twitter developers

API Status | Documentation | Discussions | Your apps | Sign out

Search

Register an Application

Create your own Twitter app

Tell us about your application.

Application Name:

SharePointTweetTrends

Description:

A SharePoint application that provides information on Twitter trends.

Application Website:

http://blogs.msdn.com/steve_fox

Where's your application's home page, where users can go to download or use it?

Organization:

Microsoft

Application Type:

☐ Client ☒ Browser

View your applications

FIGURE 6-3

OAuth 1.0a Settings
OAuth 1.0a integrations require more work.
Consumer key
kq[REDACTED]qQ
Consumer secret
iOm[REDACTED]2fhyFLU
Request token URL
https://api.twitter.com/oauth/request_token
Access token URL
https://api.twitter.com/oauth/access_token
Authorize URL
https://api.twitter.com/oauth/authorize
We support hmac-sha1 signatures. We do not support the plaintext signature method.

FIGURE 6-4

With the new application registered and the consumer key and secret in hand, you are now ready to implement OAuth in code. Rather than walk through an elaborate example, this chapter uses an existing solution that Shannon Whitley (a technologist who writes on many different subjects,

including OAuth) created. It's a simple ASP.NET application that you can make a couple of tweaks to and very quickly test out OAuth authorization with Twitter. You can directly download the code from this location: <http://voiceoftech.com/downloads/oauthtwitterexamplenet.zip>.

After downloading Whitley's OAuth sample application, unzip it and open it in Visual Studio 2010. (Note that Visual Studio 2010 will walk you through the conversion wizard to update the solution files.) To make the required changes to the code to implement OAuth against your Twitter application, first open the `web.config` file and add your consumer key and secret to the `appSettings` element, as per the following bolded code:

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="consumerKey" value="kqr*****qqQ"/>
    <add key="consumerSecret" value="iOrmvL*****yFLU"/>
  </appSettings>
  <connectionStrings/>
  ...
</configuration>
```

Note that by adding your consumer key and secret, you've created a connection between this application and the Twitter application you just registered. After adding the key, you may also want to change the default port 80 to another port number, as shown in Figure 6-5 (because you are likely running your local SharePoint instance on port 80). If you don't, the application may not run.

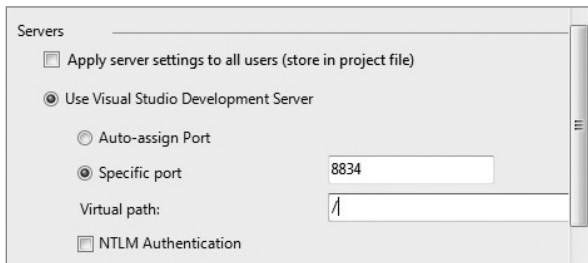


FIGURE 6-5

Lastly, you'll want to make sure that the `CallBackUrl` property is set to your local ASP.NET site. By default, it reads `http://localhost`, so you may not have to change it. The bolded line of code in the following code snippet (in `default.aspx.cs`) shows where you need to make this update:

```
...

namespace OAuthExample
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string url = "";
            string xml = "";
            OAuthTwitter oAuth = new OAuthTwitter();
```

```

        if (Request["oauth_token"] == null)
        {
            oAuth.CallBackUrl = "http://localhost";
            Response.Redirect(oAuth.AuthorizationLinkGet());
        }
        else
        {
            ...
        }
    }
}

```

After you make these changes, press F5. The application should launch and then redirect to Twitter and, using the application you registered, prompt you to sign into your site using your Twitter credentials. Figure 6-6 shows the login page that Twitter provides as a redirect from the application (and the application to which the consumer key and secret apply). After you enter your username and password, Twitter issues you an OAuth token, which enables you to access the local SharePoint site.

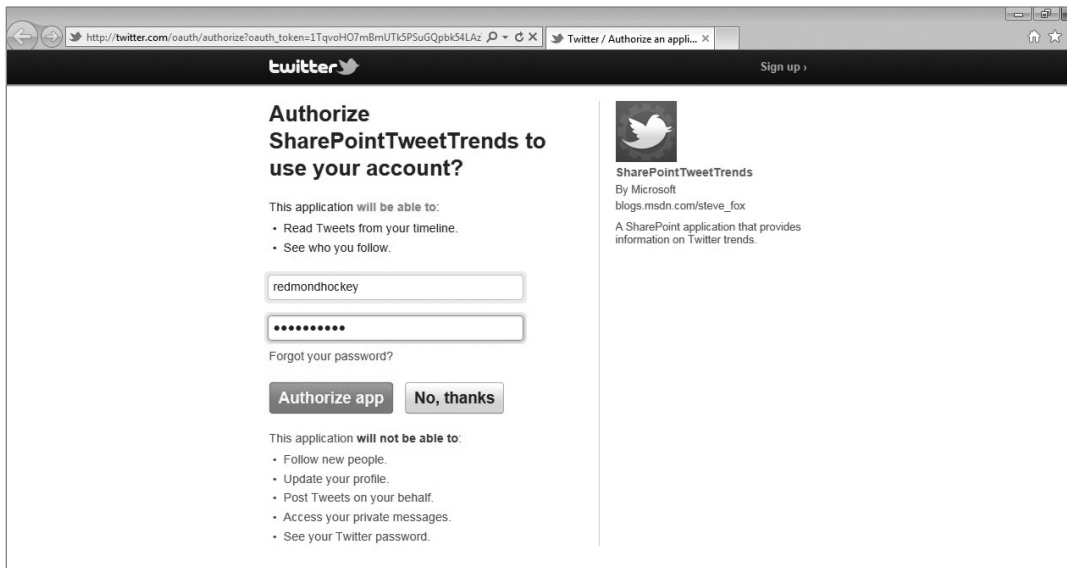


FIGURE 6-6

Note that when you're redirected to a local ASP.NET site (`http://localhost` in Figure 6-7), an OAuth token is now appended to the local site URL. This is the token issued by Twitter.

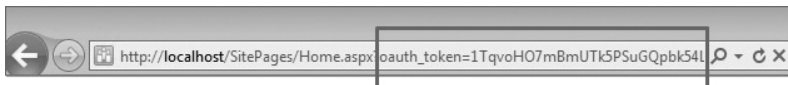


FIGURE 6-7

You can apply this standard method of authentication across many different applications, and the general process that was discussed using ContosoRadBikes.com would also apply to other sites or applications that use Twitter as the point of OAuth authorization.

Although authorization is an important aspect of Twitter application development, many other options are available to you that don't require authorization. The rest of the chapter discusses the types of applications you can build without requiring authorization, and walks you through the steps to build one using the Trends REST API.

Building Applications That Do Not Require Authorization

You can build many applications — too many, in fact — that don't require any level of authorization from Twitter. You do have to be careful, though, because your application may need to manage large amounts of data. For example, you can use the Twitter APIs to return all the tweets for a specific date or person, and with a large return data-set you'll need to manage scrolling, paging, caching, and so on, to ensure there is an optimal user experience with your application. This is less concerning for authorization and more of a concern for general application execution.

As mentioned earlier, Twitter makes heavy use of REST APIs (although you will find some .NET wrappers, such as Twitterizer, out there as well), which means that you use a URI within your application that returns some level of data that you can then parse and use in your application. In many of the Twitter REST APIs, you can opt to have your data returned in XML or JSON (JavaScript Object Notation). For those of you who have programmed .NET before, you're probably accustomed to using XML and, for example, LINQ to XML, to manage returned data. For those of you who are web developers and are, for example, using jQuery, JavaScript, or AJAX in your applications, you may prefer to use JSON. jQuery, for example, has a native set of APIs that work well with JSON.

Assuming you've signed up for a Twitter account, you can type the following REST URI into your browser and it will return information about you:

```
http://twitter.com/users/show/username_here.xml
```

The returned data for this query will be formatted similar to the following (which is truncated to optimize page space), and includes the last tweet of the username you're querying:

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
<id>35768732</id>
<name>Steve Fox</name>
<screen_name>redmondhockey</screen_name>
<location/>
<description/>
<profile_image_url>http://a3.twimg.com/profile_images/311539668/steve-fox_normal.jpg</profile_image_url> <url/>
...
<created_at>Tue Mar 08 18:23:18 +0000 2011</created_at>
<id>45187852669173760</id> <text>SharePoint Conference 2011 now open for reg:
http://www.mssharepointconference.com/Pages/default.aspx</text>
<source>web</source>
<truncated>>false</truncated>
<favorited>>false</favorited>
<in_reply_to_status_id/> <in_reply_to_user_id/> <in_reply_to_screen_name/>
```

```
<retweet_count>0</retweet_count>
<retweeted>false</retweeted>
<geo/>
<coordinates/>
<place/>
<contributors/>
</status>
  </user>
```

Likewise, you can change the format of the REST-based query to Twitter to return the same data in JSON format:

```
http://twitter.com/users/show/<username_here>.json
```

The returned data for this query will be formatted similar to the following (which, again, is truncated to save space):

```
{
  "profile_use_background_image":true,
  "protected":false,
  "profile_background_color":"C0DEED",
  "name":"Steve Fox","verified":false,
  "profile_background_image_url":"http://a3.twimg.com/images/themes/theme1/bg.png",
  ...
  "favorited":false,"truncated":false},
  "profile_background_tile":false,
  "favorites_count":0,
  "screen_name":"redmondhockey",
  "default_profile_image":false,
  "show_all_inline_media":false,
  "geo_enabled":false,
  "url":null,"id":35768732,
  "contributors_enabled":false,
  "profile_link_color":"0084B4",
  "is_translator":false,
  "created_at":"Mon Apr 27 15:25:55 +0000 2009",
  "profile_sidebar_border_color":"C0DEED",
  "followers_count":118,
  "default_profile":true,"follow_request_sent":null,
  "statuses_count":15,
  "following":null,"time_zone":null,
  "friends_count":37
}
```

Because you have the option to manage REST responses in XML or JSON, you need to decide which is appropriate for your application. In this chapter, you'll use XML. The reason we chose XML was that for .NET there is currently more support for managing XML return data than JSON, and we felt that it mapped better to Silverlight. However, you *could* use JSON parsers within Silverlight, if you choose.



If you want more background on JSON (and how it differs from XML), read the article “An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET,” at <http://msdn.microsoft.com/en-us/library/bb299886.aspx>.

Using the Twitter Trends Data

The application you’ll build is going to use the Twitter trends data, which provides trends across different pivots, such as daily and weekly trends, and trends for geographical regions. Table 6-1 provides some additional details about the currently available and common Twitter trends REST APIs. In the table, note that the URI is truncated; it does not include a prefix of `http://api.twitter.com`. For example, for the first trend (*Available*), the REST URI would be `http://api.twitter.com/1/trends/available.xml` (for XML format) or `http://api.twitter.com/1/trends.available.json` (for JSON format). Note that in the URI column in Table 6-1, the tilde (~) replaces `http://api.twitter.com`.

TABLE 6-1: Trend API

TREND	DESCRIPTION	URI
Available	Provides all available locations for trending	~/1/trends/available.format
Location	Location-specific trends	~/1/trends/woeid.format
Current	Top 10 current trending topics	~/1/trends/current.format
Daily	Top 20 trending topics per hour in a day	~/1/trends/daily.format
Weekly	Top 30 trending topics per day in a week	~/1/trends/weekly.format

Since you’ll be using the XML return format and focusing on the trends available by region, let’s take a look at what is returned when you enter the Available trends REST URI (`http://api.twitter.com/1/trends/available.xml`). The result should look similar to the following (also truncated to optimize page space):

```
<?xml version="1.0" encoding="UTF-8"?>
<locations type="array">
  <location>
    <woeid>23424969</woeid>
    <name>Turkey</name>
    <placeTypeName code="12">Country</placeTypeName>
    <country type="Country" code="TR">Turkey</country>
    <url>http://where.yahooapis.com/v1/place/23424969</url>
    <parentid>1</parentid>
```

```
</location>
<location>
  <woeid>2364559</woeid>
  <name>Birmingham</name>
  <placeTypeName code="7">Town</placeTypeName>
  <country type="Country" code="US">United States</country>
  <url>http://where.yahooapis.com/v1/place/2364559</url>
  <parentid>23424977</parentid>
...
</location>
<location>
  <woeid>2418046</woeid>
  <name>Harrisburg</name>
  <placeTypeName code="7">Town</placeTypeName>
  <country type="Country" code="US">United States</country>
  <url>http://where.yahooapis.com/v1/place/2418046</url>
  <parentid>23424977</parentid>
</location>
<location>
  <woeid>2359991</woeid>
  <name>Baton Rouge</name>
  <placeTypeName code="7">Town</placeTypeName>
  <country type="Country" code="US">United States</country>
  <url>http://where.yahooapis.com/v1/place/2359991</url>
  <parentid>23424977</parentid>
</location>
<location>
  <woeid>76456</woeid>
  <name>Santo Domingo</name>
  <placeTypeName code="7">Town</placeTypeName>
  <country type="Country" code="DO">Dominican Republic</country>
  <url>http://where.yahooapis.com/v1/place/76456</url>
  <parentid>23424800</parentid>
</location>
<location>
  <woeid>455825</woeid>
  <name>Rio de Janeiro</name>
  <placeTypeName code="7">Town</placeTypeName>
  <country type="Country" code="BR">Brazil</country>
  <url>http://where.yahooapis.com/v1/place/455825</url>
  <parentid>23424768</parentid>
</location>
</locations>
```

This feed indicates all the currently available locations for geo-specific trending. In the return data from this REST URI, you can see that the *WOEID* (that is, Yahoo's Where On Earth ID) is included. The *WOEID* is Twitter's way of representing *place* programmatically. Rather than reinvent the geographical wheel, so to speak, they borrowed an in-place ID and repurposed Yahoo's 32-bit

identifiers, which are unique and nonrepetitive — once a WOEID is assigned it never changes. This enables you to easily query a locale-specific set of tweets in the more simple case, to mashing up other data alongside the trends in a more complex scenario.

Building a geo-specific query requires that you include the WOEID within the REST URI. For example, if you take the WOEID for Turkey (which is 23424969) and then query for the top trends in this region (<http://api.twitter.com/1/trends/23424969.xml>), you'll get something similar to the following as a return data package:

```
<?xml version="1.0" encoding="UTF-8"?>
<matching_trends type="array">
  <trends as_of="2011-05-30T02:33:10Z" created_at="2011-05-30T02:29:48Z">
    <locations>
      <location>
        <woeid>23424969</woeid>
        <name>Turkey</name>
      </location>
    </locations>
    <trend query="%23orduspor"
url="http://search.twitter.com/search?q=%23orduspor">#orduspor</trend>
    <trend query="%23sikeyapiyoruz"
url="http://search.twitter.com/search?q=%23sikeyapiyoruz">#sikeyapiyoruz</trend>
    <trend query="%23burgerkingeboykot"
url="http://search.twitter.com/search?q=%23burgerkingeboykot">
      #burgerkingeboykot</trend>
    <trend query="%22Fatma+Murat%22"
url="http://search.twitter.com/search?q=%22Fatma+Murat%22">Fatma Murat</trend>
    <trend query="%22B%C3%BClent+Orta%C3%A7gil%22"
url="http://search.twitter.com/search?q=%22B%C3%BClent+Orta%C3%A7gil%22">
      Bülent Ortaçgil</trend>
    <trend query="%22Ahmet+G%C3%B6k%C3%A7ek%22"
url="http://search.twitter.com/search?q=%22Ahmet+G%C3%B6k%C3%A7ek%22">
      Ahmet Gökçek</trend>
    <trend query="%22Metin+Diyadin%22"
url="http://search.twitter.com/search?q=%22Metin+Diyadin%22">Metin Diyadin</trend>
    <trend query="%22%C3%96zge+Ulusoy%22"
url="http://search.twitter.com/search?q=%22%C3%96zge+Ulusoy%22">Özge Ulusoy</trend>
    <trend query="K%C4%B1%C4%B1%C3%A7dar%C4%9Flu"
url="http://search.twitter.com/search?q=K%C4%B1%C4%B1%C3%A7dar%C4%9Flu">
      Kılıçdaroğlu</trend>
    <trend query="%22Glee+%26+Gleeks%22"
url="http://search.twitter.com/search?q=%22Glee+%26+Gleeks%22">Glee & Gleeks</trend>
  </trends>
</matching_trends>
```

In the application that you'll build in this chapter, you'll use only a handful of the WOEIDs; however, as you explore Twitter trends further, you can flesh out your application to include other available locations. Table 6-2 provides the seven locations and accompanying WOEIDs you'll use.

TABLE 6-2: Chapter Example WOEIDs

LOCALE	WOEID
Worldwide	1
United States	23424977
Canada	23424775
Ireland	23424803
United Kingdom	23424975
Mexico	23424900
Brazil	23424768

Now that you have some background on the type of data returned from Twitter’s REST APIs, the trending-specific APIs, and the core locations you’ll be targeting in your application, let’s go ahead and create the solution.

INTEGRATING TWITTER WITH SHAREPOINT

To keep things straightforward and illustrative, this solution will mainly provide you with location-specific insights (i.e., trending) from Twitter feeds. You’ll make the feed available to SharePoint via a Silverlight-enabled web part.

The Solution Architecture

The architecture of this application will implement the REST URI within a WCF service, making it broadly available to a variety of applications. One of the key reasons for this is portability; we wanted to ensure the application worked the same in SharePoint Server (SPS) 2010 as it would in SharePoint Online. Making the application portable across the two requires the use of Silverlight (or jQuery/JavaScript), because you’re making an external service call from the web part (which is not allowed by SharePoint’s sandboxed solution architecture). Therefore, the WCF service will do the heavy lifting for you, and then you’ll implement the service in the Silverlight application, which is deployed to SharePoint. With this in mind, the architecture should look something similar to Figure 6-8.

Note that the web part (Silverlight-based) lives in SharePoint Server 2010 (or SharePoint Online). The Silverlight application calls the `GetTwitterTrends` method, which can be deployed to your local IIS or can be deployed to Windows Azure and subsequently interact with Twitter. While deploying to Windows Azure is optional, we are deploying it locally in this chapter; deploying it to Windows Azure supports the portability across SPS and SharePoint Online. (Note that Chapter 2 walked you through how to build and deploy a WCF service to Windows Azure. You can apply the same method in this chapter if you want to deploy the service to Windows Azure.)

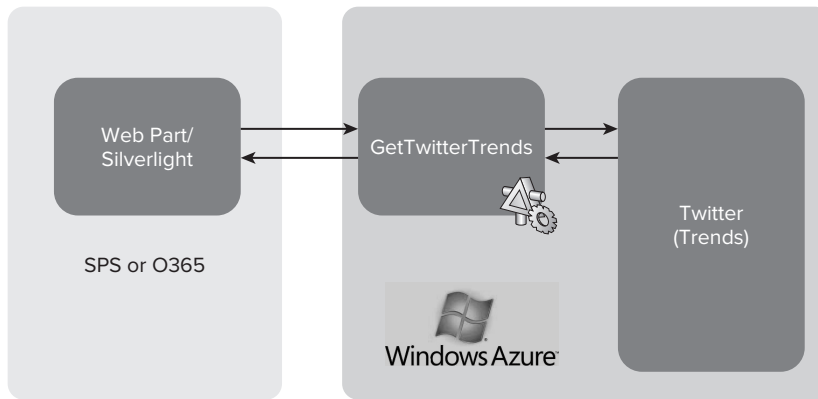


FIGURE 6-8

Creating the Twitter and SharePoint Solution

To build the application, you'll work through three primary tasks:

1. Create the service that enables you to interact with the Twitter trends reporting feed. As a part of this process, you'll need to deploy the service for implementation in the Silverlight client.
2. Create a Silverlight client application that consumes the WCF service to surface trend data.
3. Deploy the Silverlight application to SharePoint.

This basic application is fairly straightforward; however, you'll be able to use the solution as a bridge to build broader mashups and applications that leverage trend (or other) data feeds from Twitter.

Creating the Twitter Service

You have a variety of options when building applications using the Twitter REST APIs. For example, you might encapsulate the calls within an `HttpWebRequest` object and then parse the return data directly from within a client or web application. You might also abstract the call to Twitter with a WCF service; doing this enables you to re-purpose the service across multiple applications (especially if you deploy the service to Windows Azure). We chose the WCF service proxy as the path, so in this section of the chapter you'll walk through how to create the service proxy, which you'll consume later on in the Silverlight client application.

1. Open Visual Studio 2010 and click File ⇨ New Project.
2. Under Installed Templates, select Other Project Types ⇨ Visual Studio Solutions ⇨ Blank Solution.
3. Provide a name for your project (such as **MyTwitterTrendSolution**) and click OK (see Figure 6-9).

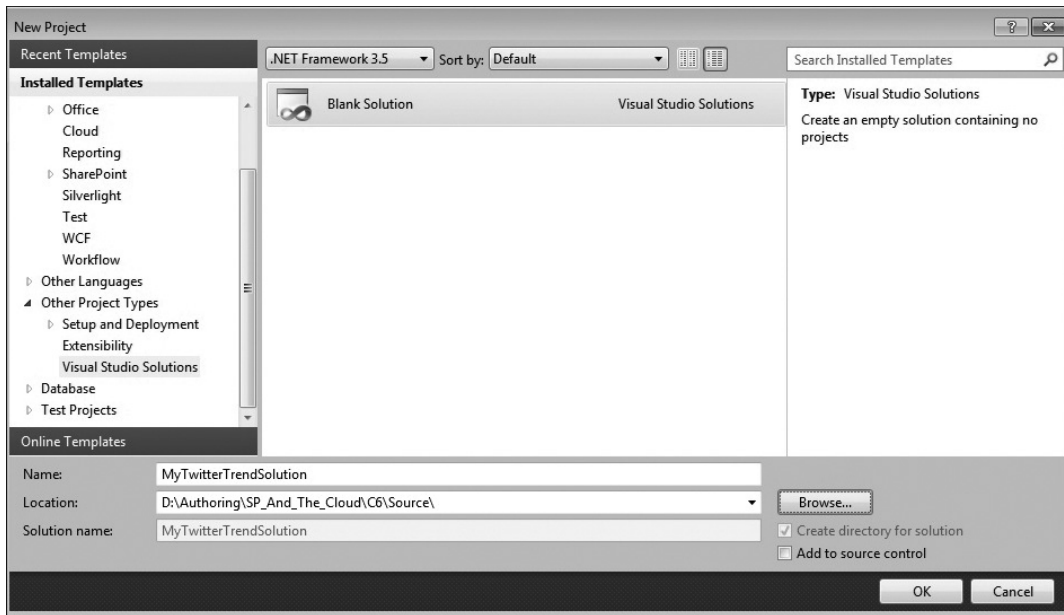


FIGURE 6-9

4. After Visual Studio creates the solution, right-click the new project and select Add ➞ New Project.
5. Select Cloud under Installed Templates and then select Windows Azure Project.
6. Provide a name for the project (e.g., **MyTwitterTrendCloudService**), and click OK (see Figure 6-10).

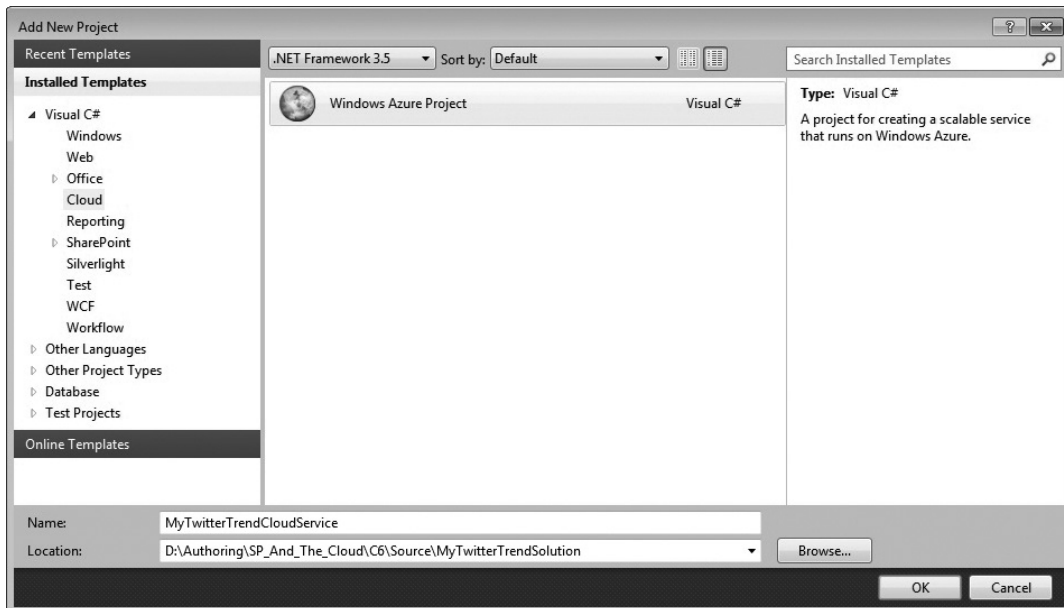


FIGURE 6-10

7. When you're prompted with the New Windows Azure Project wizard, select the WCF Service Web Role and then click the right-arrow button (>) to add the role to the project (see Figure 6-11).

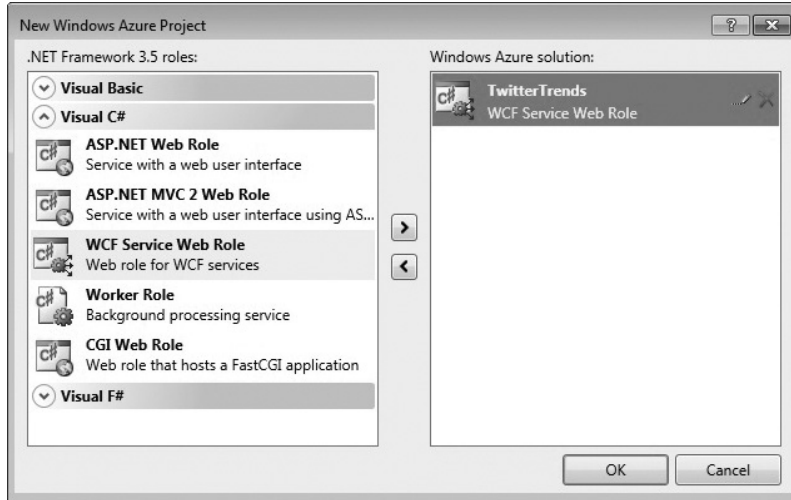


FIGURE 6-11

8. Click the small pencil icon to edit the name of the service and amend the service name to something more intuitive (e.g., **TwitterTrends**). Click OK when you are done.

At this point, you have a shell cloud project created with some default code in it. The project is a WCF service, and, even though you're using the cloud WCF service role, you do have the option to deploy the service to Windows Azure or to simply deploy and use in your local Internet Information Services (IIS). You have this option because the core service contract and code files are similar to those you would have if you were to create a WCF service application; the major difference in this project is that the cloud project provides you with a set of configuration files and additional classes that allow your code to run in Windows Azure. For example, note in Figure 6-12 that the MyTwitterTrendCloudService project has the one role you created (which corresponds to your service role). It also has a service configuration file and a service definition file — these files enable your service to run in the cloud and configure some of the options of the service runtime (e.g., number of role instances).

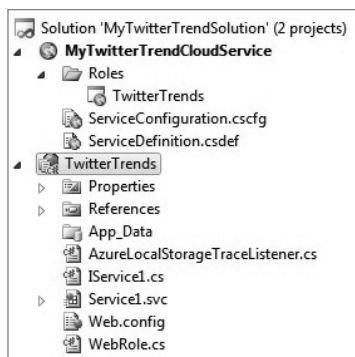


FIGURE 6-12

The following code snippet shows the service configuration file, which includes the role name, the number of instances that you want to manage in Windows Azure, and a setting to use the local development storage. Also, note that the `Instances count` is set to 2. By default, this is set to 1, which means one role running your application. However, to ensure you have failover and reliability (e.g., when machines are being patched), you should always configure this to be a minimum of 2.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceConfiguration serviceName="MyTwitterTrendCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration"
  osFamily="1"
  osVersion="*">
  <Role name="TwitterTrends">
    <Instances count="2" />
    <ConfigurationSettings>
      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
        value="UseDevelopmentStorage=true" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

The following service definition file provides additional configuration information about your application, such as endpoint settings, bindings, and local storage and logging information:

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="MyTwitterTrendCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="TwitterTrends">
    <Sites>
      <Site name="Web">
        <Bindings>
          <Binding name="Endpoint1" endpointName="Endpoint1" />
        </Bindings>
      </Site>
    </Sites>
    <Endpoints>
      <InputEndpoint name="Endpoint1" protocol="http" port="80" />
    </Endpoints>
    <Imports>
      <Import moduleName="Diagnostics" />
    </Imports>
    <LocalResources>
      <LocalStorage name="TwitterTrends.svclog" sizeInMB="1000"
        cleanOnRoleRecycle="false" />
    </LocalResources>
  </WebRole>
</ServiceDefinition>
```



```

    </LocalResources>
  </WebRole>
</ServiceDefinition>

```

In the solution, though, note that the core service (i.e., `TwitterTrends`) contains the `WebRole` class and `AzureLocalStorageTraceListener` class, which provide you with startup events that manage application tracing in the cloud.

Because you're going to be using Silverlight, you must include a cross-domain policy to enable cross-domain application communication. The cross-domain policy file is an XML file that grants web clients, such as Silverlight applications, permission to access and handle data across multiple domains. For example, when one domain makes a request for data in another domain, the cross-domain policy enables the request to be satisfied and the transaction (e.g., to get the data) to be completed. To do this, you'll add two XML files to your application: `crossdomain.xml` and `clientaccesspolicy.xml`.

9. Right-click the service project (e.g., `TwitterTrends`) and select **Add** ➤ **New Item**. Under **Data**, select **XML** and type in **`crossdomain.xml`** as the new filename.
10. In the `crossdomain.xml` file, copy and paste the following XML code:


```

<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-http-request-headers-from domain="*" headers="*" />
</cross-domain-policy>

```
11. Repeat the previous step, adding a new XML file called `clientaccesspolicy.xml`. In this new XML file, copy and paste the following XML code:


```

<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="SOAPAction">
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true" />
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>

```
12. Rename the service and interface classes to something more intuitive (e.g., `GetTwitterTrends` and `IGetTwitterTrends`). You can do this by right-clicking `Service1` in the main class file and selecting **Refactor** ➤ **Rename** (see Figure 6-13).

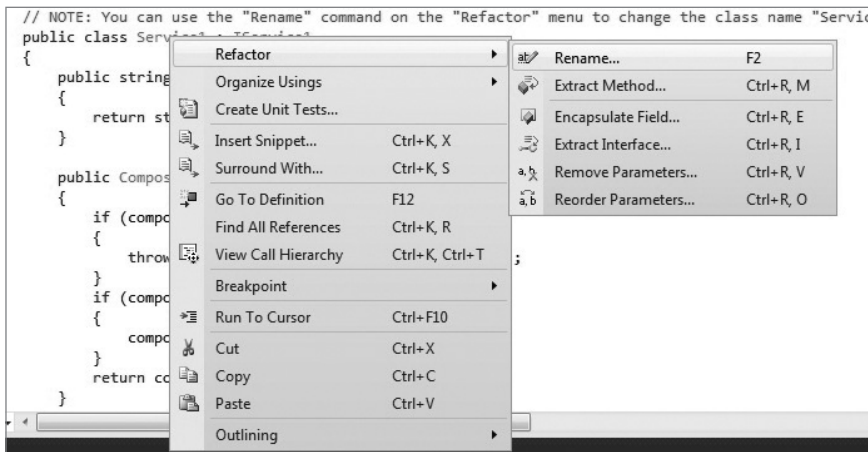


FIGURE 6-13

13. Double-click the new service interface class (e.g., `IGetTwitterTrends`) and replace the code in it with the following bolded code:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace MyTwitterService
{
    [ServiceContract]
    public interface IGetTwitterTrends
    {
        [OperationContract]
        [WebGet()]
        [XmlSerializerFormat()]
        List<Trend> GetTrends(string GeoCode);
    }
}
```

code snippet 076576 Ch06_Code.zip/MyTwitterService.csproj

This operation contract defines a method that takes a string as a parameter (which will be the location that is transformed into a WOEID on the server side) and returns a list collection of `Trend` objects, which is a simple class you'll create next.

14. Right-click the project and select Add ➞ Class. Name the class **Trend**, and then select Add. When the class has been added, amend the class code as per the following bolded code. This



is a very simple class because you'll be using only one element within the return data — the trend. However, if you want to use other parts of the return data, you can extend the properties of this class.

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

namespace TwitterTrends
{
    public class Trend
    {
        public string Name { get; set; }
    }
}
```

15. Open the core service class and replace the default code with the following bolded code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Net;
using System.Xml.Linq;

namespace TwitterTrends
{
    public class GetTwitterTrends : IGetTwitterTrends
    {
        List<Trend> listOfTrends = new List<Trend>();
        int geoCodeForTrends = 0;

        public List<Trend> GetTrends(string GeoCode)
        {
            //Method to get the correct geo-code.
            SetGeoCode(GeoCode);

            //Creates the URI for Twitter trends.
            string trendURI = "http://api.twitter.com/1/trends/" +
            geoCodeForTrends.ToString() + ".xml";
            //string trendURI = "http://api.twitter.com/1/trends/1.xml";
            //Calls the Twitter trend feed.
            WebClient clientXml = new WebClient();
            Uri urlXml = new Uri(string.Format(trendURI));
            string newsXml = clientXml.DownloadString(urlXml);

            //Transform the feed and select the child nodes of the trends
            element.

            XDocument xmlTweetTrends = XDocument.Parse(newsXml);
            XElement trendElement = xmlTweetTrends.Root.Element("trends");
            IEnumerable<XElement> trendList = trendElement.Elements();
        }
    }
}
```

```
//Create the return object (list collection) from top queries.
foreach (XElement x in trendList)
{
    Trend temp = new Trend();
    temp.Name = x.Value.ToString();
    listOfTrends.Add(temp);
}

//Dispose of the WebClient object.
clientXml.Dispose();

//Return list collection to calling application.
return listOfTrends;
}

private int SetGeoCode(string GeoCode)
{
    //Set geo code based on string from calling application.
    if (GeoCode == "Worldwide")
    {
        geoCodeForTrends = 1;
    }
    else if(GeoCode == "Canada")
    {
        geoCodeForTrends = 23424775;
    }
    else if(GeoCode == "Ireland")
    {
        geoCodeForTrends = 23424803;
    }
    else if(GeoCode == "United Kingdom")
    {
        geoCodeForTrends = 23424975;
    }
    else if(GeoCode == "Mexico")
    {
        geoCodeForTrends = 23424900;
    }
    else if(GeoCode == "Brazil")
    {
        geoCodeForTrends = 23424768;
    }
    else if(GeoCode == "United States")
    {
        geoCodeForTrends = 23424977;
    }
    else if(GeoCode == null)
    {
        geoCodeForTrends = 1;
    }
}
```

```

        //Return the set geo code to construct trend URL.
        return geoCodeForTrends;
    }
}
}

```

code snippet 076576 Ch06_Code.zip/MyTwitterService.csproj

This code primarily accomplishes three things. First, it constructs the correct REST URI based on information passed to it. That is, the `GeoCode` variable represents the location, which is then passed to the `SetGeoCode` method. The `SetGeoCode` method returns the proper WOEID, which is then used in the construction of the proper REST URI. Second, the method retrieves the location-specific trend data using the `WebClient` object. The `WebClient` class is a common way to send and retrieve data from web-based resources using URIs, so it's a straightforward way to retrieve Twitter trend data in this scenario. The data is retrieved using the `DownloadString` method, which downloads the string representation of the Twitter data. Notice that to deserialize the data, the code targets a specific set of elements (the `trend` elements) that are nested within the `trends` element.



For those of you deserializing JSON, you'll find that Twitter trend data has dynamic properties that make the structure a little tricky to manage. For example, JSON has a dynamically generated date property that makes deserialization trickier. An interesting blog post addresses the difficulty in object mapping when deserializing JSON: <http://stackoverflow.com/questions/2352536/json-twitter-list-in-c-net>.

When you've completed the deserialization of the XML data, the code then passes the generated list collection (`listOfTrends`) back to the calling application. Again, you may opt to keep the custom class (especially if you're looking at expanding the use of the class to include other trend information), or you could deserialize into an `ArrayList` object.

Now that you've built the main service for the Twitter trend data retrieval, you can deploy the service. Because you created a cloud project, you can deploy it to Windows Azure or you can deploy the service to your local IIS. If you don't have a Windows Azure account set up, deploy it locally to test the service.

1. To deploy to your local IIS, right-click the service project and select Publish.
2. In the Publish Web dialog, shown in Figure 6-14, select File System as the publish method, provide a target location (e.g., local folder), and then click Publish.
3. After you've successfully published, open IIS and then right-click Sites and select Add Web Site.



FIGURE 6-14

4. In the Add Web Site dialog (see Figure 6-15), provide a site name, add the path to the folder you published your WCF service to, and provide a port number other than 80 (which is your default port).

If you also want to explicitly set the authentication to your credentials, click Connect as and enter your domain information.

5. Click OK.



FIGURE 6-15

You should now be able to see the deployed service classes and files in your TwitterTrends site, as shown in Figure 6-16.

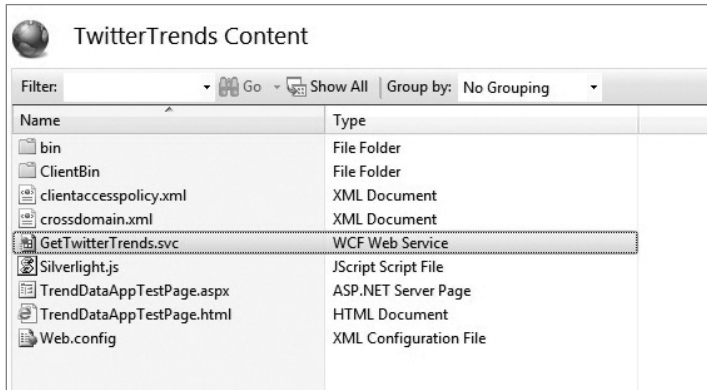


FIGURE 6-16

If you right-click the service and select Browse, you will see the service definition (and be able to copy and paste the service URL for implementation in other solutions).

If you *do* have a Windows Azure account, you can also deploy and run the WCF service from the cloud. When you publish to Windows Azure, you're essentially publishing the service to an IIS instance that is hosted on a Windows Server instance running in a data center. Therefore, to publish the service, you need to first create a hosted service in your Windows Azure Developer portal. Then you publish the service to the hosted instance of IIS instead of your local instance.

1. Navigate to <https://windows.azure.com/default.aspx> and sign in to your Windows Azure portal (see Figure 6-17).
2. Click Hosted Services ⇄ Storage Accounts ⇄ CDN. Select a subscription, a name for the service, and a region, and then you can either deploy code now to the new service (Deploy to Production) or wait until later to do it (Do Not Deploy). Select Do Not Deploy, and return to your Visual Studio solution.
3. In the Visual Studio project, right-click the Web role project and select Publish. This prompts you with the Deploy Windows Azure project dialog, as shown in Figure 6-18). Click Create Service Package Only, and click OK.
4. This prompts Visual Studio to package up the files into code and configuration files, after which Windows Explorer automatically opens to these two newly generated files. Copy the directory path to your clipboard, and return to your Windows Azure developer portal.
5. Select the service you just created, and then click New Production Deployment.
6. In the Create a new Deployment dialog, provide a deployment name and then click Browse Locally. Paste the directory path when prompted, and add the `TwitterService.cspkg` and `ServiceConfiguration.cscfg` files to your hosted service, and then click OK (see Figure 6-19).

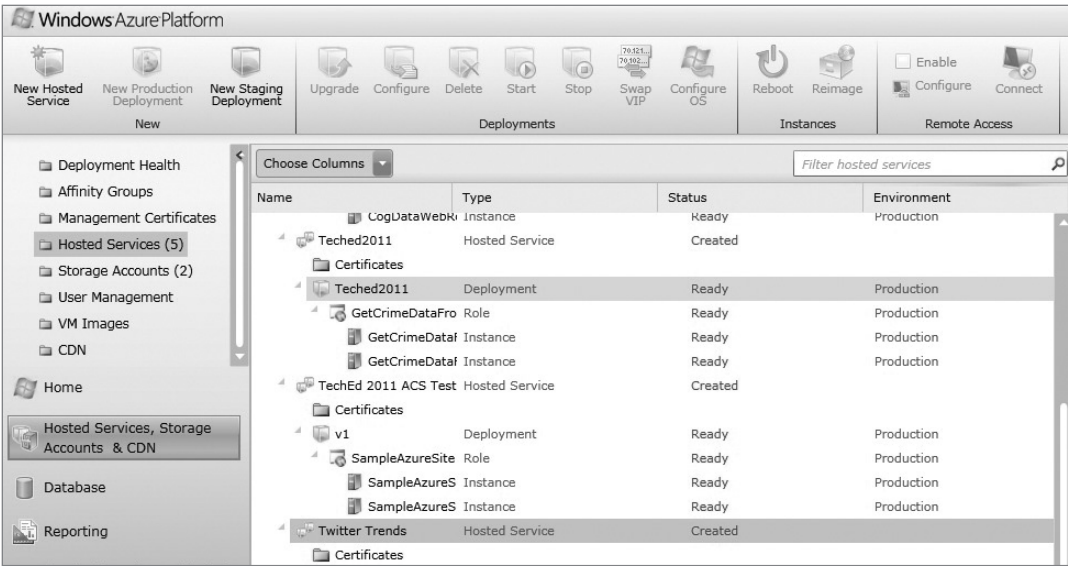


FIGURE 6-17

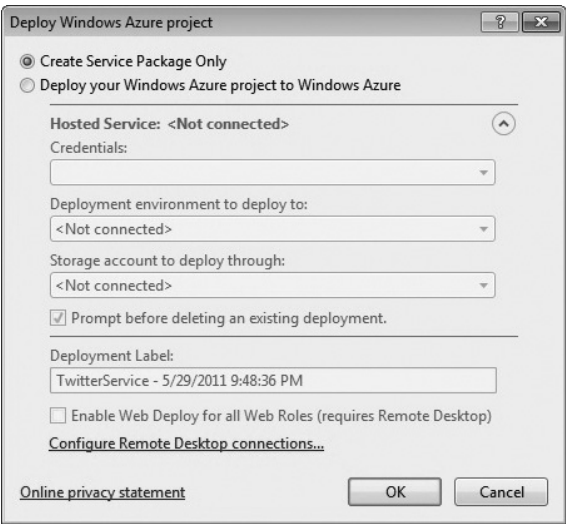
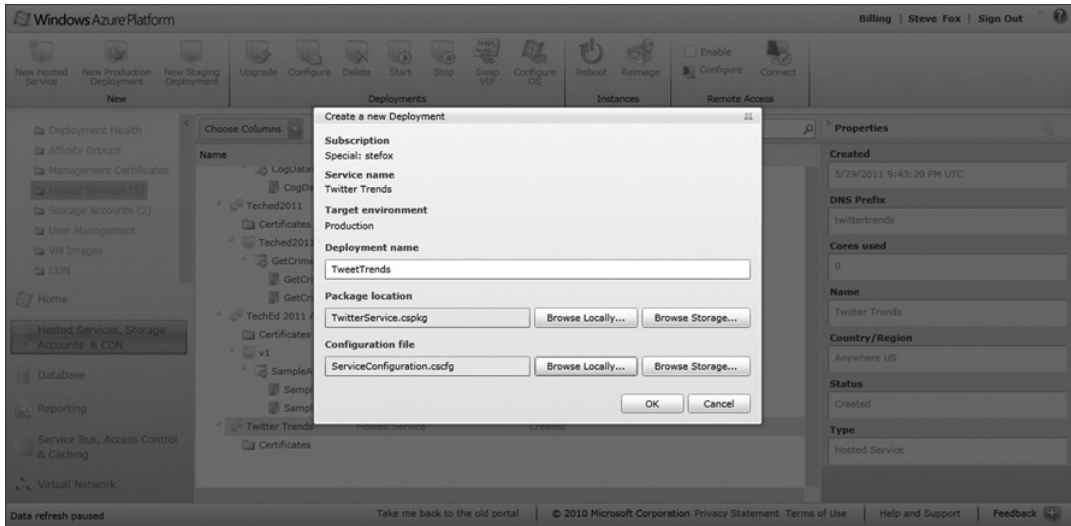


FIGURE 6-18

**FIGURE 6-19**

Uploading and preparing the role for use may take a few minutes, so you can grab a quick coffee while you're waiting. After the service code has been uploaded, you will see a Ready status, as shown in Figure 6-20.

Twitter Trends	Hosted Service	Created	
<ul style="list-style-type: none"> Certificates <ul style="list-style-type: none"> TweetTrends <ul style="list-style-type: none"> MyTwitterService <ul style="list-style-type: none"> MyTwitterService_IN_0 MyTwitterService_IN_1 	<ul style="list-style-type: none"> Deployment Role Instance Instance 	<ul style="list-style-type: none"> Ready Ready Ready Ready 	<ul style="list-style-type: none"> Production Production Production Production

FIGURE 6-20

If you select the service (e.g., TweetTrends), you can click the DNS name of the service and append it to the WCF service name, and then you should see the service definition page appear, indicating your service is ready for use. As shown in Figure 6-21, the GetTwitterTrends service is now ready for use.

There were a number of steps to create and deploy the service, but the code to do so wasn't very complex, and you now have a service that you've deployed to either your local IIS, your Windows Azure instance, or both. If you deployed the service to Windows Azure, as described earlier, you can use this service for applications that live both on-premises (SPS) and in the cloud (SharePoint Online). Obviously, your local IIS can only be used for on-premises applications.

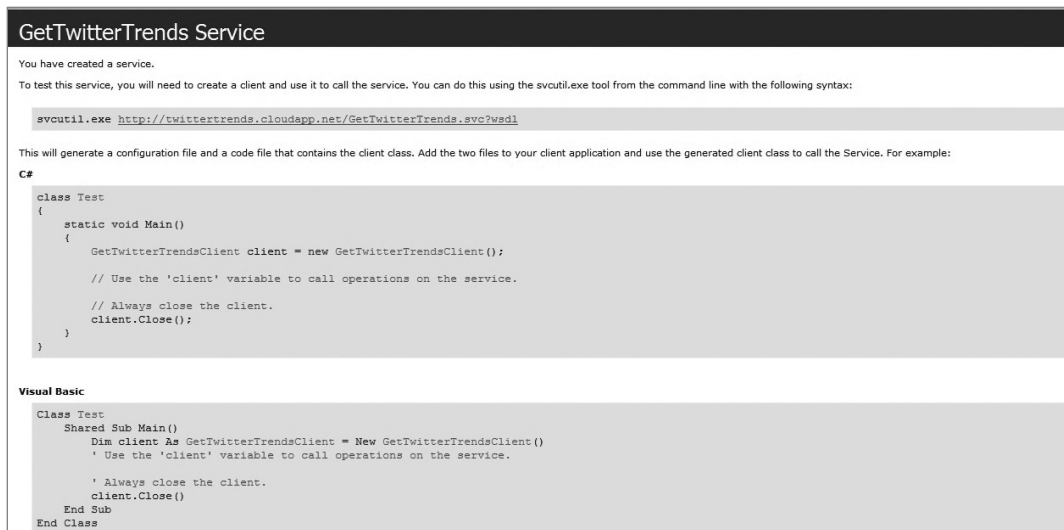


FIGURE 6-21

Creating the Silverlight Client Application

Now that you've built the service, it's time for the second part of the solution development: building the Silverlight client application that will consume the WCF service and the XML feed coming from Twitter trends.

1. Open the Visual Studio solution, right-click it, and select Add ⇄ New Project.
2. Select Silverlight under Installed Templates and then select Silverlight Application.
3. Provide a name for the project (e.g., **TrendDataApp**) and click OK. When prompted, uncheck Host the Silverlight application in a new or existing Web site in the solution, and then click OK.
4. Add the service reference to the WCF service you just deployed by right-clicking the project and selecting Add Service Reference.
5. Add the service URI (e.g., `http://twittertrends.cloudapp.net/GetTwitterTrends.svc`) to the Address field and click Go. (You can use either your locally deployed service or your Windows Azure deployed service here.)
6. Provide a namespace for the service (e.g., **GetTrendsSvc**) and click Add. You are now ready to use the WCF service you just created.
7. Right-click `MainPage.xaml` and select View in Designer. You can drag and drop the controls represented in the code snippet from the Toolbox to the Designer surface, or you can replace the XAML with the following bolded XAML code. (If you copy and paste, you may get a compile error; this is because you need to add the `System.Windows.Controls.Data.Input` library, which is added automatically when you drag and drop the controls onto the designer surface.)



Available for
download on
Wrox.com

```
<UserControl
xmlns:toolkit="http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
x:Class="TrendDataApp.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
d:DesignHeight="438" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White" Height="434">
        <Button Content="Trends" Height="23" HorizontalAlignment="Right"
Margin="0,47,27,0" Name="btnTrends" VerticalAlignment="Top" Width="75"
Click="btnTrends_Click" />
        <ListBox Height="310" HorizontalAlignment="Left" Margin="30,95,0,0"
Name="lstbxTrends" VerticalAlignment="Top" Width="342">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border Margin="5" BorderThickness="1" BorderBrush="Black"
CornerRadius="4" HorizontalAlignment="Stretch">
                        <Grid Margin="3">
                            <Grid.RowDefinitions>
                                <RowDefinition></RowDefinition>
                                <RowDefinition></RowDefinition>
                            </Grid.RowDefinitions>
                            <TextBlock x:Name="txtblkLocation" Width="200"
FontFamily="Arial" FontSize="8" FontWeight="Bold" Text="{Binding
Location}"></TextBlock>
                            <TextBlock x:Name="txtblkSearchs" Grid.Row="1"
FontFamily="Arial" FontSize="12" Text="{Binding Name}"></TextBlock>
                        </Grid>
                    </Border>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>

        <sdk:Label Height="20" HorizontalAlignment="Left" FontSize="12"
FontWeight="Black" Margin="29,12,0,0" Name="lblTitle" Content="Global Twitter Search
Trends" VerticalAlignment="Top" Width="209" />
        <ComboBox Height="23" HorizontalAlignment="Left" FontSize="8"
Margin="94,47,0,0" Name="comboboxGeos" VerticalAlignment="Top" Width="188">
            <ComboBoxItem>Worldwide</ComboBoxItem>
            <ComboBoxItem>Canada</ComboBoxItem>
            <ComboBoxItem>Ireland</ComboBoxItem>
            <ComboBoxItem>United Kingdom</ComboBoxItem>
            <ComboBoxItem>Mexico</ComboBoxItem>
            <ComboBoxItem>Brazil</ComboBoxItem>
            <ComboBoxItem>United States</ComboBoxItem>
        </ComboBox>
        <sdk:Label Height="16" HorizontalAlignment="Left" Margin="30,50,0,0"
Name="lblGeo" Content="Country:" VerticalAlignment="Top" Width="67" />
    </Grid>
</UserControl>
```

code snippet 076576 Ch06_Code.zip/TrendDataApp.cs

The UI is straightforward and includes a combo box for the seven locations selected for this application; data-bound text boxes for the location and trend within a list box; and a button to trigger the loading of the data into the list box.

After you've amended the XAML code, your UI should look similar to Figure 6-22.

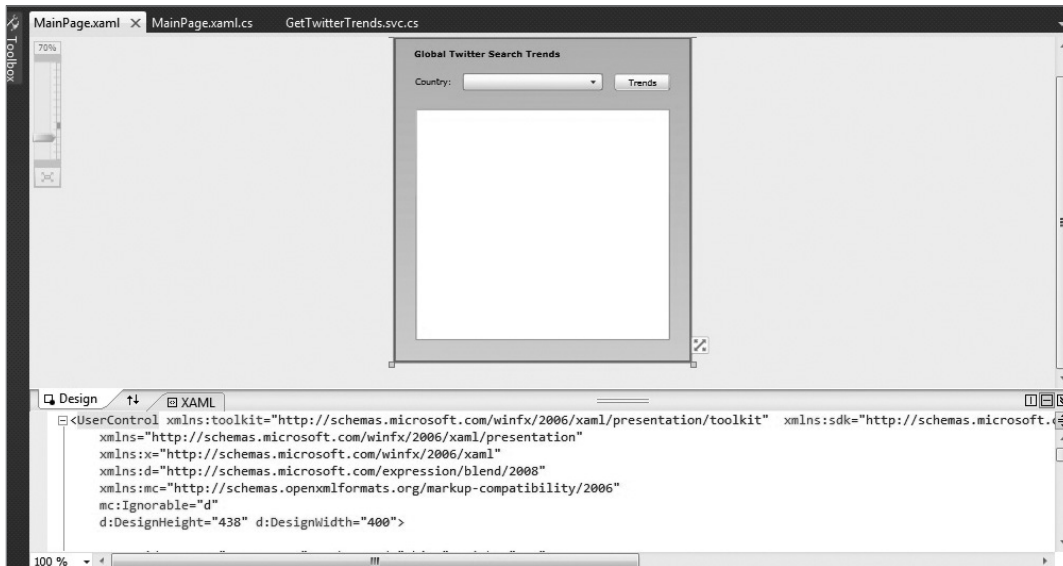


FIGURE 6-22

8. Right-click the project and select Add ➞ Class.
9. Call the new class `Trend`, and then amend the code as shown in bold:

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace TrendDataApp
{
    public class Trend
    {
        public string Location { get; set; }
        public string Name { get; set; }
    }
}
```

10. Right-click the `MainPage.xaml` file and select View Code.

11. In the code-behind (MainPage.xaml.cs), amend the code as shown in bold:



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using TrendDataApp.GetTrendsSvc;

namespace TrendDataApp
{
    public partial class MainPage : UserControl
    {
        List<Trend> listOfQueries = new List<Trend>();
        string geoFilter = "";

        public MainPage()
        {
            InitializeComponent();
        }

        private void btnTrends_Click(object sender, RoutedEventArgs e)
        {
            geoFilter = ((ComboBoxItem)comboBoxGeos.SelectedItem).Content.ToString();

            GetTwitterTrendsClient myprxy = new GetTwitterTrendsClient();
            myprxy.GetTrendsAsync(geoFilter);
            myprxy.GetTrendsCompleted += new
            EventHandler<GetTrendsCompletedEventArgs>(myprxy_GetTrendsCompleted);
        }

        void myprxy_GetTrendsCompleted(object sender, GetTrendsCompletedEventArgs e)
        {
            var resultData = e.Result;

            int i = 0;

            foreach (var item in resultData)
            {
                Trend temp = new Trend();

                if (i == 0)
                {
                    temp.Location = geoFilter;
                }
                else
                {
                    temp.Name = item.Name;
                }
            }
        }
    }
}

```

```

        }

        listOfQueries.Add(temp);

        i++;
    }

    lstbxTrends.ItemsSource = listOfQueries;

    MessageBox.Show("Done");
}
}
}

```

code snippet 076576 Ch06_Code.zip/TrendDataApp.csproj

You can see from this code that the WCF service does most of the work; the client application implements the service using a service proxy (`myproxy`), asynchronously calls the `GetTrends` method (using `GetTrendsAsync`), passing in the selected location, and then handles completion of the call with the `GetTrendsCompleted` event handler. The `GetTrendsCompleted` event handler simply iterates through the return data (`resultData`) and on the first pass sets the `Location` property of the `Trend` object to the current filter; then it sets the `Name` property to the trend being returned from the Twitter feed. The list collection (`listOfQueries`) is then data-bound to the list box, which displays the returned Twitter data (see Figure 6-23).

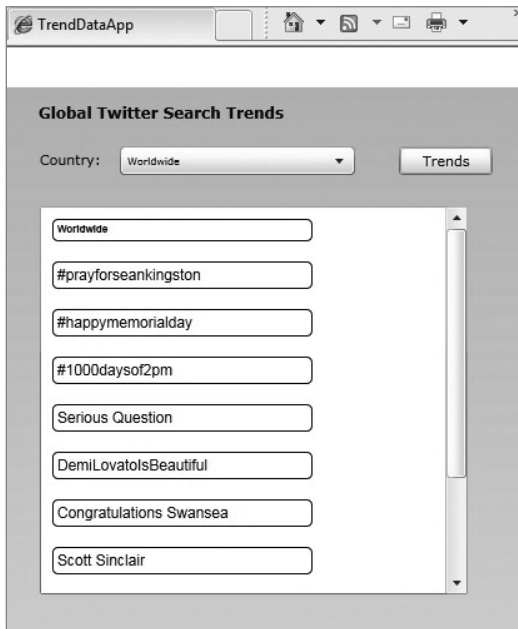


FIGURE 6-23

Deploying the Silverlight Application

Now that you've built the Silverlight application, it's time to deploy it to SharePoint. To do this, you have a couple of options. The first is more manual, and the second is automated using a VSIX project template (i.e., a Visual Studio extension). We will use the manual method in this section, as it cross-applies well to both SPS and SharePoint Online.

1. Open your SharePoint site and create a new document library called **XAPS**. To do this, click Site Actions ⇨ View all site content ⇨ Create, select Library, provide the name (XAPS) for the library, and click Create.
2. When you've created the document library, navigate to it and click Add Document. Browse to the .xap file you just built in the last exercise and upload it to the document library.
3. When it is uploaded, right-click the link and select Copy Shortcut.
4. Navigate to any web page in your SharePoint site and click Site Actions ⇨ Edit Page. Click the Insert tab and then select Web Part.
5. Select the Media and Content category, and then select the Silverlight web part and click Add.
6. Paste the shortcut to the .xap file into the URL field and then click OK. Figure 6-24 shows the new application deployed to SharePoint.

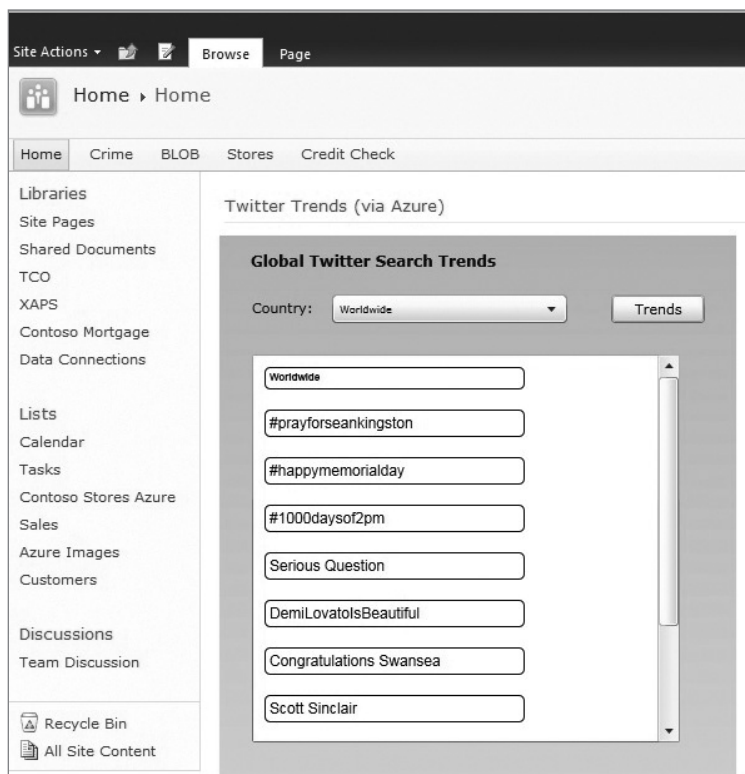


FIGURE 6-24

You added the web part manually; however, it's worth noting how to deploy the Silverlight application to SharePoint using the Silverlight and SharePoint web part. You add a Silverlight web part to SharePoint using the Silverlight web part for SharePoint VSIX project template from the Visual Studio Gallery (<http://visualstudiogallery.msdn.microsoft.com/e8360a85-58ca-42d1-8de0-e48a1ab071c7>). When you install the VSIX (Visual Studio add-in), you have the option to add a custom Silverlight Custom Web Part project template to your project. When you add this to your current solution, it automatically detects the existing Silverlight application you've built and then builds the SharePoint web part infrastructure wrapper code. The web part can then be directly deployed to SharePoint and added to a SharePoint page like any other web part.

SUMMARY

In this chapter, you learned about Twitter development at a high level. In particular, you learned that there are different types of Twitter development, some of which include authentication, and some of which that don't — and you can build very interesting applications with either of these. You also walked through the process of creating an application that uses Twitter trends and exposes location-specific trends in a Silverlight application, which was deployed to SharePoint. You learned how to use Windows Azure (for the WCF service that queries Twitter) and how you can use either the manual XAPS document library deployment or a more structured deployment using the Silverlight for SharePoint web part project template from the Visual Studio Gallery.

This simple example application illustrates how you can integrate trends from Twitter and then deploy the application into SharePoint. You can extend this application to include additional meta-data from Twitter trends, or you can add additional queries from Twitter (such as lookups on the WOEID codes instead of hard-coding them) or mashup other cloud-based applications such as Bing maps to show location, incorporate customer or locale-specific news, and many other types of integrated solutions. You could also further integrate the application into SharePoint using the SharePoint client object model (e.g., add list items into a SharePoint list).

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- ▶ **Twitter API documentation** — <http://apiwiki.twitter.com/w/page/22554679/Twitter-API-Documentation>
- ▶ **Overview of oDATA/REST** — www.odata.org/home
- ▶ **Overview of JSON** — <http://msdn.microsoft.com/en-us/library/bb299886.aspx>
- ▶ **Web part extensibility project** — <http://visualstudiogallery.msdn.microsoft.com/e8360a85-58ca-42d1-8de0-e48a1ab071c7>
- ▶ **Silverlight and SharePoint training kit** — <http://msdn.microsoft.com/trainingcourse.aspx>

7

Using Bing Maps in Your SharePoint Business Solutions

WHAT'S IN THIS CHAPTER?

- Understanding Bing Maps and geographical (geo) web services
- Using Bing Maps controls in applications
- Integrating the Bing Maps Silverlight control and geo web services in a SharePoint application

Increasingly, developers are building applications that introduce geographical (geo) data into the main user experience. With the help of geo data, this additional context (such as customer addresses within a map control or meteorological data superimposed on maps) provides improved decision-making and insight into application data. In this chapter, you'll explore how to integrate geo data into SharePoint using Bing Maps. Specifically, this chapter will provide you with an overview of Bing Maps, discuss how you can integrate the SharePoint client object model with the Bing Maps API, and then walk you through how to create a simple application that displays store information stored within a SharePoint list in a Bing Maps Silverlight control.

OVERVIEW OF BING MAPS

Over the past few years, applications have become more integrated with different types of geo data. Geo data can help inform business decisions of all types. For example, integrating geo data into the context of applications such as reporting dashboards, sales management applications, or even customer management solutions has become important to help

business professionals visualize which territories are doing well versus those that are doing poorly. Furthermore, plant and facility managers who manage multiple plants across the country, climatologists who are seeking weather patterns, or even pleasure-craft retirees all require some degree of geo data-bound knowledge.

With the growing popularity of SharePoint, many developers are asking how to integrate geo data applications with SharePoint, and which types of solutions can be built. The answer to the first question is that you can use Bing Maps as the geo service for SharePoint, and the answer to the second question is that there are many ways to integrate these two technologies.

Bing Maps is a set of services offered by Microsoft as part of the Bing suite of services. Bing Maps provides a number of geo services that can be accessed via the browser. For example, Figure 7-1 shows a number of native Bing Maps services, such as get directions, view traffic, view businesses, and so on.

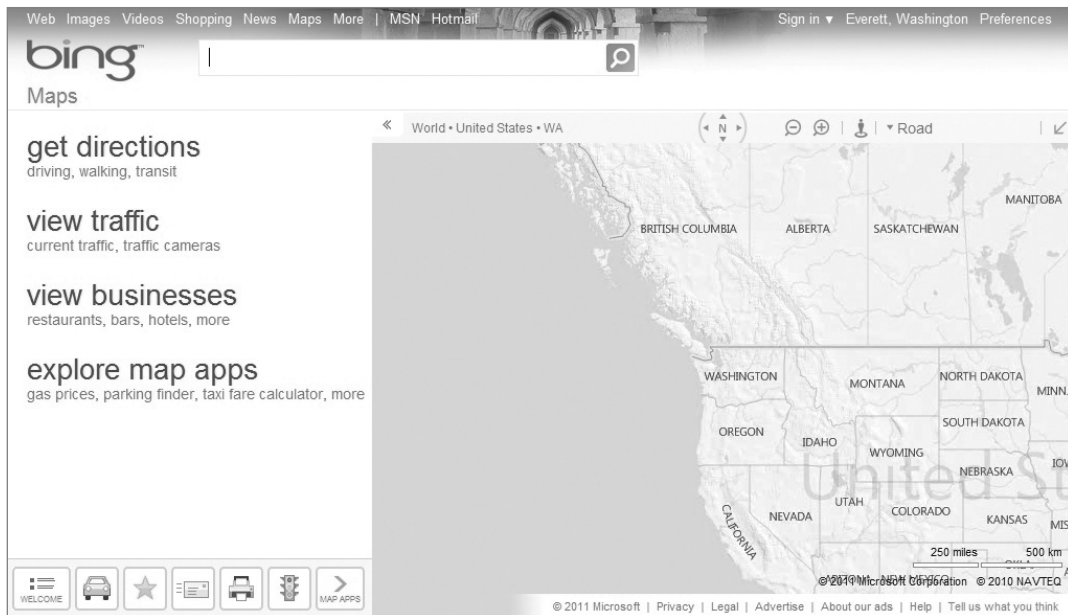


FIGURE 7-1

You can also find a number of Bing Maps Map Apps, which provide additional functionality beyond what is offered by Bing Maps. To explore the Map Apps, click the Map Apps button, which opens the Map Apps gallery (see Figure 7-2).

Given all the earthquake activity of late, one interesting application is the Earthquakes in Last 7 Days Map App, which shows current earthquake activity in the world that exceeds a Richter scale magnitude of 2.5. Figure 7-3 shows what this application looks like with a focus on North America.

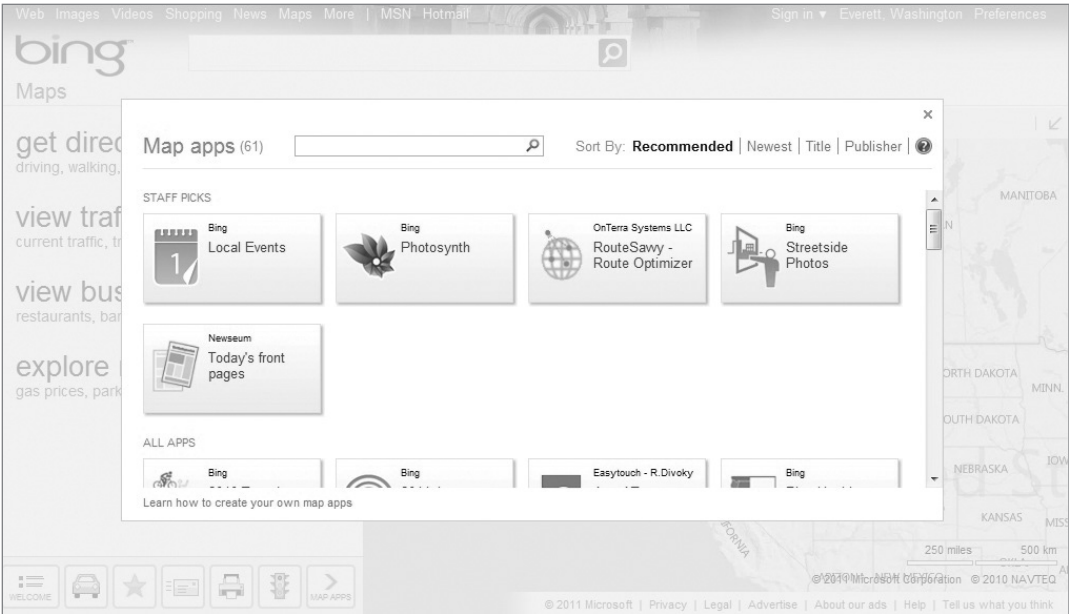


FIGURE 7-2

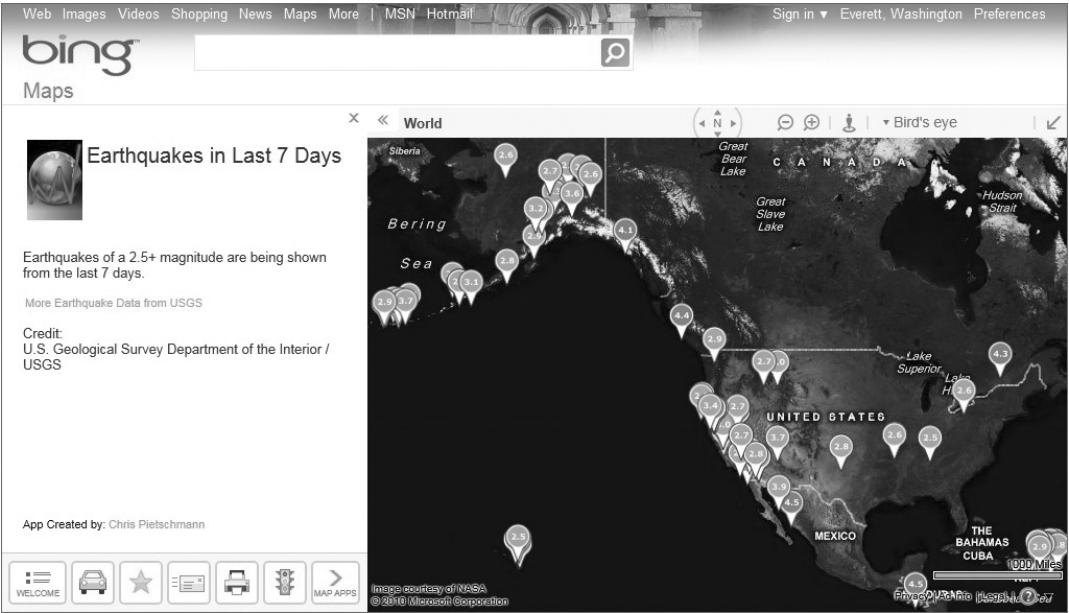


FIGURE 7-3

While there are many useful Map Apps and native browser features of Bing Maps, as a developer you can also create your own applications with a core set of flexible APIs and web services. The APIs and services that Bing Maps offers are extensive and rich. For example, you can do all of the following:

- Leverage a core set of SOAP web services to do geo look-ups on addresses.
- Use the REST services to create maps with pushpins, look up addresses, retrieve imagery metadata, and so on.
- Use Silverlight or AJAX controls to create a map that integrates with your application (or extend that map for your application).

These options are not mutually exclusive; for example, you can use the geocoding web services within a Silverlight Bing Maps control, if desired.

Using the APIs and services, you can design and build map-centric applications that use Bing Maps. For example, IDV Solutions provides a number of interesting applications that leverage Bing Maps, including one that uses automobile accident data to display the pervasiveness of accidents within a given area. As shown in Figure 7-4, different geographical areas can be specified, and you can filter based on a given time period and other metadata.



FIGURE 7-4

To help you get started with development, Bing Maps not only provides you with rich developer documentation, you also have interactive SDKs at your disposal. The interactive SDKs provide a way for you to toggle across code and previews that show application possibilities through a dynamic preview. In the code-behind, you can see, for example, HTML or XAML markup, JavaScript, or managed code. (XAML is the UI markup language that sits behind a Silverlight application.) Figure 7-5 shows the Show Me and Source Code views in the Silverlight Control Interactive SDK.

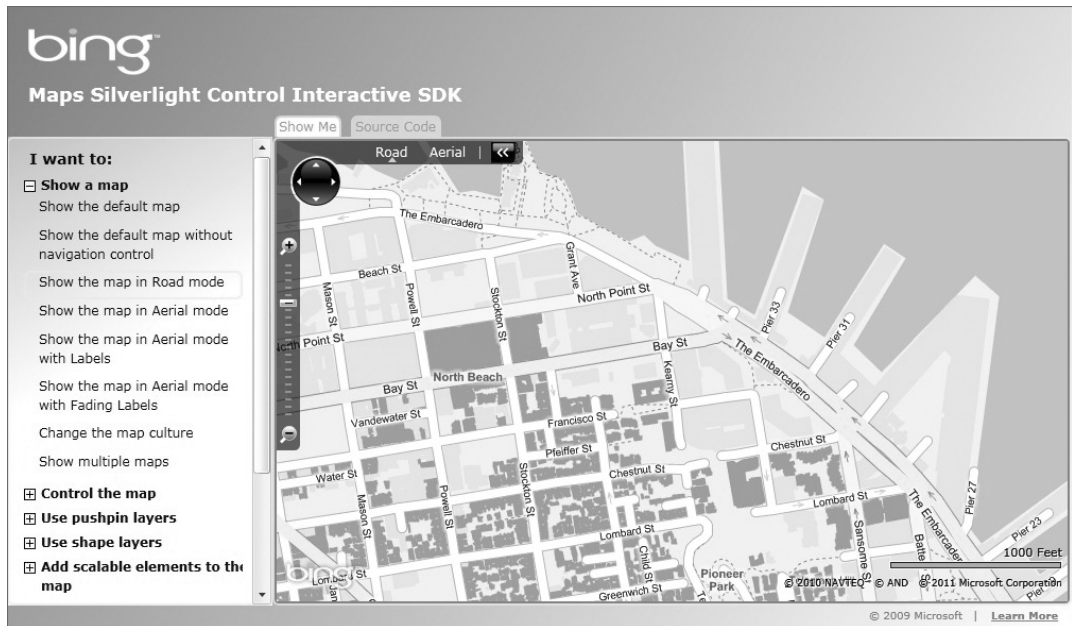


FIGURE 7-5

The default view for the Silverlight Control Interactive SDK is the Show Me view. Conversely, clicking the Source Code view displays the following XAML code-behind, which you need to have in your Silverlight application to render the map (refer to Figure 7-5):

```
<UserControl x:Class="MapControlInteractiveSdk.Tutorials.TutorialRoadMode"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:m="clr-namespace:Microsoft.Maps.MapControl;assembly=Microsoft.Maps
    .MapControl">
    <Grid x:Name="LayoutRoot" Background="White">
        <m:Map CredentialsProvider="{StaticResource MyCredentials}" Mode="Road"
        Center="37.806029,-122.407007" ZoomLevel="16" />
    </Grid>
</UserControl>
```

In the code-behind, you'll notice a couple of important things. The first is the reference to the `Microsoft.Maps.MapControl`, and the second is the `CredentialsProvider` property on the `Map` element. These represent, respectively, the Silverlight Map control, which you must download and add as a reference to your Silverlight project, and a set of developer credentials that you need to sign up for at the Bing Maps Developer site.

In this chapter, you'll use the Silverlight control to create an integrated application for SharePoint. To download this control, visit www.microsoft.com/maps/developers/web.aspx. This will get you started with the Bing Maps developer experience. You can go straight to the Silverlight control download page, if you want. The link is listed in the Additional References section in this chapter (or you could simply Bing "Bing Maps Silverlight Control."

After you download the Silverlight control, you must set up a developer account. This is very easy to do and requires only that you have a Live ID. To sign up for a Bing Maps developer account, visit www.bingmapsportal.com/. The most important item you need to build your Bing Maps application is the *developer key*; this key will be used in your application to authenticate the application against a valid ID when using the core APIs and web services. The process to set up your account takes only a few minutes.

Bing Maps Controls

You now know a little bit about Bing Maps and likely have had a chance to explore the Bing Maps sites and the Map Apps gallery. Hopefully, you've also downloaded the Bing Maps Silverlight control and set up your developer account. So, what now? Because this chapter focuses on using the Silverlight control to integrate Bing Maps with SharePoint, it's worth pointing out that you have a couple of options for Bing Maps controls. That is, you don't necessarily need to use the Silverlight control if you don't want to.

The first option is the Bing Maps AJAX control. This is a JavaScript-driven Bing Maps control that can be used in a flexible manner. If you're looking for a simple, client-side control that requires no managed code, this is the control for you. In the following code snippet, the JavaScript loads a map called `myBingMap` into a DIV. The required JavaScript library is loaded dynamically (as opposed to adding a reference to a project with the Silverlight control), and, voilà, your Bing Map Ajax control magically appears.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script type="text/javascript" src="http://ecn.dev.virtualearth.net/mapcontrol/
      mapcontrol.ashx?v=6.2"></script>
    <script type="text/javascript">
      var map = null;

      function GetMap()
      {
        map = new VEMap('myBingMap');
        map.LoadMap();
      }
    </script>
  </head>
  <body onload="GetMap();" >
    <div id='myBingMap' style="position:relative; width:600px;
      height:700px;"></div>
  </body>
</html>
```

The preceding code is not overly impressive; it merely adds an instance of the map to the page using the AJAX Bing Maps control. However, you can extrapolate how adding JavaScript functions that leverage the core AJAX APIs would enhance the Bing Maps experience. You can also integrate

seamlessly with SharePoint by using the SharePoint client object model (which supports JavaScript, Silverlight, and .NET applications). We'll explore the SharePoint client object model more deeply later in the chapter.

The second option is the Silverlight Bing Maps control. I am partial to this for three reasons. First, you can create a great user experience by virtue of Silverlight. True, it may not be fully cross-platform and cross-device, and if you're looking to maximize your device and platform compatibility (e.g., integrate with web *and* iPad), then your default choice should be the AJAX Bing Maps control. However, if your target deployment is less broad and you want to amp up your user experience, you can tap into the Silverlight capabilities to create very rich and compelling solutions.

Second, if you want to tie your solution into cross-domain platforms such as Windows Azure, leveraging the Silverlight control for this is also an easier and richer proposition, especially if you're looking at porting your application to SharePoint Online and Office 365 and storing data or services in Windows Azure.

Lastly, for those of you who want to create a great user experience *and* keep within the managed-code world, the Silverlight control should be your choice. The advantage here is that not only can you use Visual Studio for your coding and debugging experience, you can use Expression Blend as a design tool, a perfect complement.

Leveraging Geo-Location Web Services

One of the key applications of Bing Maps is geo-coding location data. You can use Bing Maps in a variety of ways to get the information you need for your solution, ranging from using a specific type of control (e.g., Silverlight Bing Maps control) to a specific API (e.g., SOAP web services). When building apps that integrate with SharePoint, you'll likely use either the REST APIs or the SOAP services.

For the SOAP services, you can use four core services. Table 7-1 lists the services, along with the service references.

TABLE 7-1: Service URLs

SERVICE NAME	SERVICE URL
Geocode Service	http://dev.virtualearth.net/webservices/v1/geocodeservice/geocode-service.svc
Imagery Service	http://dev.virtualearth.net/webservices/v1/imageryservice/imagery-service.svc
Route Service	http://dev.virtualearth.net/webservices/v1/routeservice/routeservice.svc
Search Service	http://dev.virtualearth.net/webservices/v1/searchservice/searchservice.svc

Enter the service URL in your browser to explore the service definition. If you want to test the functionality, you can create a simple console app that implements the service.



Bing Maps provides a rich set of developer documents. To get started with the Bing Maps API and services, see <http://msdn.microsoft.com/en-us/library/dd877180.aspx>.

You can also use REST to interact with Bing Maps. When using the REST URIs, you can return the results in either XML or JSON. For example, create a new Windows Forms application, add a button called `btnGet` to the form, and double-click the button to add a new event handler. If you add the following code to the code-behind, you'll see the return data as either XML or JSON. By adding `bingRestURIXML` in the `Create` method, you'll see XML as the return data, and with `bingRestURIJSON`, you'll see JSON. (If you set a breakpoint where the `MessageBox.Show` method is called, you'll be able to see the contents of `responseString`.) The major difference in the URI is the omission of `o=xml&` from the first URI, which by default returns the data as JSON:

```
...
    Uri bingRestURIXML = new Uri("http://dev.virtualearth.net/REST/v1/Locations/US/WA/98052/Redmond/1%20Microsoft%20Way?o=xml&key=<your bing developer key here>");

    Uri bingRestURIJSON = new Uri("http://dev.virtualearth.net/REST/v1/Locations/US/WA/98052/Redmond/1%20Microsoft%20Way?key=<your bing developer key here>");
    string responseString = string.Empty;

    public Form1()
    {
        InitializeComponent();
    }

    private void btnGet_Click(object sender, EventArgs e)
    {
        HttpWebRequest req = WebRequest.Create(bingRestURIXML) as HttpWebRequest;
        using (HttpWebResponse resp = req.GetResponse() as HttpWebResponse)
        {
            StreamReader reader = new StreamReader(resp.GetResponseStream());
            responseString = reader.ReadToEnd();
        }
        MessageBox.Show("done");
    }
}
```

...

Although the REST services are more lightweight than the Windows Communication Foundation (WCF) services (custom services you would build, deploy, and manage yourself), you need to parse the return data (whether XML or JSON) to use the specific data points that are returned.

INTEGRATING THE BING MAPS API WITH SHAREPOINT

In the application you'll build in this chapter, you'll use the Bing Maps Silverlight control. You can integrate the Silverlight control with SharePoint in a couple of ways. For example, you can use the native Silverlight web part, which is by far the easiest way to deploy a Silverlight application to

SharePoint, or *wrap* the Silverlight application with a web part and deploy as a SharePoint Solution Package (WSP) — a special build package of your application that is native to SharePoint. With the latter, you have more granular control over your web part (see the “Additional References” section for links to more information).

Within the Silverlight application that you deploy to SharePoint, you can use one of two ways to push and pull data from SharePoint. You can use the SharePoint client object model or you can use the native web services that ship with SharePoint. The web services are good (and out of the box), but in some cases you’ll need to manage complex XML return data objects — this can be tricky to parse. The client object model, especially for the Silverlight or Bing Maps AJAX controls, provides a more efficient and streamlined way to interact programmatically with SharePoint.

Leveraging the Client Object Model

The SharePoint client object model (SP COM) is a remote API that enables you to work with SharePoint from a .NET, Silverlight, or JavaScript application. For Bing Maps, this works well and gives you options for the AJAX and Silverlight Bing Maps controls.

If you’ve used the SharePoint server object model, you’ll recognize a lot of the syntax; it’s similar in structure. For example, as shown in the following code snippet, you first set up the context with your SharePoint site. This code snippet configures the context explicitly using the SharePoint site URL, but you can also set it dynamically using the `Current` property.

```
...
private void Connect()
{
    context = new ClientContext("http://blueyonderdemo");
    context.Load(context.Web);
    context.Load(context.Web.Lists);
    stores = context.Web.Lists.GetByTitle("Store Sales");
    context.Load(stores);
    context.Load(stores.RootFolder);
    var camlQuery = new CamlQuery();
    camlQuery.ViewXml = "<View/>";
    storeItems = stores.GetItems(camlQuery);
    context.Load(storeItems);
    context.ExecuteQueryAsync(OnLoadItemsSucceeded, onQueryFailed);
}
...
```

One of the key differences in the SP COM (as opposed to the SharePoint server object model) is that you batch process the communication with the SharePoint site. This provides users with a more efficient and manageable experience when interacting with SharePoint. For example, as shown in the preceding code snippet, you create the `context` object and then issue a number of methods and properties against that object. In this case, you load the `Web` (SharePoint site), and a specific list called `Store Sales` from the SharePoint site, and create a generic CAML query (roughly equivalent to a `SELECT *` in SQL) to return all the list items. You then execute all the batched properties and methods by calling the `ExecuteQuery` method (in .NET applications) or the `ExecuteQueryAsync` method (in Silverlight and JavaScript applications). Because the preceding code snippet is executed within a Silverlight application, it uses the `ExecuteQueryAsync` method, which has two delegates as parameters — one that executes on success and another on fail.

THE SOLUTION ARCHITECTURE

Given that you are going to create an application in the space of one chapter, we want to keep the solution architecture fairly straightforward. That said, you'll be creating an application that will be deployable on either SharePoint Server 2010 or SharePoint Online in Office 365. With this parity in mind, you need to architect your solution with an external service call mediated from a Silverlight application (because you cannot execute an external service call from a sandboxed solution in SharePoint Online).

The architecture of the solution is outlined at a high level in Figure 7-6. Note that the Bing Maps Silverlight control will be integrated with a wider Silverlight application and deployed to SharePoint. The Silverlight application will not only leverage the Bing Maps Silverlight control, but also use the SOAP service to do a dynamic geo-lookup for latitude and longitude. The Silverlight application will then interact with a SharePoint list in two ways:

- It will enable the addition of user-entered data and the look-up from the Bing SOAP service.
- It will support a read from that same list to display the store and sales information in a data-bound control.

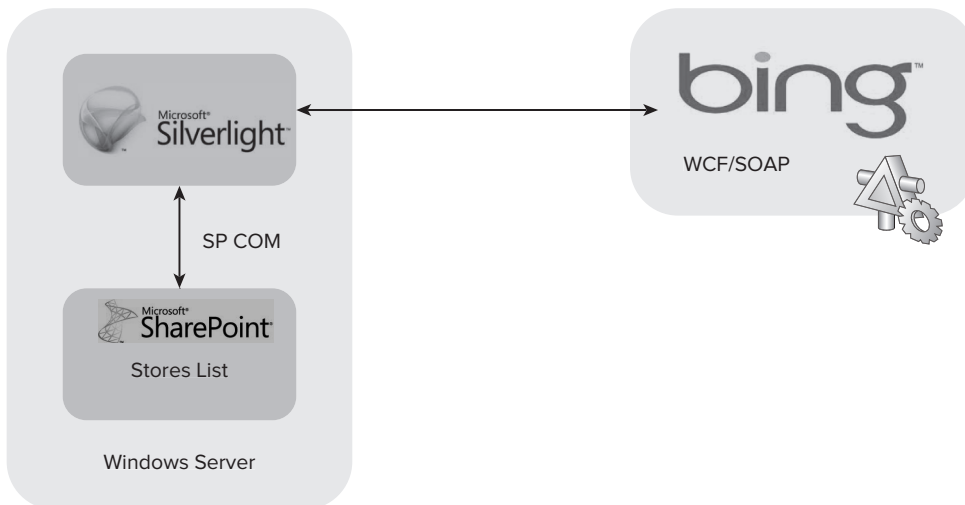


FIGURE 7-6

With the architecture in hand, let's begin coding the solution.



Before you start, be sure to download the Silverlight Bing Maps control and obtain your Bing Maps developer key.

CREATING A SHAREPOINT SOLUTION USING BING MAPS

Assuming you have downloaded the Silverlight control and signed up for a Bing Maps developer account, you are ready to open Visual Studio 2010 and create a new Silverlight application that will house the Bing Maps and SharePoint integration. This is the first step in creating the Bing Maps solution.

The solution you'll create in this chapter will enable a user to do four main things:

- Read a list into a data-bound list box
- Do a geo-code look-up for the latitude and longitude of a specific address
- Programmatically add a new store (list item) to a SharePoint list
- Add a pushpin for a specific store to a Bing map

This solution will leverage the SharePoint client object model to interact with a SharePoint site and integrate the Bing Maps functionality using the Silverlight Bing Maps control and SDK.

Let's get started building the application!

Creating the Bing Map Application

First, you need to create a new Silverlight application to display the Bing Maps application.

1. Open Visual Studio 2010.
2. Select File ⇨ New Project.
3. In the New Project dialog that appears, under Installed Templates, click Silverlight ⇨ Silverlight Application, and then provide a name for the project (e.g., BingMapSolution) and a location. When you are done, click OK (see Figure 7-7).
4. Accept the default options in the New Silverlight Application dialog, and click OK.
5. After the Silverlight project has been created, right-click it and select Add Reference.
6. Click the Browse tab and then browse to `Microsoft.Maps.MapControl.dll`. (You should find it in `C:\Program Files (x86)\Bing Maps Silverlight Control\V1\Libraries`.) You need to add this DLL to the project to access the core Bing Maps APIs. Click OK to add it to the project.
7. Double-click the `MainPage.xaml` file and amend the XAML code as per the following bolded code:

```
<UserControl x:Class="BingMapSolution.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```

mc:Ignorable="d"
xmlns:m="clr-namespace:Microsoft.Maps.MapControl;assembly=Microsoft.Maps
.MapControl"
d:DesignHeight="300" d:DesignWidth="400">
<Grid x:Name="LayoutRoot" Background="White">
    <m:Map x:Name="MainMap" CredentialsProvider="<Bing Map Developer Key Here>"
        AnimationLevel="Full"
        Mode="Aerial"
        Center="47.620574,-122.34942"
        ZoomLevel="16">
        <m:Pushpin Location="47.620574,-122.34942" />
    </m:Map>
</Grid>
</UserControl>

```

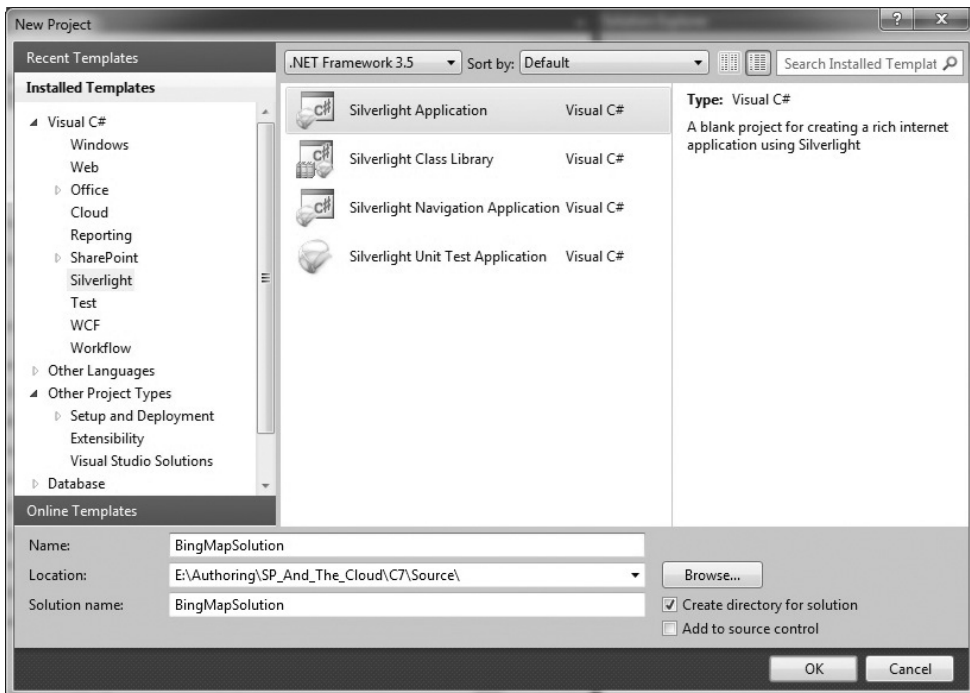


FIGURE 7-7

When you've added the preceding XAML code, your application should now look like Figure 7-8.

8. Press F6 to build the project. When the project builds successfully, press F5 to debug it. Figure 7-9 shows an aerial shot of the specified location.

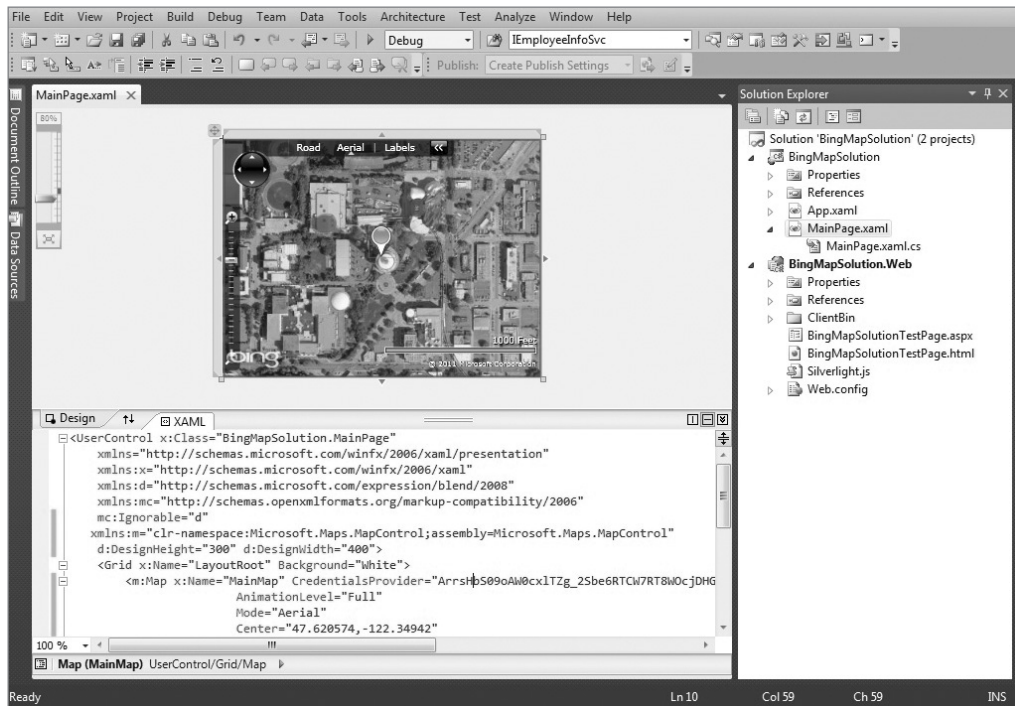


FIGURE 7-8



FIGURE 7-9

At this point, you have a working, albeit simple, application that exposes the Bing Maps map with a center point of 47.620574 latitude and -122.34942 longitude, which points to the Space Needle in Seattle, Washington. However, to map the solution to the architecture, you need some sort of data-management capability. This means you need a way to do the following:

- Input data
- Read the data into the Bing Maps application

For an application like this, you have a number of data-management options. For example, if you're planning to deploy this application to an on-premises instance of SharePoint, you could leverage SQL Server, SharePoint list data, or a locally (in-domain) deployed web service. If you were deploying this to SharePoint Online, you would not be able to access the on-premises SharePoint list data or SQL Server without an intermediary service bus, such as the Windows Azure AppFabric service bus. You could do this, but you could also keep things a little more lightweight and store the data in SQL Azure or in a SharePoint list in SharePoint Online. For our purposes, because we want an application that deploys to both SharePoint Server (that is on-premises) or SharePoint Online, we'll choose the SharePoint list as our source of data management.



At the time of writing, SharePoint Online does not yet support SQL Azure and Business Connectivity Services (BCS). However, by the time you are reading this book, you should have BCS support in SharePoint Online. Thus, you'll be able to integrate SQL Azure data with SharePoint Online using BCS to create an external list. You can then use the external list as the data input to the Silverlight application.

Let's go ahead and extend the user interface in the Silverlight application to enable data entry into the list and create the store information list in SharePoint.

Creating the Application UI

Follow these steps to extend the simple Bing Maps UI to include a set of other controls that will enable users to add and view SharePoint list information (which represent store locations within the Pacific Northwest in the United States):

1. Open the Visual Studio solution and double-click the `MainPage.xaml` file.
2. Amend the earlier XAML code as per the following bolded code. Note that if you copy and paste, you'll need to add `System.Windows.Controls.Data.Input` to the project, but if you drag and drop the controls from the Toolbox, this library will be automatically added to the project for you.



Available for
download on
Wrox.com

```
<UserControl xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
x:Class="BingMapSolution.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
```

```

xmlns:m="clr-namespace:Microsoft.Maps.MapControl;assembly=Microsoft.Maps.
MapControl"
d:DesignHeight="700" d:DesignWidth="884">
  <Grid x:Name="LayoutRoot" Width="872" Height="694">
    <Grid.Background>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="LightBlue" Offset="1"/>
        <GradientStop Color="White" Offset="0"/>
      </LinearGradientBrush>
    </Grid.Background>

    <Grid.RowDefinitions>
      <RowDefinition/>
    </Grid.RowDefinitions>

    <ListBox x:Name="lstStores" Margin="13,49,585,0" Height="247"
VerticalAlignment="Top">
      <ListBox.ItemTemplate>
        <DataTemplate>
          <Border Margin="5" BorderThickness="2" BorderBrush="DarkBlue"
CornerRadius="4" HorizontalAlignment="Stretch">
            <Grid Margin="3">
              <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
              </Grid.RowDefinitions>
              <TextBlock x:Name="txtbxStoreName" Width="250"
FontFamily="Arial" FontSize="10" FontWeight="Bold" Text="{Binding StoreName}"></
TextBlock>
              <TextBlock x:Name="txtbxStoreAddress" Grid.Row="1"
FontFamily="Arial" FontSize="8" Text="{Binding StoreAddress}"></TextBlock>
              <TextBlock x:Name="txtbxStoreCity" Grid.Row="2"
FontFamily="Arial" FontSize="8" Text="{Binding StoreCity}"></TextBlock>
              <TextBlock x:Name="txtbxStoreState" Grid.Row="3"
FontFamily="Arial" FontSize="8" Text="{Binding StoreState}"></TextBlock>
              <TextBlock x:Name="txtbxStoreZip" Grid.Row="4"
FontFamily="Arial" FontSize="8" Text="{Binding StoreZip}"></TextBlock>
              <TextBlock x:Name="txtbxStoreLat" Grid.Row="5"
FontFamily="Arial" FontSize="8" Text="{Binding StoreLatitude}"></TextBlock>
              <TextBlock x:Name="txtbxStoreLong" Grid.Row="6"
FontFamily="Arial" FontSize="8" Text="{Binding StoreLongitude}"></TextBlock>
            </Grid>
          </Border>
        </DataTemplate>
      </ListBox.ItemTemplate>
    </ListBox>

    <m:Map x:Name="MainMap" CredentialsProvider="<insert your Bing developer key
here>"
      AnimationLevel="Full"
      Mode="Aerial"

```



```

        Center="47.620574,-122.34942"
        ZoomLevel="16" Margin="301,49,27,42">
    <m:MapLayer x:Name="StorePushpins"/>
    <m:Pushpin Location="47.620574,-122.34942" />
</m:Map>
<Button Content="Add Store" Height="23"
    HorizontalAlignment="Left" Margin="60,647,0,0"
    Name="btnGetStores" VerticalAlignment="Top" Width="75"
Click="btnGetStores_Click" />
<Button Content="Get Stores" Height="23" HorizontalAlignment="Left"
    Margin="59,312,0,0" Name="btnAddStores"
    VerticalAlignment="Top" Width="75" Click="btnAddStores_Click" />
<sdk:Label Content="Store Locations" Foreground="Black" FontFamily="Arial
Black" Height="28"
    HorizontalAlignment="Left"
    Margin="15,21,0,0" Name="lblStores" VerticalAlignment="Top"
Width="120" />
<sdk:Label Content="New Store" Foreground="Black" FontFamily="Arial Black"
Height="28"
    HorizontalAlignment="Left" Margin="15,354,0,0" Name="lblNewStore"
VerticalAlignment="Top" Width="120" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,381,0,290"
Name="txtbxStoreName" Width="187" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,410,0,0"
Name="txtbxStoreCity" VerticalAlignment="Top" Width="187" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,439,0,0"
Name="txtbxStoreAddress" VerticalAlignment="Top" Width="187" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,468,0,0"
Name="txtbxStoreState" VerticalAlignment="Top" Width="187" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,497,0,0"
Name="txtbxStoreZipCode" VerticalAlignment="Top" Width="187" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="83,526,0,0"
Name="txtbxStoreSales" VerticalAlignment="Top" Width="187" />
<sdk:Label Height="20" Content="Name:" HorizontalAlignment="Left"
Margin="15,388,0,0" Name="lblStoreName" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="Address:" Height="20" HorizontalAlignment="Left"
Margin="15,419,0,0" Name="lblStoreCity" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="City:" Height="20" HorizontalAlignment="Left"
Margin="15,449,0,0" Name="lblStoreAddress" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="State:" Height="20" HorizontalAlignment="Left"
Margin="15,474,0,0" Name="lblStoreState" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="Zip Code:" Height="20" HorizontalAlignment="Left"
Margin="15,503,0,0" Name="lblStoreZipCode" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="Sales:" Height="20" HorizontalAlignment="Left"
Margin="15,532,0,0" Name="lblStoreSales" VerticalAlignment="Top" Width="62" />
<sdk:Label Content="Store Map" FontFamily="Arial Black" Foreground="Black"
Height="28" HorizontalAlignment="Left" Margin="293,21,0,0" Name="label1"
VerticalAlignment="Top" Width="120" />
<Button Content="Zoom In" Height="23" HorizontalAlignment="Left"
Margin="157,312,0,0" Name="btnZoom" VerticalAlignment="Top" Width="75" Click="btnZoom_
Click" />

```



```

        <Button Content="Clear" Height="23" HorizontalAlignment="Left"
Margin="157,647,0,0" Name="btnClear" VerticalAlignment="Top" Width="75"
Click="btnClear_Click" />
        <TextBox IsEnabled="True" Height="23" HorizontalAlignment="Left"
Margin="83,581,0,0" Name="txtbxLatitude" VerticalAlignment="Top" Width="187" />
        <TextBox IsEnabled="True" Height="23" HorizontalAlignment="Left"
Margin="83,610,0,0" Name="txtbxLongitude" VerticalAlignment="Top" Width="187" />
        <sdk:Label Content="Latitude:" Height="20" HorizontalAlignment="Left"
Margin="15,587,0,0" Name="lblLatitude" VerticalAlignment="Top" Width="62" />
        <sdk:Label Content="Longitude:" Height="20" HorizontalAlignment="Left"
Margin="15,616,0,0" Name="lblLongitude" VerticalAlignment="Top" Width="62" />
        <CheckBox Content="Look Up Geo-Code" Height="16" HorizontalAlignment="Left"
Margin="14,557,0,0" Name="chkbxLatLong" Checked="chkbxLatLong_Checked"
VerticalAlignment="Top" Width="143" />

    </Grid>
</UserControl>

```

code snippet 076576 Ch07_Code.zip/BingMapSolution.cs

The XAML code is quite lengthy, and there are some items that are worth calling out. One is the formatting of the list box, which uses the `Grid.RowDefinition` property to provide some structure to the list box. The list box is data-bound using a set of text blocks located in specific rows within the `Grid`. The `MainMap` Bing Maps object also contains one `MapLayer`, which provides a layer within the map for adding items such as pushpins or polygon overlays. You have four buttons, each with a corresponding event:

- The `btnGetStores_Click` event retrieves the store list items and then populates the list box using the SharePoint client object model.
- The `btnAddStores_Click` event takes user-entered information from the text boxes and, assuming the `chkbxLatLong` checkbox is checked, will do a look-up to retrieve the latitude and longitude for the address entered.
- The `btnClear_Click` event clears the text boxes for data reentry.
- The `btnZoom_Click` event takes the latitude and longitude from the selected item in the list box and creates a `Location` object, which is then used to refocus the Bing Maps map on the specified store location.

When you've added the new XAML code to `MainPage.xaml`, your application should look similar to Figure 7-10.

Now that you've created the core UI for the Bing Maps application, you need to add the service reference that will be used to retrieve the latitude and longitude from the address entered by the end user. The URL for this service is <http://dev.virtualearth.net/web-services/v1/geocodeservice/geocodeservice.svc>. Go ahead and open your Internet browser and type in this address to ensure you can see the service. You should see something similar to Figure 7-11.

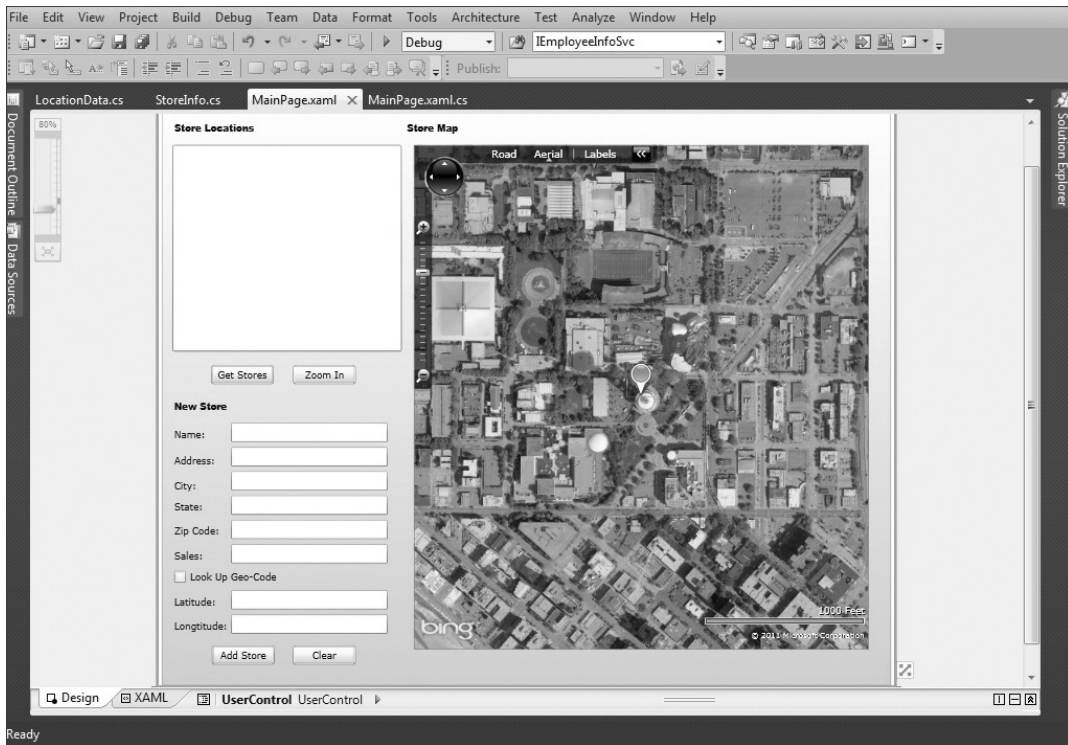


FIGURE 7-10

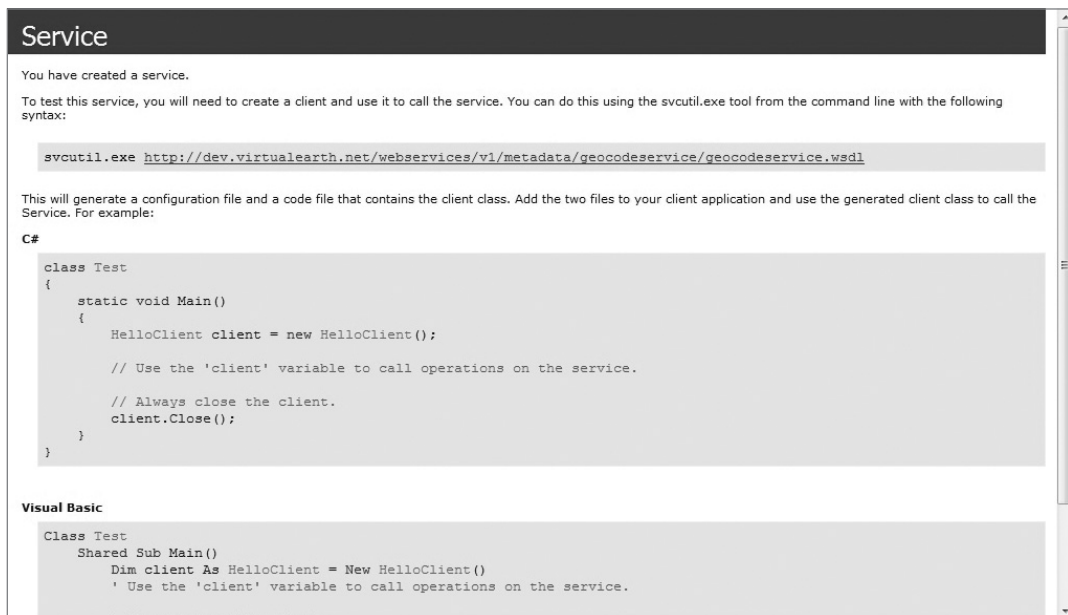


FIGURE 7-11

3. Right-click the project and select Add Service Reference. Add the service URI into the Address field and click Go, as shown in Figure 7-12.

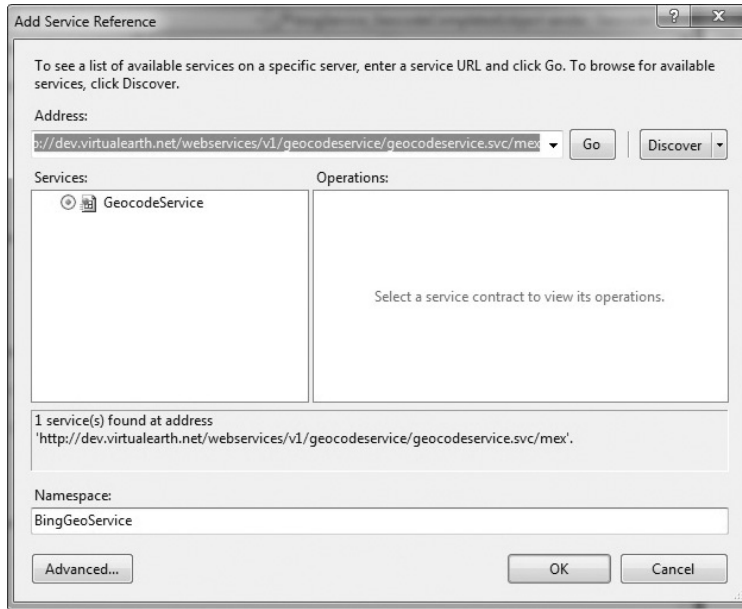



FIGURE 7-12

4. Provide a name for the service reference (e.g., BingGeoService) in the Namespace field. Click Advanced, and in the Collection Type drop-down list, select System.Array.
5. You now need to add the SharePoint client object model libraries. To do this, right-click the Silverlight project and select Add Reference. Click Browse and then browse to `c:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin\` and add `Microsoft.SharePoint.Client.Silverlight.dll` and `Microsoft.SharePoint.Client.Silverlight.Runtime.dll`.
6. Right-click the project and select Add New  Class. Provide a name for the class (e.g., StoreInfo), and click Add. Amend the class code as shown in bold in the following code snippet. This class is the in-memory representation of the store information and the latitude and longitude that will be returned from the service call.



Available for
download on
Wrox.com

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
```

```

using Microsoft.Maps.MapControl;

namespace BingMapSolution
{
    public class StoreInfo
    {
        public string StoreName { get; set; }
        public string StoreAddress { get; set; }
        public string StoreCity { get; set; }
        public string StoreState { get; set; }
        public string StoreZip { get; set; }
        public string StoreSales { get; set; }
        public string StoreLatitude { get; set; }
        public string StoreLongitude { get; set; }
    }
}

```

code snippet 076576 Ch07_Code.zip/StoreInfo.cs

7. Right-click the MainPage.xaml file and select View Code. Amend the code in the MainPage.xaml.cs file as per the following bolded code:



Available for
download on
Wrox.com

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Threading;
using Microsoft.SharePoint.Client;
using Microsoft.Maps.MapControl;
using BingMapSolution.BingGeoService;

namespace BingMapSolution
{
    public partial class MainPage : UserControl
    {
        private ClientContext context;
        private List<StoreInfo> storeItems;
        private List<StoreInfo> stores;
        private List<StoreInfo> listOfStores;

        List<StoreInfo> listOfStoreSummaries = new List<StoreInfo>();

        string strStoreName = null;
        string strStoreCity = null;
        string strStoreAddress = null;
        string strStoreState = null;
        string strStoreZip = null;
    }
}

```

```

string strStoreSales = null;
string strStoreLatitude = null;
string strStoreLongitude = null;

private List oList;
private string siteUrl = "http://blueyonderdemo";

public MainPage()
{
    InitializeComponent();
}

private void btnAddStores_Click(object sender, RoutedEventArgs e)
{
    Connect();
}

private void Connect()
{
    context = new ClientContext("http://blueyonderdemo");
    context.Load(context.Web);
    context.Load(context.Web.Lists);
    stores = context.Web.Lists.GetByTitle("Store Sales");
    context.Load(stores);
    context.Load(stores.RootFolder);
    var camlQuery = new CamlQuery();
    camlQuery.ViewXml = "<View/>";
    storeItems = stores.GetItems(camlQuery);
    context.Load(storeItems);
    context.ExecuteQueryAsync(OnLoadItemsSucceeded, onQueryFailed);
}

private void OnLoadItemsSucceeded(object sender,
ClientRequestSucceededEventArgs args)
{
    Dispatcher.BeginInvoke(ShowItems);
}

private void ShowItems()
{
    listOfStores = new List<StoreInfo>();

    foreach (ListItem listItem in storeItems)
    {
        listOfStores.Add(
            new StoreInfo()
            {
                StoreName = listItem["Title"].ToString(),
                StoreAddress = listItem["Address"].ToString(),
                StoreCity = listItem["City"].ToString(),
                StoreState = listItem["State"].ToString(),
                StoreZip = listItem["Zip"].ToString(),
                StoreSales = listItem["Sales"].ToString(),
                StoreLatitude = listItem["Latitude"].ToString(),
            }
        );
    }
}

```

```
        StoreLongitude = listItem["Longitude"].ToString()
    });
}

lstStores.ItemsSource = listOfStores;
}

private void btnZoom_Click(object sender, RoutedEventArgs e)
{
    Location locationFilter = new Location();
    StoreInfo tempStoreRecord = new StoreInfo();

    tempStoreRecord = (StoreInfo)lstStores.SelectedItem;
    string storeLatitude = tempStoreRecord.StoreLatitude;
    string storeLongitude = tempStoreRecord.StoreLongitude;

    locationFilter.Latitude = Double.Parse(storeLatitude);
    locationFilter.Longitude = Double.Parse(storeLongitude);

    MainMap.SetView(locationFilter, 10);
}

private void btnGetStores_Click(object sender, RoutedEventArgs e)
{
    strStoreName = txtbxStoreName.Text;
    strStoreCity = txtbxStoreCity.Text;
    strStoreAddress = txtbxStoreAddress.Text;
    strStoreState = txtbxStoreState.Text;
    strStoreZip = txtbxStoreZipCode.Text;
    strStoreSales = txtbxStoreSales.Text;
    strStoreLatitude = txtbxLatitude.Text;
    strStoreLongitude = txtbxLongitude.Text;

    ClientContext clientContext = new ClientContext(siteUrl);
    Web oWebsite = clientContext.Web;
    ListCollection collList = oWebsite.Lists;
    oList = clientContext.Web.Lists.GetByTitle("Store Sales");
    ListItem oListItem = oList.AddItem(new ListItemCreationInformation());
    oListItem["Title"] = strStoreName;
    oListItem["Address"] = strStoreCity;
    oListItem["City"] = strStoreAddress;
    oListItem["State"] = strStoreState;
    oListItem["Zip"] = strStoreZip;
    oListItem["Sales"] = strStoreSales;
    oListItem["Latitude"] = strStoreLatitude;
    oListItem["Longitude"] = strStoreLongitude;
    oListItem.Update();

    clientContext.Load(oList, list => list.Title);

    clientContext.ExecuteQueryAsync(onQuerySucceeded, onQueryFailed);
}

private void onQuerySucceeded(object sender, ClientRequestSucceededEventArgs
args)
```

```

    {
        Dispatcher.BeginInvoke(AddItem);
    }

    void AddItem()
    {
        Pushpin pushpin = new Pushpin();
        Location pushpinLocation = new Location();
        pushpinLocation.Latitude = Double.Parse(strStoreLatitude);
        pushpinLocation.Longitude = Double.Parse(strStoreLongitude);

        pushpin.Width = 7;
        pushpin.Height = 10;
        pushpin.Tag = strStoreName;
        pushpin.Location = pushpinLocation;
        StorePushpins.AddChild(pushpin, pushpinLocation, PositionOrigin.Center);

        MessageBox.Show("Store was successfully added.");
    }

    private void onQueryFailed(object sender, ClientRequestFailedEventArgs args)
    {
        Dispatcher.BeginInvoke(EventFailed);
    }

    void EventFailed()
    {
        MessageBox.Show("Request failed.");
    }

    private void btnClear_Click(object sender, RoutedEventArgs e)
    {
        txtbxStoreName.Text = "";
        txtbxStoreAddress.Text = "";
        txtbxStoreCity.Text = "";
        txtbxStoreState.Text = "";
        txtbxStoreZipCode.Text = "";
        txtbxStoreSales.Text = "";
    }

    private void chkbxLatLong_Checked(object sender, RoutedEventArgs e)
    {
        GeocodeRequest bingRequest = new GeocodeRequest();

        bingRequest.Credentials = new Credentials();
        bingRequest.Credentials.ApplicationId = "<your key here>";

        bingRequest.Query = txtbxStoreAddress.Text + "," + txtbxStoreCity.Text +
            "," + txtbxStoreState.Text + "," + txtbxStoreZipCode.Text;

        ConfidenceFilter[] filters = new ConfidenceFilter[1];
        filters[0] = new ConfidenceFilter();
        filters[0].MinimumConfidence = Confidence.High;

        GeocodeOptions bingOptions = new GeocodeOptions();

```

```

        bingOptions.Filters = filters;

        bingRequest.Options = bingOptions;

        if (bingRequest.Query != "")
        {
            GeocodeServiceClient geocodeService = new GeocodeServiceClient("Basic
HttpBinding_IGeocodeService");
            geocodeService.GeocodeCompleted += new EventHandler<GeocodeCompletedE
ventArgs>(bingService_GeocodeCompleted);
            geocodeService.GeocodeAsync(bingRequest);
        }
        else
        {
            MessageBox.Show("Please enter a valid Address.");
        }
    }

    void bingService_GeocodeCompleted(object sender, GeocodeCompletedEventArgs e)
    {
        GeocodeResponse bingResponse = e.Result;

        if (bingResponse.Results.Length > 0)
        {
            txtbxLatitude.Text = bingResponse.Results[0].Locations[0].Latitude.
ToString();
            txtbxLongitude.Text = bingResponse.Results[0].Locations[0].Longitude.
ToString();
        }
        else
        {
            MessageBox.Show("No Results found");
        }
    }
}

```

code snippet 076576 Ch07_Code.zip/MainPage.xaml.cs

At this point, you've finished adding all of the main code into your Bing maps application, so let's take a look at what it does. The code you added accomplishes four main tasks, each of which is tied to a user control:

- It loads the list data from the SharePoint list and displays it in the data-bound text blocks in the list box.
- It adds data to the SharePoint list.
- It enables users to zoom in on a particular store that has been added to the application.
- It enables data that has been entered into the text boxes to be cleared.

Let's look at each of these in sequence. To load the list data, the user must click the Get Stores button, which invokes the `btnAddStores_Click` method. The only method within

the `btnAddStores_Click` event is the `Connect` method, which sets the context for the SharePoint site and then calls the `ExecuteQueryAsync` method to asynchronously batch process the commands to SharePoint. As discussed earlier, because you are using Silverlight, you need to manage the asynchronous calls and threading, so you invoke the `ShowItems` method by using the `Dispatcher.BeginInvoke` method. Using an in-memory object and list collection (`listOfStores`), you then iterate over the list items and populate the list collection. Lastly, you bind the list collection to the list box (actually the text blocks in the list box), and the information from your SharePoint list is then displayed in the Silverlight application.

```
...
private void btnAddStores_Click(object sender, RoutedEventArgs e)
{
    Connect();
}

private void Connect()
{
    context = new ClientContext("http://blueyonderdemo");
    context.Load(context.Web);
    context.Load(context.Web.Lists);
    stores = context.Web.Lists.GetByTitle("Store Sales");
    context.Load(stores);
    context.Load(stores.RootFolder);
    var camlQuery = new CamlQuery();
    camlQuery.ViewXml = "<View/>";
    storeItems = stores.GetItems(camlQuery);
    context.Load(storeItems);
    context.ExecuteQueryAsync(OnLoadItemsSucceeded, onQueryFailed);
}

private void OnLoadItemsSucceeded(object sender,
ClientRequestSucceededEventArgs args)
{
    Dispatcher.BeginInvoke(ShowItems);
}

private void ShowItems()
{
    listOfStores = new List<StoreInfo>();

    foreach (ListItem listItem in storeItems)
    {
        listOfStores.Add(
            new StoreInfo()
            {
                StoreName = listItem["Title"].ToString(),
                StoreAddress = listItem["Address"].ToString(),
                StoreCity = listItem["City"].ToString(),
                StoreState = listItem["State"].ToString(),
                StoreZip = listItem["Zip"].ToString(),
                StoreSales = listItem["Sales"].ToString(),
                StoreLatitude = listItem["Latitude"].ToString(),
                StoreLongitude = listItem["Longitude"].ToString()
            }
        );
    }
}
```

```
        });  
    }  
  
    lstStores.ItemsSource = listOfStores;  
}  
...
```

When the SharePoint data is loaded into the application, you can zoom in and focus on one of the items displayed in the list box. This requires the use of the `Location` object, which is specific to the Bing Maps API. You can see in the following code snippet that you are creating a new instance of the `Location` object, retrieving the latitude and longitude from the selected list box item (with an explicit cast to the `StoreInfo` object), and then calling the `SetView` method against your Bing Maps map (`MainMap`), which repositions the map to the latitude and longitude set on the `locationFilter` object:

```
...  
private void btnZoom_Click(object sender, RoutedEventArgs e)  
{  
    Location locationFilter = new Location();  
    StoreInfo tempStoreRecord = new StoreInfo();  
  
    tempStoreRecord = (StoreInfo)lstStores.SelectedItem;  
    string storeLatitude = tempStoreRecord.StoreLatitude;  
    string storeLongitude = tempStoreRecord.StoreLongitude;  
  
    locationFilter.Latitude = Double.Parse(storeLatitude);  
    locationFilter.Longitude = Double.Parse(storeLongitude);  
  
    MainMap.SetView(locationFilter, 10);  
}  
...
```

Next, you display the data in the Bing Maps application — again using the SharePoint client object model. Here, you're setting the context with a class-level variable (`siteUrl`) and then creating a list item by calling the `AddItem` method and creating a new instance of `ListItemCreationInformation`. Based on what the user entered into the textboxes, you can then map the list item fields (e.g., `oListItem["Title"]`) to the strings extracted from the textboxes (e.g., `strStoreName`). The `Update` method, via the `ExecuteAsyncQuery` method, then adds a record to the list. Note that when the item is successfully added to the list, a `Pushpin` object is created, which is a native object in Bing Maps, and then added to the map.

```
...  
private void btnGetStores_Click(object sender, RoutedEventArgs e)  
{  
    strStoreName = txtbxStoreName.Text;  
    strStoreCity = txtbxStoreCity.Text;  
    strStoreAddress = txtbxStoreAddress.Text;  
    strStoreState = txtbxStoreState.Text;  
    strStoreZip = txtbxStoreZipCode.Text;  
    strStoreSales = txtbxStoreSales.Text;  
    strStoreLatitude = txtbxLatitude.Text;  
    strStoreLongitude = txtbxLongitude.Text;  
  
    ClientContext clientContext = new ClientContext(siteUrl);  
    Web oWebsite = clientContext.Web;
```

```

        ListCollection collList = oWebsite.Lists;
        oList = clientContext.Web.Lists.GetByTitle("Store Sales");
        ListItem oListItem = oList.AddItem(new ListItemCreationInformation());
        oListItem["Title"] = strStoreName;
        oListItem["Address"] = strStoreCity;
        oListItem["City"] = strStoreAddress;
        oListItem["State"] = strStoreState;
        oListItem["Zip"] = strStoreZip;
        oListItem["Sales"] = strStoreSales;
        oListItem["Latitude"] = strStoreLatitude;
        oListItem["Longitude"] = strStoreLongitude;
        oListItem.Update();

        clientContext.Load(oList, list => list.Title);

        clientContext.ExecuteQueryAsync(onQuerySucceeded, onQueryFailed);
    }

    private void onQuerySucceeded(object sender, ClientRequestSucceededEventArgs
args)
    {
        Dispatcher.BeginInvoke(AddItem);
    }

    void AddItem()
    {
        Pushpin pushpin = new Pushpin();
        Location pushpinLocation = new Location();
        pushpinLocation.Latitude = Double.Parse(strStoreLatitude);
        pushpinLocation.Longitude = Double.Parse(strStoreLongitude);

        pushpin.Width = 7;
        pushpin.Height = 10;
        pushpin.Tag = strStoreName;
        pushpin.Location = pushpinLocation;
        StorePushpins.AddChild(pushpin, pushpinLocation, PositionOrigin.Center);

        MessageBox.Show("Store was successfully added.");
    }
    ...

```

The final event is the clearing of the text boxes, which simply sets each of the text boxes to null.



When you add an address in the textboxes, you'll want to make sure the address is valid. There is no address-to-zip-code check in this code before calling the Bing Maps Web service, so if the zip code does not map to the street and city address, the Pushpin object will not be created because an exception will occur.

8. Press F6 to build the application. If it successfully builds, you'll be able to deploy the application to SharePoint.

Before you deploy the Silverlight application to SharePoint, you must have a SharePoint list with the appropriate fields in it. Referring back to the client object model code, you need eight columns in your list (name, address, city, state, and zip code for store, sales, and latitude and longitude).

Creating a Store and Sales List in SharePoint

Creating the list for the store and sales information is straightforward:

- 1. Navigate to your SharePoint site and select Site Actions ⇨ View All Site Content.
- 2. Click the Create button. In the Filter By pane, select List ⇨ Custom list.
- 3. Provide a name for your list (e.g., `Store Sales`) and click Create.
- 4. In your new list, click the List tab and then select List Settings. Click the Title field and change the title to `Name`. Then click the Create Column link and add another column called Address of type `Single line of text`. Do the same for all the other required columns (which should be named `City`, `State`, `Zip`, `Sales`, `Latitude`, and `Longitude`). When you've created all the new columns, your new list should resemble Figure 7-13.

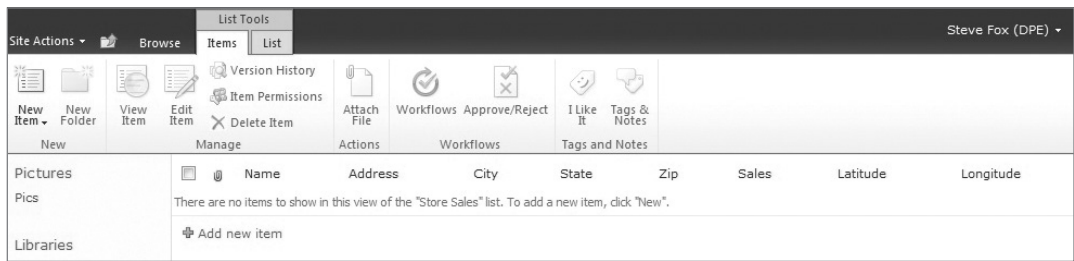


FIGURE 7-13

Now that you've added the SharePoint list to your site, you can deploy the Silverlight application you built to SharePoint.



If you want, you can programmatically create a new list. This can be done in a couple of ways. First, you could check your Silverlight application to see if a list exists; and if it doesn't, create it using the SharePoint client object model. Another way is to deploy the Silverlight application along with a SharePoint project that contains the list definition and instance. This also gives you the opportunity to wrap the Silverlight application in a web part project so that you can deploy the entire solution as a WSP.

Deploying the Silverlight Bing Application

The final process to get your application working is to deploy the Silverlight application to SharePoint. To deploy the Silverlight application to SharePoint, you will use the native Silverlight web part to host the deployed Silverlight application.

1. To create a new document library, navigate to your SharePoint site by clicking Site Actions ⇨ View All Site Content.
2. Click Create. Then, in the Filter-by pane, select Library.
3. Click Document Library, provide a name for the library (e.g., XAPS) and then click Create.
4. Click Add Document in the newly created XAPS document library, and then navigate to where your built .xap file is located (typically in the ~/bin/debug folder of your project). Your XAP file should now be in the newly created document library, as shown in Figure 7-14.

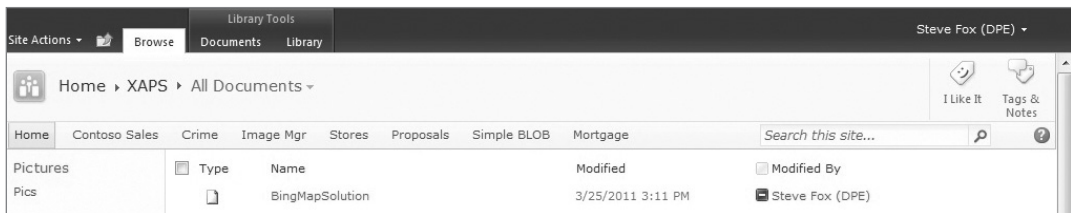


FIGURE 7-14

5. Right-click the shortcut for the added .xap file and select Copy Shortcut.
6. Click Site Actions ⇨ Edit Page.
7. Click Add a New Web Part or select the Insert tab and then select Web Part.
8. Click the Media and Content category and select Silverlight Web Part.
9. Click Add and then paste the shortcut to the .xap file from your XAPS document library into the URL field. After you add the Silverlight application, you may need to adjust the height and width of the web part.

When complete, you now have a Silverlight application in SharePoint that can not only read data from the Store Sales list, but also write data back to that list. Note that you are also leveraging the Bing Maps map to display the newly added records (by doing a dynamic geo-code look-up on the address). Figure 7-15 shows what this experience should look like.

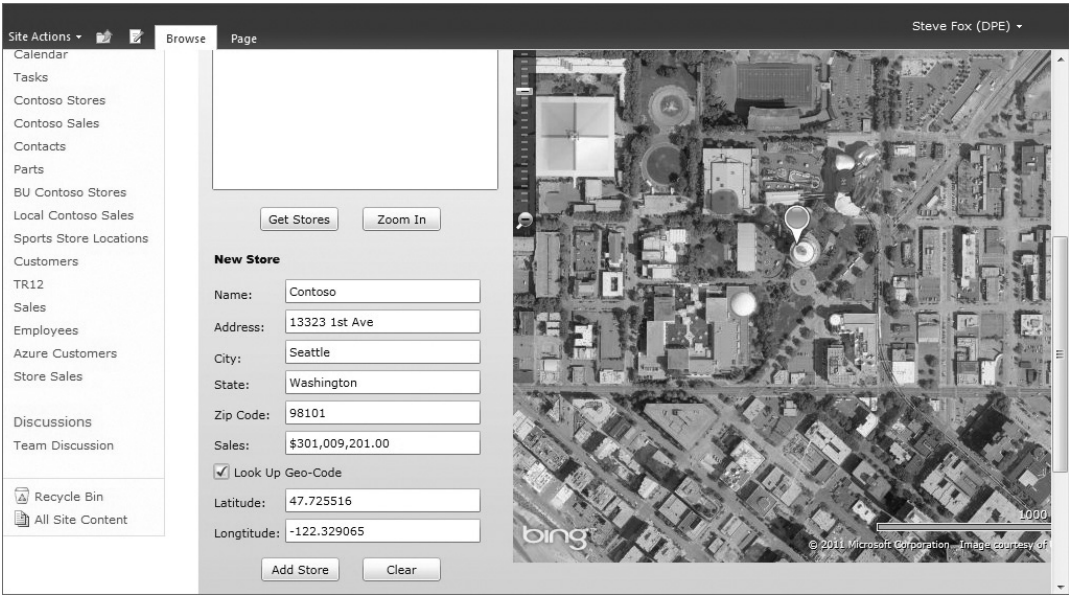


FIGURE 7-15

Go ahead and add a record to the application using the textbox controls. Then click the Look Up Geo-Code checkbox, and the application will concatenate the address information and look up the latitude and longitude for you. When you click the Add Store button, your information is then added to SharePoint. The result of this is shown in Figure 7-16.

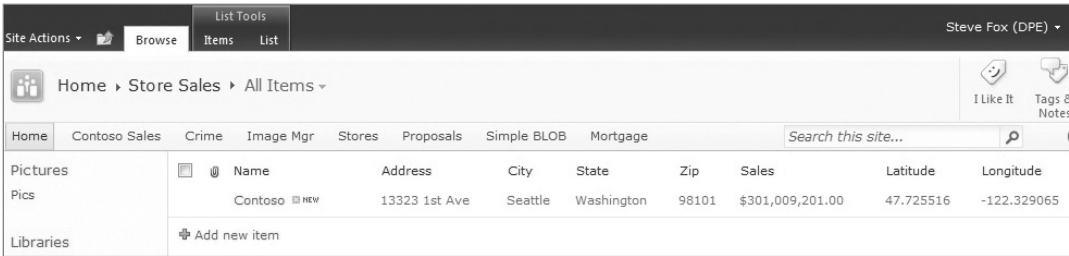


FIGURE 7-16

Note that when you return to the Silverlight application and click the Get Stores button, all entries that have been added to the SharePoint list are displayed in the list box. Figure 7-17 shows another entry added to the list box that is displayed in the Silverlight application.

The last event is triggered by the Zoom In button, so select one of the stores and click it to see the focus of the map change to the selected store.

You can build many more applications using Bing Maps, ranging from the easy to the complex. Bing Maps has a rich set of APIs that cater to many different scenarios, so if you require JavaScript instead of Silverlight, you can use the AJAX control; or for lightweight calls, use the REST APIs.

Either way, there is a lot there for you to explore. This chapter has only scraped the surface of what you can do.

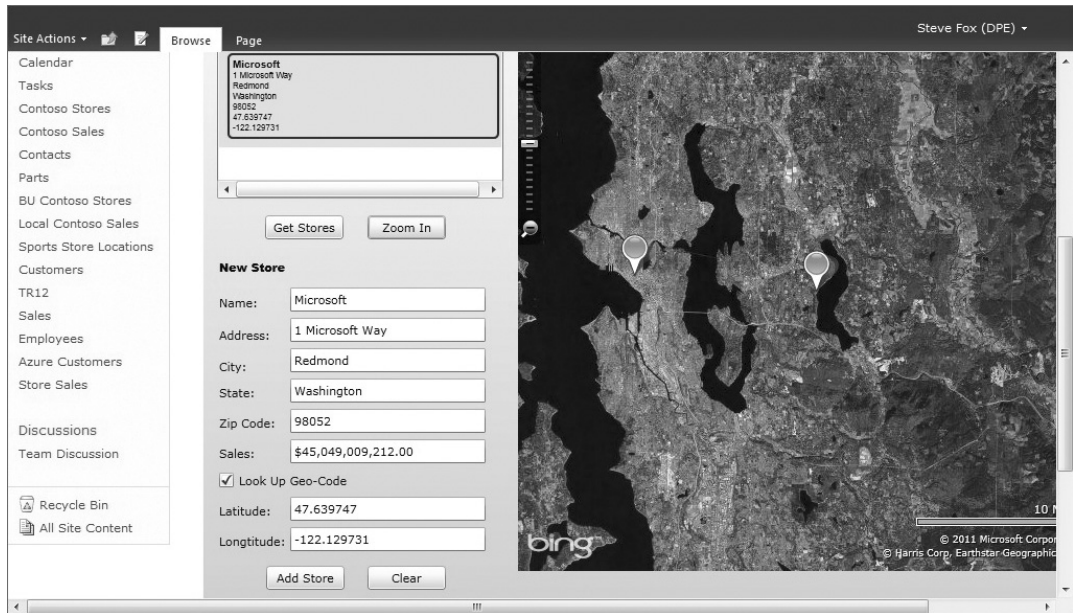


FIGURE 7-17

SUMMARY

You will see many more applications building in the ability to use geo data in some capacity. We're seeing more of these applications come online every day, and the data is just too rich to ignore within web or cloud-based applications. Further, geo data will enable you to build compelling applications that cut across devices and platforms, such as surfacing customer sales and location data on a Windows Phone 7.

That said, the goal of this chapter was to provide you with an introduction to Bing Maps geo services. There are other services out there, such as Google Maps, but Bing Maps provide you with an easy way to integrate directly with SharePoint through the use of the SharePoint client object model. You also have flexibility, with the option of using the Bing Maps Silverlight control or the Bing Maps AJAX control, both of which are compatible with the SharePoint client object model and natively supported in SharePoint.

In this chapter, you learned about Bing Maps; specifically, you learned what they are, the different APIs that you can use to leverage the public services, and how to build an application for SharePoint using the Bing Maps Silverlight control. Using the Silverlight control, you created an application that both reads and writes data to SharePoint and then displays that data in a Bing map and creates a pushpin for a store location. You also did a dynamic look-up to retrieve the

latitude and longitude of the store to ensure you have the exact location of the store that Bing understands.

As you explore the functionality and services offered through Bing Maps (and Bing more generally), you'll see an interesting path to enrich your applications with geo data.

In the next chapter, we'll take a turn away from Bing Maps geo services and talk more about building cloud-based financial models using Excel Services.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- **Web part extensibility project** — <http://visualstudiogallery.msdn.microsoft.com/e8360a85-58ca-42d1-8de0-e48a1ab071c7>
- **Silverlight and SharePoint training kit** — www.microsoft.com/download/en/details.aspx?amp;displaylang=en&id=10808
- **Bing maps developer center** — www.microsoft.com/maps/developers/web.aspx
- **Bing Maps Silverlight control download location** — www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=beb29d27-6f0c-494f-b028-1e0e3187e830
- **Silverlight web part for SharePoint** — <http://blogs.msdn.com/b/pstubbs/archive/2011/04/04/add-silverlight-web-parts-to-sharepoint-using-new-visual-studio-extension.aspx>

8

Financial Modeling with Excel Services and Windows Azure

WHAT'S IN THIS CHAPTER?

- Understanding opportunities for integrating Windows Azure with Office and Office Services in SharePoint 2010
- Understanding the concepts behind the Windows Azure Table service
- Extending an Excel financial model to access Windows Azure table data using WCF Data Services
- Building an ECMAScript Content Editor web part that calls an Azure-hosted WCF service to retrieve Windows Azure table data for use in the financial model under Excel Services

Using Microsoft Excel on the desktop for financial models is a broad practice across all vertical industries and helps enable both daily decision making and analysis for long-range planning. There are two excellent options to enhance financial models to give them greater reach.

First, in many cases, financial models are bound to the desktop — that is, although they can be shared via e-mail or saved to a file share, the models are used only on the desktop in the Excel client application. However, financial models uploaded to SharePoint are unbound from the desktop, enabling users who don't have Excel to experience the power of the model via Excel Services in their browser.

Second, the financial models often contain data connections that generally access on-premises data. With the advent of SQL Azure, the Windows Azure Blob service, and the Windows Azure Table service, cloud-based data sources are also available to these models.

Whether the financial models are used in the Excel client or under Excel Services, Azure cloud-based data can be accessed to enrich the models. This chapter describes the patterns for accessing Azure table data from both the client and SharePoint.

OVERVIEW OF EXCEL FINANCIAL MODELS

Since its release to business customers in May 2010, Microsoft Office 2010 has sold at a record pace. One might have anticipated this would be the case, because during one seven-month stretch of the product's public beta, one million downloads were being installed each month. This demand, coupled with the fact that more than 90 million businesses worldwide have chosen Office for their end user productivity suite, means there is a significant surface area for developers to land business solutions built on Office.

One such business solution is an Excel financial model. Microsoft Excel has been used extensively across the private and public sector for creating financial models. These models range from fairly simple mortgage and debt reduction calculators to complex amortization schedules and Monte Carlo simulations to help with risk and decision analysis when there is uncertainty in the mix. These and other models created in Excel across the spectrum of engineers, statisticians, financial analysts, scientists, and others have proven invaluable in supporting day-to-day business decisions and long-term planning.

Figure 8-1 shows a simple financial model for calculating and comparing a number of monthly mortgage payment amounts. As with all financial models, this model requires a set of inputs to generate the outputs. In this model, a real estate agent provides the mortgage amount, the number of years for the loan, and the fixed interest rate. The financial model then calculates additional interest rates with corresponding monthly payments for comparison and a chart. Although the real estate agent provides only the single fixed interest rate, the model uses that to calculate two lower interest rates and a monthly payment at a half percent lower for each. It also calculates four higher interest rates and monthly payments at a half percent higher for each. This simple example shows the power of a financial model to take key input values and calculate outcomes that can support analysis and decision-making.

In the mortgage payment financial model example, the end user — in this case, the real estate agent — needs to know and provide the fixed interest rate values, but financial models can also rely on data retrieved from disparate sources. How much better would this financial model be if the real estate agent could make interest rate selections based on authoritative data? Rather than needing to know or find the interest rate being offered that day for the number of years desired for the loan, what if the agent could select that value, or values, from a data source? What if the selection could be made based on the rate being offered by any given lending institution for that day? Excel provides a number of built-in ways to access external data — e.g., SQL Server, Analysis Server, OLE DB, and Microsoft Query and ODBC. These data-access technologies have been relied upon for years both to simplify the end-user experience and to enrich the outputs of financial models.

Now, cloud-based options are available for hosting your authoritative data, such as SQL Azure, Azure blobs, and Azure tables. The great news is that these sources for authoritative data also can be tapped for use in your financial models. This chapter focuses on accessing data in Azure tables.

You will see two patterns for integrating Azure table data into the mortgage financial model: when it's used in the Excel client and when it's used in SharePoint under Excel Services.

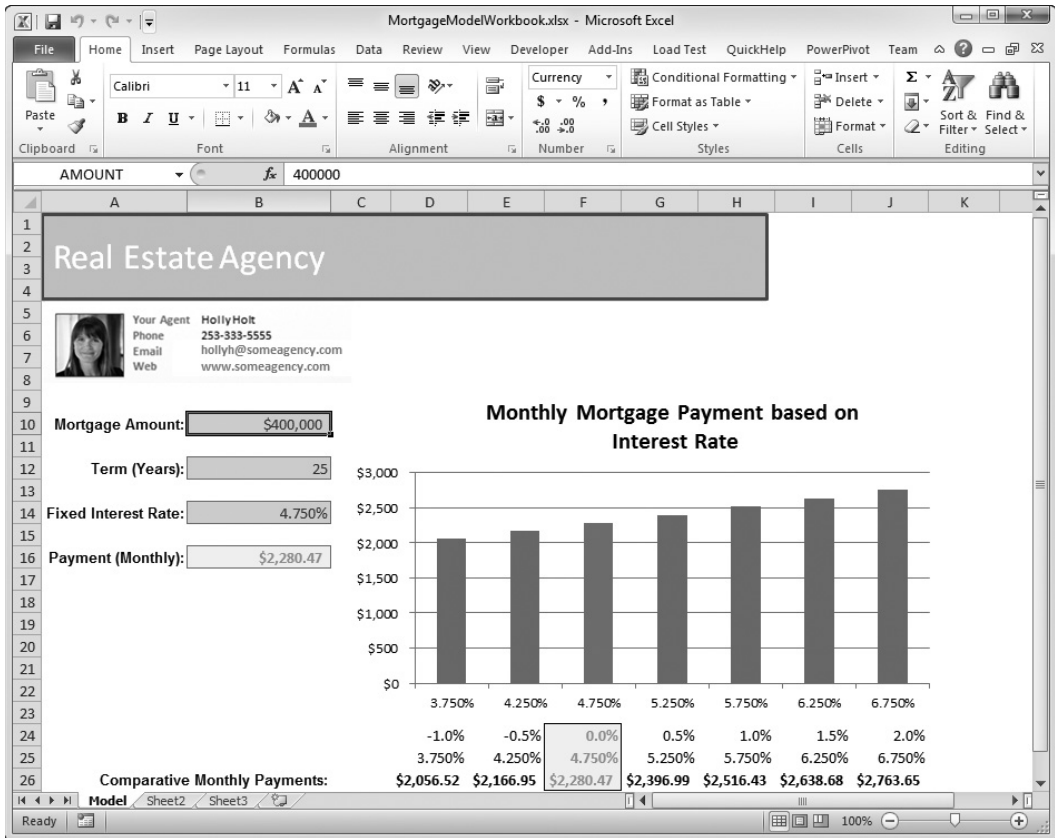


FIGURE 8-1

INTEGRATING WINDOWS AZURE WITH OFFICE

Windows Azure and the pervasive use of Office offer a significant opportunity for Office solution developers and end users alike. Solution developers can have confidence that their data endpoints are always available at scale and that their Office solution integrates directly within a context the end user has already mastered. For the end user, Office solutions bring Azure-based data and information to them within the applications they are already using.

However, it's not only Office client application use that end users are familiar with; tens of thousands of users also interact daily with SharePoint and Excel Services. Consider the mortgage financial model. If you provide an Azure-enabled version of this in the Excel client, end users will also expect the same functionality when they use the model under Excel Services. Therefore, it is important to architect your solutions to take advantage of your Azure-based data from both contexts.

From the client, you can use .NET to build your solution; from SharePoint, you can use a variety of web-related technologies to access your Azure data. Let's take a closer look.

Office Clients and Azure

In the Office clients, you can exploit the power of the .NET Framework to build your Office and Azure solutions. If you have not previously developed Office solutions but have developed rich client applications on .NET, consider landing your next solution within Office. The benefit is that you'll be able to leverage all your current skills and then learn some new ones related to the different Office client APIs. Figure 8-2 shows the various Office 2010 project templates available in Visual Studio 2010.

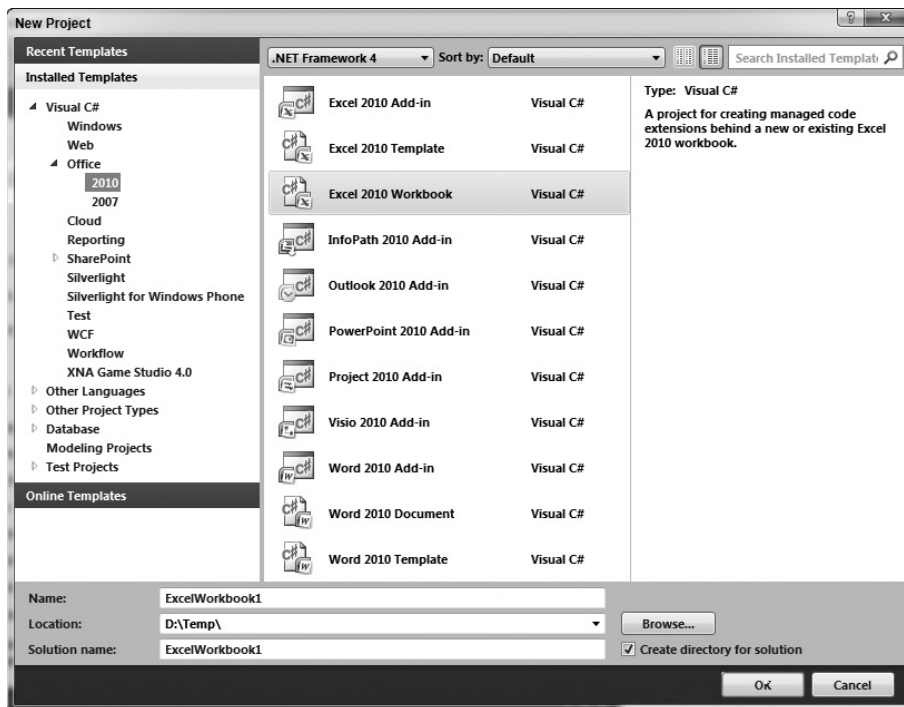


FIGURE 8-2

Each project template is an opportunity to land Azure-based data in an Office client. Notice that some of the project names end with Add-in, whereas others end with Workbook, Document, or Template. An Office client add-in (considered an application-level add-in) will load each time the client starts. Therefore, once installed, the add-in is always available when the Office client is running. Workbook, document, and template projects (considered document-level add-ins) are add-ins, but they are associated with a specific workbook, document, or template, respectively. A document-level add-in will load only when that specific workbook, document, or template is opened.

When you are building your Office add-in, application-level add-ins surface their UI in a task pane, whereas document-level add-ins surface theirs in an action pane. Although this is a small nuance, in either case you design the end user experience for the task or action pane as you would for a rich client application. You can use either Windows Forms controls or Windows Presentation Foundation (WPF). Some developers host a browser control in the pane and surface the UI as a Silverlight

application. You simply use what you are already familiar with to design the end user task or action pane experience. Additionally, document-level solutions let you place controls directly onto the document's surface for the end user to use. These, too, can be sourced with external data, so there's a tremendous range of solutions you can build.

For data access in Office client add-ins, you can use WCF Data Services to connect directly to your Azure data, or you can build a WCF service to front your Azure data. Again, in either case, you are using the tools you already know when developing for Office. You'll be working through a document-level solution in this chapter, so you will get some experience in developing an Office add-in. For additional information about Office development, visit msdn.microsoft.com/office.

Office Services and Azure

Essentially, Office Services made their debut in MOSS 2007 with Excel Services and InfoPath Forms Services. However, with the advent of SharePoint 2010, the presence of Office Services increased to include Word Automation Services, Access Services, and Visio Services as well. This further incorporation of Office capabilities, coupled with enriched capabilities in both Excel Services and InfoPath Forms Services, has enabled powerful end-to-end Office solutions that incorporate both the Office clients and Office Services.

However, the significant difference between the Office clients and their respective Office Services counterparts is that for the most part, Office Services are not extensible. For instance, you cannot develop an add-in for the Excel, Word, or Visio client and have that add-in function under Excel, Word, or Visio Services, respectively. Add-ins built for the client do not function on the server.

Of course, this has implications when, for example, you upload your fully functional, Azure data-dependent financial model to SharePoint. Although end users can interact with the document itself under Excel Services, the client add-in methods to retrieve the Azure data are no longer functional. How, then, do you compensate for this reduction in functionality? After all, end users expect client functionality to be available under Excel Services, too.

To resolve this in SharePoint, you need to use web-based technologies, as the documents are being accessed in the browser, not in the rich client. For Excel Services and Visio Services, you can use the ECMAScript (think JavaScript) object model. Although it is not a deep object model for manipulating the documents, the ECMAScript object model opens up the world of JavaScript's capabilities for accessing Azure-based storage services and enables you to manipulate Excel and Visio documents in the browser.

In this chapter you'll use the Excel ECMAScript object model in a SharePoint Content Editor web part to retrieve your Azure table data via an Azure-hosted WCF service. The user will interact with the Content Editor web part to select the desired input data for a mortgage financial model hosted in an Excel Web Access web part on the same page. Here again you'll see the capability to reach Azure data for Office but from SharePoint using web-based technologies.

THE SOLUTION ARCHITECTURE

The architecture for this solution is relatively straightforward. You'll use a single Azure table and access it in two different ways. First, from the Excel client, you'll access the Azure table using WCF Data Services. The document-level add-in for Excel has all the data access plumbing built in, so the

Azure table data is being retrieved by the add-in itself. As shown in Figure 8-3, the Excel client add-in accesses the Windows Azure Table service via WCF Data Services.

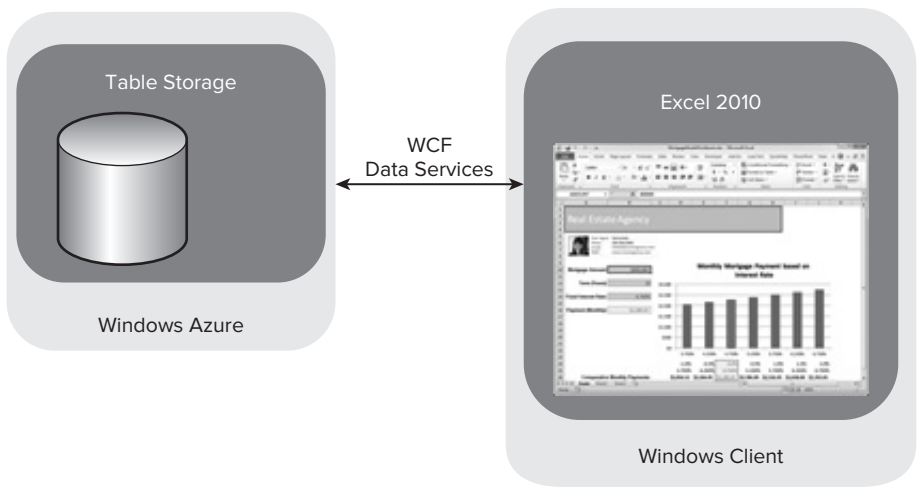


FIGURE 8-3

Second, to make the Azure table data available from ECMAScript on SharePoint, an Azure-hosted WCF service is used. The WCF service itself simply uses the same WCF Data Services constructs that are used in the Excel client add-in. However, for access via ECMAScript, a service endpoint is provided for the requested data; the ECMAScript does not access the Azure table directly. This same WCF service could have been used by the Excel client, but this example demonstrates two different patterns for accessing Azure table data: one direct and one via a service.

Figure 8-4 shows that the Azure-based WCF service is called from the Content Editor web part.

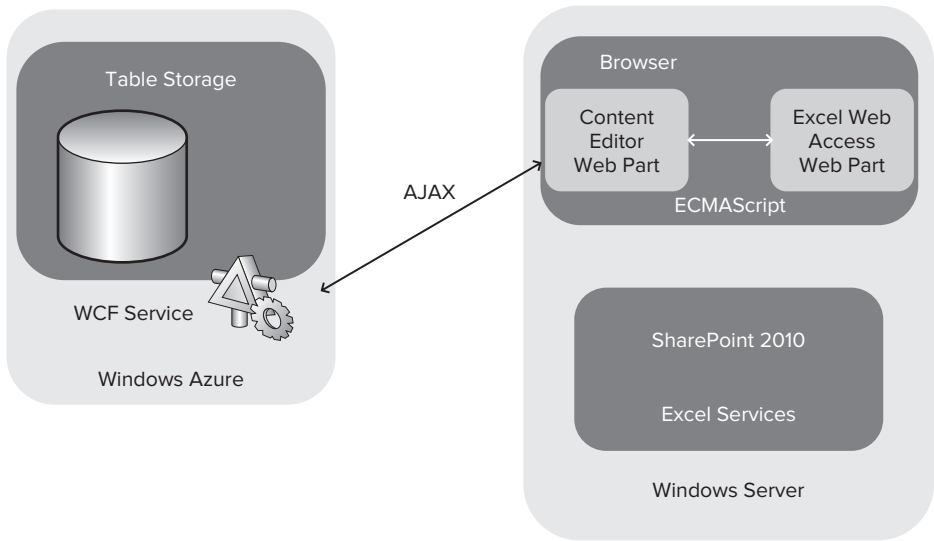


FIGURE 8-4

Accessing Azure Table Data via WCF Data Services

Before you can access Azure table data, you must first set up your storage account. If you haven't done so yet, navigate to your Windows Azure management portal, click Storage Accounts in the left navigation bar, and then click New Storage Account in the ribbon. As shown in Figure 8-5, the storage account name you choose will be used across your Blob, Table, and Queue data stores. When you choose OK, the Windows Azure management portal page will show the URLs for each store and provide a View button to see your <Hidden> access keys. You will need to use one of these keys, generally the primary access key, in your application that is connecting to any of these data stores.

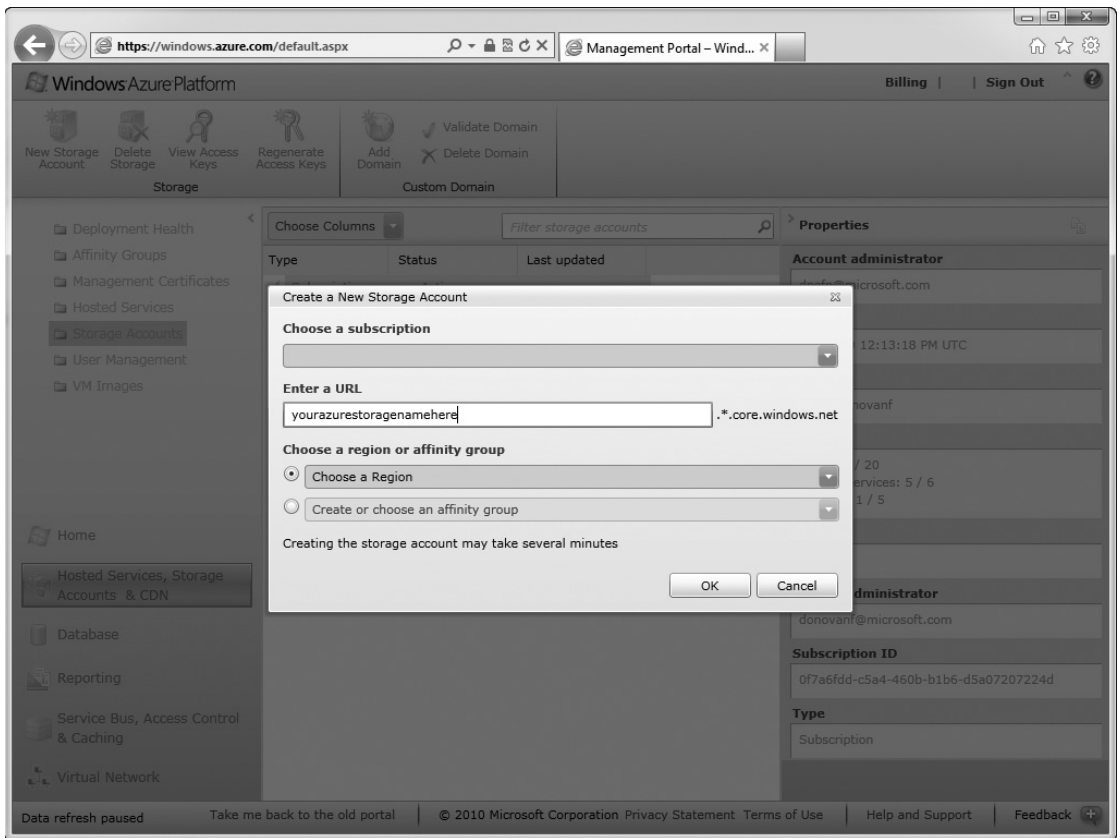


FIGURE 8-5

Now that you have your Windows Azure Table account, you can add data to it — but first some concepts. The first concept to wrap your mind around is that Azure table storage is not a relational data store — that's SQL Azure. One way to think of Azure table storage is that it is a highly scalable store for data that can be characterized by rows and columns. Each row can contain a maximum of 255 columns, and data in the entire row cannot exceed 1 MB in size. Each column can be defined by any one of the data types: Binary, Bool, DateTime, Double, GUID, Int, Int64, and String. Although visualizing table data laid out as rows and columns provides a nice mental model, these are not the terms used to describe the table data model.

Figure 8-6 shows the data model for Windows Azure tables. At the highest level is your table storage account. Your account can contain a set of tables, which can contain a set of entities (rows), which can contain a set of properties (columns).

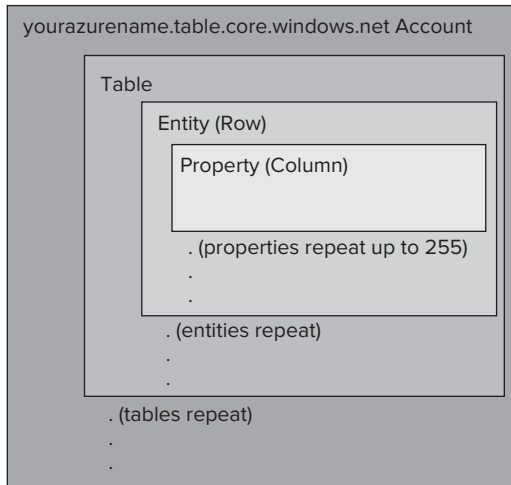


FIGURE 8-6

When you create a table, you provide a table name that must be unique across all your tables. Each entity (row) has two required properties (columns): a `PartitionKey` and a `RowKey`, which together are the unique key for the entity. You must provide a unique value for the `PartitionKey`, but providing the value for the `RowKey` is optional. The `RowKey` must be unique within its `PartitionKey`, but if you do not provide the unique value, Azure will generate the `RowKey` value automatically.



The `PartitionKey` key values you use are significant in terms of how Windows Azure will optimize and manage your table's partitions across its storage nodes. A discussion of this is beyond the scope of this chapter, so refer to the Windows Azure documentation, at www.microsoft.com/windowsazure/Whitepapers/WindowsAzureTable/, to learn how to select partition names that enable your tables to perform at scale.

With your account in place, it's time to get to work. In the following steps, you'll build a quick console application to create a table, and load a couple of entities into it. You'll then view your table in the Visual Studio 2010 Server Explorer. The work you do here will serve as the basis for what you'll do later in the chapter.

1. Open Visual Studio 2010 as Administrator.
2. Select File ⇨ New Project.
3. In the New Project dialog, click Windows ⇨ Console Application, provide a name for the project (e.g., `LendingRateTable`) and a location for the project, and click then OK (see Figure 8-7).

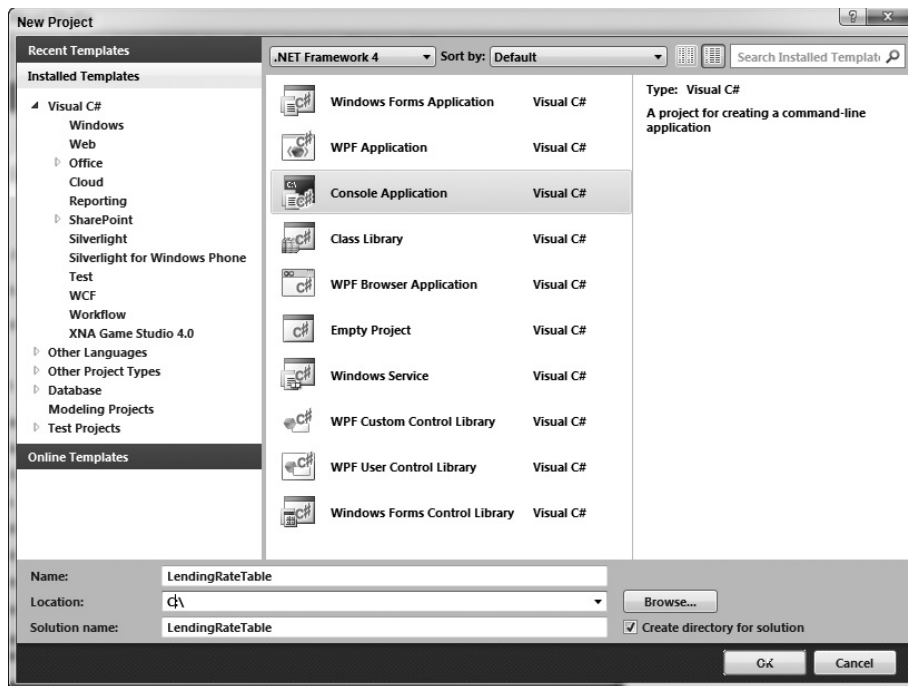


FIGURE 8-7

4. After the project is created, right-click on the LendingRateTable project in the Solution Explorer and choose Add References. From the .NET tab, select System.Data.Services.Client and Microsoft.WindowsAzure.StorageClient, and then click OK. (If you are prompted to retarget to the .NET Framework 4, do so by confirming the prompts.)
5. Right-click the LendingRateTable project and choose Add ➤ New Item. Select Class and name it LendingInstitutionRate.cs.
6. Replace all the code in the class with the following code:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure.StorageClient;

namespace LendingRateTable
{
    public class LendingInstitutionRate : TableServiceEntity
    {
        public string LendingInstitution { get; set; }
        public string ID { get; set; }
        public string Instrument { get; set; }
        public int Year { get; set; }
        public double Rate { get; set; }
        public double Points { get; set; }
    }
}
```

```

public double APR { get; set; }
public string Lock { get; set; }
public string InstrumentRateDisplayName { get; set; }

public LendingInstitutionRate(string ID)
{
    PartitionKey = "LendingRates";
    RowKey = ID;
}

public LendingInstitutionRate()
{
    PartitionKey = "LendingRates";
}
}
}

```

code snippet 076576 Ch08_Code.zip/LendingInstitutionRate.cs

7. Press F6 to confirm that your project builds. If there are errors, it may be because you need to retarget the project to the .NET Framework 4. Save all the files, right-click on the project in the Solution Explorer, and select Properties. Confirm that the Target Framework is the .NET Framework 4, not the .NET Framework 4 Client Profile. If necessary, change it to .NET Framework 4 and respond Yes to reload the solution.

The `LendingRateTable` class models the schema for this entity (again, think row) in the Azure table; and each property (think column) in the class is defined with one of the supported data types for Azure tables. By adding the `Microsoft.WindowsAzure.StorageClient` namespace, you can derive your class from `TableServiceEntity`, which already defines the mandatory table properties for an entity, such as `PartitionKey` and `RowKey`. Notice that there are two constructors for the class. For simplicity, each constructor has a hard-coded value for the `PartitionKey`, but the first constructor also accepts a string for the unique `RowKey` value. The second constructor has no `RowKey` provided, so Azure will create a unique `RowKey` value when this constructor is used to create a new entity.

8. Add another new class named `LendingDataContext.cs` to the project.
9. Replace all the code in the class with the following code:



Available for
download on
Wrox.com

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure;

namespace LendingRateTable
{
    public class LendingDataContext : TableServiceContext
    {
        public LendingDataContext(string baseAddress, StorageCredentials credentials)
            : base(baseAddress, credentials)
        {

```

```

    }

    public IQueryable<LendingInstitutionRate> LendingInstitutionRate
    {
        get
        {
            return this.CreateQuery<LendingInstitutionRate>("LendingInstitutionRate");
        }
    }
}

```

code snippet 076576 Ch08_Code.zip/LendingDataContext.cs

10. Press F6 to confirm that your project builds.

The `TableServiceContext` is an Azure-specific `DataServiceContext`, which is one of the main classes in WCF Data Services. In this class, `LendingDataContext` derives from `TableServiceContext` and will provide the runtime context for this data service. Although the nature of a data service is stateless, the context within which an application interacts with the data service is not. Therefore, at runtime, `LendingRateContext` will represent a connection in WCF Data Services to your storage account and will provide the APIs for querying, inserting, updating, and deleting entities.

Lastly, you will add a data source class to your project.

11. Add another new class named `LendingDataSource.cs` to the project.
12. Replace all the code in the class with the following code:



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

namespace LendingRateTable
{
    public class LendingDataSource
    {
        private static CloudStorageAccount storageAccount;
        private LendingDataContext context;

        static LendingDataSource()
        {
            try
            {
                storageAccount = CloudStorageAccount.DevelopmentStorageAccount;

                CloudTableClient c = new CloudTableClient(
                    storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);

                if (!c.DoesTableExist("LendingInstitutionRate"))
                {

```

```
        CloudTableClient.CreateTablesFromModel(typeof(LendingDataContext),
            storageAccount.TableEndpoint.AbsoluteUri,
            storageAccount.Credentials);
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}

public LendingDataSource()
{
    this.context = new LendingDataContext(
        storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);

    this.context.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1));
}

// Retrieve all lending institution rates entities in the table
public IEnumerable<LendingInstitutionRate> Select()
{
    var results = from g in this.context.LendingInstitutionRate
        where g.PartitionKey == "LendingRates"
        select g;
    return results;
}

// Add a single lending institution rate entity
public void AddLendingInstitutionRate(LendingInstitutionRate newItem)
{
    this.context.AddObject("LendingInstitutionRate", newItem);
    this.context.SaveChanges();
}

// Use to add multiple lending institutions in a batch - one at a time.
// Then, call Save() after adding the institutions
public void AddLendingInstitutionRates(LendingInstitutionRate newItem)
{
    this.context.AddObject("LendingInstitutionRate", newItem);
}

// Save the entities to Azure
public void Save()
{
    this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
}

// Delete a specific entity
public void DeleteLendingInstitutionRate(LendingInstitutionRate item)
{
    this.context.DeleteObject(item);
    this.context.SaveChanges();
}
```

```

    }

    // Delete ALL entities in the table at one time
    public void DeleteLendingInstitutionRates()
    {
        foreach (var p in this.Select())
        {
            this.context.DeleteObject(p);
        }
        this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
    }
}

```

code snippet 076576 Cb08_Code.zip/LendingDataSource.cs

13. Press F6 to confirm that your project builds.

Although this is a chunky code snippet, it's doing some very straightforward things. The `LendingDataSource` class is the glue that brings everything together. Find the static `LendingDataSource()` constructor. This constructor is executed first and essentially determines whether the table already exists in Azure. If not, it creates the table using the static `CreateTablesFromModel` method in the `CloudTableClient` class. Then the public `LendingDataSource()` constructor is executed. Its role is to wire up the data context by passing in the necessary parameters to connect to your Azure table.

Next, as you browse down the code, you'll see the methods that enable querying, adding, saving, and deleting entities in the table. In the `Select` method, you can see how easy it is to use a LINQ query against the table. The context is connected to the table, so this query retrieves all the entities with the `PartitionKey` equal to `LendingRates`. In this case, you have only one partition in the table, but you can see how easy it is to access table data.

To add an entity, look at the `AddLendingInstitutionRate` method. Here, you pass in a `LendingInstitutionRate` object representing your entity, and it is passed to the `AddObject` method on the context. The data context tracks all pending changes but does not apply them until its `SaveChanges` method is called. In the `AddLendingInstitutionRate` method, the entity is added and then `SaveChanges` is called immediately. One entity is added and then saved.

The data context also supports the notion of batches. In some cases, you may want to do a series of adds or deletes and then only commit those when these are completed. This is how the methods `AddLendingInstitutionRates` and `Save` are used. The `AddLendingInstitutionRates` method can be called multiple times to add entities; but it's not until the `Save` method is called that the entities are actually stored in the Azure table. Notice that the context `SaveChanges` method passes in the enum `System.Data.Services.Client.SaveChangesOptions.Batch`. This notifies the context to save all pending changes in a single batch. Another example of this can be observed in the `DeleteLendingInstitutionRate` and `DeleteLendingInstitutionRates` methods.

Now that you have all the classes in place to access your Azure table, you're ready to put them to use.

14. In Visual Studio, click the Program.cs tab.
15. Place the following code in the Main method:



Available for
download on
Wrox.com

```

Console.WriteLine("Creating two test entities...");

LendingDataSource ds = new LendingDataSource();

// So this can be run more than one time
// (i.e. RowKeys must be unique)
ds.DeleteLendingInstitutionRates();

// Create an entity and add it
LendingInstitutionRate lir = new LendingInstitutionRate();
lir.PartitionKey = "LendingRates";
lir.RowKey = "1";
lir.LendingInstitution = "Woodgrove Bank";
lir.ID = "1";
lir.Instrument = "30 Year Fixed";
lir.Year = 30;
lir.Rate = 0.05;
lir.Points = 0.0;
lir.APR = 0.051;
lir.Lock = "30 day";
lir.InstrumentRateDisplayName = "Woodgrove Bank 30 Year 5%";
ds.AddLendingInstitutionRates(lir);

// Create a second entity and add it
LendingInstitutionRate lir2 = new LendingInstitutionRate();
lir2.PartitionKey = "LendingRates";
lir2.RowKey = "2";
lir2.LendingInstitution = "Fabrikam Loans";
lir2.ID = "2";
lir2.Instrument = "25 Year Fixed";
lir2.Year = 25;
lir2.Rate = 0.04;
lir2.Points = 0.0;
lir2.APR = 0.041;
lir2.Lock = "20 day";
lir2.InstrumentRateDisplayName = "Fabrikam Loans 25 Year 4%";
ds.AddLendingInstitutionRates(lir2);

// Save all pending changes in a single batch
ds.Save();

Console.WriteLine("\nEntities created.");
Console.WriteLine("\nRetrieving Lending Institution Entities...");

foreach (var r in ds.Select())
{
    Console.WriteLine("Institution name: {0}",
        r.LendingInstitution);
}

```

```

    }

    Console.WriteLine("\nPress <enter> to end...");
    Console.ReadLine();
}

```

code snippet 076576 Ch08_Code.zip/Program.cs

16. Press F5 to run the application. The entities will be stored in the development table storage account via the Azure storage emulator.

Here you see the magic come together. First, a new `LendingDataSource` object is instantiated. Note an immediate call to `DeleteLendingInstitutionRates` to remove any previously stored entities. This is because you are adding only two entities and the `RowKey` values are hard-coded. If you run this application a second time without deleting the previously added entities, you'll receive a runtime error.

After adding and saving the two entities, you immediately call the `Select` method to retrieve all the entities and display them in the console window. When the execution is complete, your console window should look like Figure 8-8.

You can further verify your table data in Visual Studio.



FIGURE 8-8

17. Open the Server Explorer and navigate down the tree to the `LendingInstitutionRate` node under Tables. When you double-click the table name, you should see the entities, as shown in Figure 8-9.

You now have mastered the basic components for implementing Azure table data into your solutions. You'll apply this knowledge in a couple of different ways — to build a WCF service that provides your table data to any calling program, and to build an Excel add-in that accesses the table data directly.

Using a WCF Service to Provide Azure Table Data

Accessing Windows Azure Table data directly using WCF Data Services is great, but it's not necessarily the only way to go. This section describes how to build an Azure-hosted WCF service that

accesses Azure table data and returns JavaScript Object Notation (JSON) to the calling application. In this scenario, the WCF service is a publicly available service, so it requires no authentication (whereas accessing the Windows Azure Table service directly always requires authentication, so that doesn't work well when you want to make the table data broadly available). Therefore, in this case, the WCF service authenticates to the Table service, but there is no requirement for the caller to authenticate to use the WCF service.

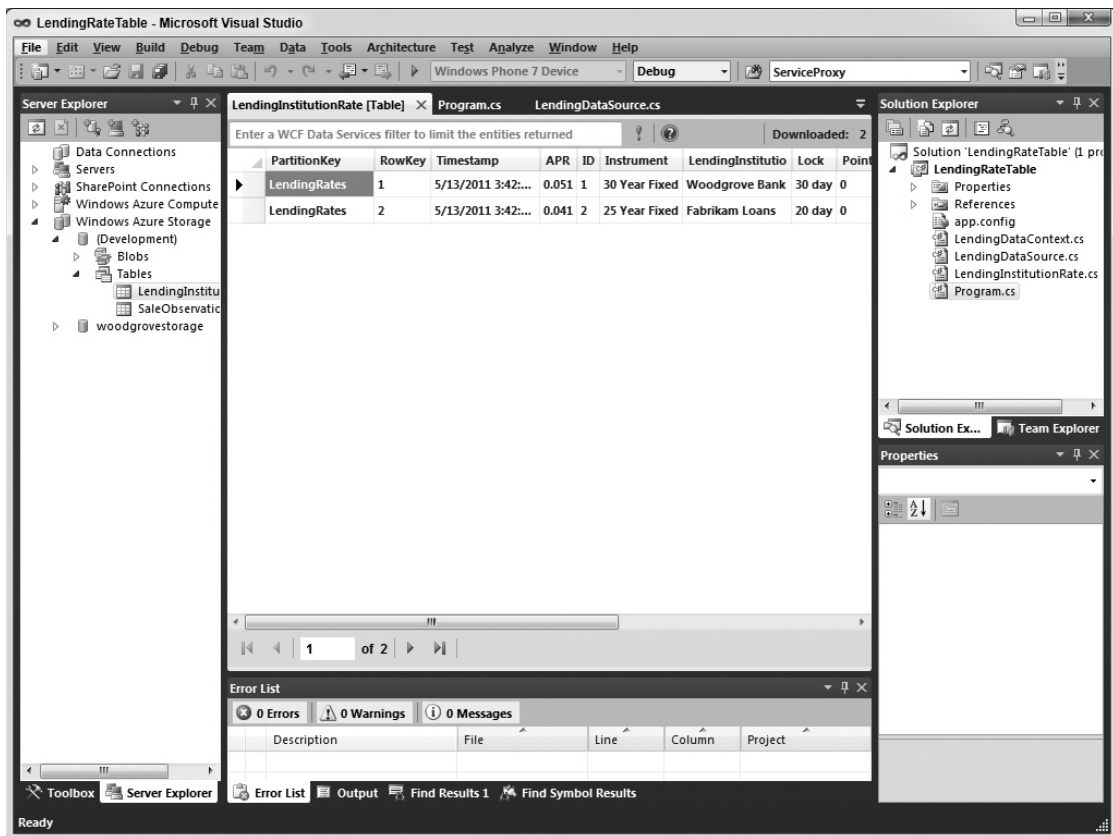


FIGURE 8-9

Perform the following steps to create a JSON-enabled Azure WCF service that accesses table data:

1. Open Visual Studio 2010 as Administrator.
2. Select File ⇨ New Project.
3. In the New Project dialog, click Cloud ⇨ Windows Azure Project, provide a name (e.g., **LendingRatesJSON**) and a location for the project, and then click OK.
4. In the New Windows Azure Project dialog, shown in Figure 8-10, click WCF Service Web Role and then click the top arrow between the panes.

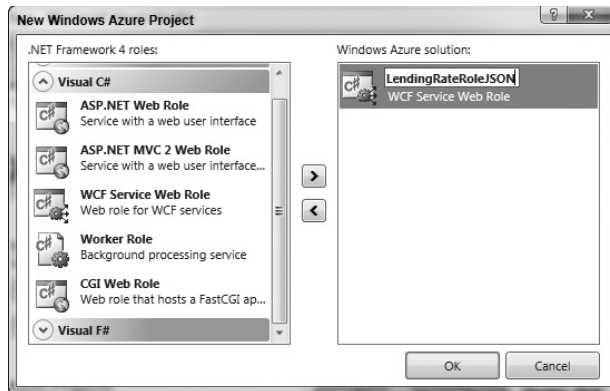


FIGURE 8-10

5. Hover over the WCFServiceWebRole in the right-hand pane, click the pencil icon, and rename the web role (e.g., **LendingRateRoleJSON**). Click OK.

This project creates a default *Service.svc* file that you will not use because your WCF service will return JSON to the calling program.

6. Right-click on the LendingRateRoleJSON project and choose Add ⇄ New Item. Choose Web from the installed templates list and select AJAX-enabled WCF Service. Rename the service (e.g., **LendingRateJSONService.svc**), and then click OK.
7. Replace all the code for the service with the following:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Text;

namespace LendingRateRoleJSON
{
    [AspNetCompatibilityRequirements(RequirementsMode =
        AspNetCompatibilityRequirementsMode.Allowed)]
    public class LendingRateJSONService:ILendingRateJSONService
    {
        // To use HTTP GET, add [WebGet] attribute. (Default ResponseFormat is
        // WebMessageFormat.Json)
        // To create an operation that returns XML,
        //     add [WebGet(ResponseFormat=WebMessageFormat.Xml)],
        //     and include the following line in the operation body:
        //         WebOperationContext.Current.OutgoingResponse.ContentType =
        //         "text/xml";

        public String[] GetLendingInstitutions()
        {
```

```
LendingDataSource ds = new LendingDataSource();

//Get the specific lending institution names (sorted) that have rates
var results = (from i in ds.Select()
               orderby i.LendingInstitution
               select i.LendingInstitution).Distinct();

return results.ToArray();
}

public IEnumerable<OutputInstitutionRate>
GetALendingInstitutionsRates(string lendingInstitutionName)
{
    //lendingInstitutionName = "Fabrikam Loans";
    LendingDataSource ds = new LendingDataSource();

    List<OutputInstitutionRate> oirList = new List<OutputInstitutionRate>();

    //Get all the rates a specific lending institution offers
    foreach (var i in
        ds.GetALendingInstitutionsRates(lendingInstitutionName))
    {
        OutputInstitutionRate oir = new OutputInstitutionRate();
        oir.APR = i.APR;
        oir.ID = i.ID;
        oir.Instrument = i.Instrument;
        oir.InstrumentRateDisplayName = i.InstrumentRateDisplayName;
        oir.LendingInstitution = i.LendingInstitution;
        oir.Lock = i.Lock;
        oir.Points = i.Points;
        oir.Rate = i.Rate;
        oir.Year = i.Year;
        oirList.Add(oir);
    }

    oirList = (from u in oirList
               orderby u.InstrumentRateDisplayName
               select u).ToList();

    return oirList;
}

public OutputInstitutionRate
GetASpecificLendingInstitutionsRate(string lendingRateIdentifier)
{
    //lendingRateIdentifier = "1";
    LendingDataSource ds = new LendingDataSource();

    List<OutputInstitutionRate> oirList = new List<OutputInstitutionRate>();

    //Get a specific lending rate for a lending institution by ID
    foreach (var i in
        ds.GetASpecificLendingInstitutionsRate(lendingRateIdentifier))
```

```

    {
        OutputInstitutionRate temp = new OutputInstitutionRate();
        temp.APR = i.APR;
        temp.ID = i.ID;
        temp.Instrument = i.Instrument;
        temp.InstrumentRateDisplayName = i.InstrumentRateDisplayName;
        temp.LendingInstitution = i.LendingInstitution;
        temp.Lock = i.Lock;
        temp.Points = i.Points;
        temp.Rate = i.Rate;
        temp.Year = i.Year;
        oirList.Add(temp);
    }

    OutputInstitutionRate oir = new OutputInstitutionRate();

    if (oirList.Count > 0)
    {
        oir.APR = oirList[0].APR;
        oir.ID = oirList[0].ID;
        oir.Instrument = oirList[0].Instrument;
        oir.InstrumentRateDisplayName =
            oirList[0].InstrumentRateDisplayName;
        oir.LendingInstitution = oirList[0].LendingInstitution;
        oir.Lock = oirList[0].Lock;
        oir.Points = oirList[0].Points;
        oir.Rate = oirList[0].Rate;
        oir.Year = oirList[0].Year;
    }
    else
    {
        oir = null;
    }

    return oir;
}
}
}

```

code snippet 076576 Ch08_Code.zip/LendingRateJSONService.svc.cs

Ignore the syntax errors for now because you need to add the interface for the AJAX-enabled service next. Primarily, what you have here are the three methods that will return the Azure table data to the calling program.

The first method, `GetLendingInstitutions()`, returns an array with a sorted list of the lending institutions that have interest rate data available. The `GetALendingInstitutionsRates()` method takes an input parameter of the lending institution's name and passes back a collection of all the interest rate programs it offers. Lastly, the `GetASpecificLendingInstitutionsRate()` method accepts an input parameter of a specific program identifier to return all the information about a given lending rate.

8. To add the interface, right-click on the LendingRateRoleJSON project and choose Add ⇨ New Item. Choose Code from the installed templates list and then select Interface. Rename the interface (e.g., **ILendingRateJSONService.cs**) and click OK.
9. Replace all the code for the interface with the following:



Available for
download on
Wrox.com

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

namespace LendingRateRoleJSON
{
    [ServiceContract(Namespace = "LendingRates", Name = "JSONService")]
    interface ILendingRateJSONService
    {
        [OperationContract, WebGet]
        string[] GetLendingInstitutions();

        [OperationContract, WebGet]
        IEnumerable<OutputInstitutionRate> GetALendingInstitutionsRates(String
            lendingInstitutionName);

        [OperationContract, WebGet]
        OutputInstitutionRate GetASpecificLendingInstitutionsRate(String
            lendingRateIdentifier);
    }
}
```

code snippet 076576 Ch08_Code.zip/ILendingRateJSONService.cs

The preceding interface defines the methods that the client application will call via GET. The client proxy will refer to these method names verbatim in the call to the JSON-enabled WCF service.

Now that you have the service and the interface defined, you need to finish with two classes and the WCF Data Services context and data source, like you built previously.

10. Right-click on the LendingRateRoleJSON project and choose Add ⇨ Class. Rename the class (e.g., **LendingInstitutionRate.cs**) and then click OK.
11. Replace all the code in the class with the following:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.WindowsAzure.StorageClient;

namespace LendingRateRoleJSON
{

```

```

public class LendingInstitutionRate : TableServiceEntity
{
    public string LendingInstitution { get; set; }
    public string ID { get; set; }
    public string Instrument { get; set; }
    public int Year { get; set; }
    public double Rate { get; set; }
    public double Points { get; set; }
    public double APR { get; set; }
    public string Lock { get; set; }
    public string InstrumentRateDisplayName { get; set; }

    public LendingInstitutionRate(string ID)
    {
        PartitionKey = "LendingRates";
        RowKey = ID;
    }

    public LendingInstitutionRate()
    {
        PartitionKey = "LendingRates";
    }
}

```

code snippet 076576 Ch08_Code.zip/LendingInstitutionRate.cs

Just like the earlier example, this is the class for the Azure table entity, complete with `PartitionKey` and `RowKey`. However, because the calling program has no notion of the structure of the Azure entity, you'll use a second class to model the structure of the object that will be returned as a JSON object. It's identical to the table entity class except it does not have the keys.

12. Right-click on the `LendingRateRoleJSON` project and choose `Add ⇨ Class`. Rename the class (e.g., **OutputInstitutionRate.cs**) and then click OK.
13. Replace all the code in the class with the following:



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Runtime.Serialization;

namespace LendingRateRoleJSON
{
    [DataContract]
    public class OutputInstitutionRate
    {
        [DataMember]
        public string LendingInstitution { get; set; }
        [DataMember]
        public string ID { get; set; }
    }
}

```

```

    [DataMember]
    public string Instrument { get; set; }
    [DataMember]
    public int Year { get; set; }
    [DataMember]
    public double Rate { get; set; }
    [DataMember]
    public double Points { get; set; }
    [DataMember]
    public double APR { get; set; }
    [DataMember]
    public string Lock { get; set; }
    [DataMember]
    public string InstrumentRateDisplayName { get; set; }
}
}

```

code snippet 076576 Ch08_Code.zip/OutputInstitutionRate.cs

The `OutputInstitutionRate` class is decorated with the `DataContract` attribute so that the class can be serialized and deserialized as it is passed between the service and the client. The `DataMember` attribute marks each of the properties so that they can be serialized as well.

Next, you add the data context class to your AJAX-enabled WCF service. This is essentially the same code you used previously.

14. Right-click on the `LendingRateRoleJSON` project and choose **Add ➦ Class**. Rename the class (e.g., **LendingDataContext.cs**) and then click **OK**.
15. Replace all the code in the class with the following:



Available for
download on
Wrox.com

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

namespace LendingRateRoleJSON
{
    public class LendingDataContext : TableServiceContext
    {
        public LendingDataContext(string baseAddress, StorageCredentials credentials)
            : base(baseAddress, credentials)
        {
        }
    }

    public IQueryable<LendingInstitutionRate> LendingInstitutionRate
    {
        get
        {
            return this.CreateQuery<LendingInstitutionRate>
                ("LendingInstitutionRate");
        }
    }
}

```

```

    }
  }
}

```

code snippet 076576 Ch08_Code.zip/LendingDataContext.cs

16. To add the data source to your service, right-click on the LendingRateRoleJSON project and choose Add ⇨ Class. Rename the class (e.g., **LendingDataSource.cs**) and then click OK.
17. Replace all the code in the class with the following:



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

namespace LendingRateRoleJSON
{
    public class LendingDataSource
    {
        private static CloudStorageAccount storageAccount;
        private LendingDataContext context;

        static LendingDataSource()
        {
            try
            {
                storageAccount =
CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;
AccountName={youraccountname};AccountKey={youraccountkey}");

                CloudTableClient c = new CloudTableClient(
                    storageAccount.TableEndpoint.AbsoluteUri,
                    storageAccount.Credentials);

                if (!c.DoesTableExist("LendingRates"))
                {
                    CloudTableClient.CreateTablesFromModel(typeof(LendingDataContext),
                        storageAccount.TableEndpoint.AbsoluteUri,
                        storageAccount.Credentials);
                }
            }
            catch (Exception e)
            {
                //System.Windows.Forms.MessageBox.Show(e.Message);
            }
        }

        public LendingDataSource()
        {

```

```
        this.context = new LendingDataContext(
            storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);

        this.context.RetryPolicy = RetryPolicies.Retry(3,
            TimeSpan.FromSeconds(1));
    }

    public IEnumerable<LendingInstitutionRate> Select()
    {
        var results = from g in this.context.LendingInstitutionRate
            where g.PartitionKey == "LendingRates"
            select g;

        return results;
    }

    public IEnumerable<LendingInstitutionRate> GetALendingInstitutionsRates(String
lendingInstitutionName)
    {
        var results = from g in this.context.LendingInstitutionRate
            where g.PartitionKey == "LendingRates" && g.LendingInstitution ==
lendingInstitutionName
            select g;

        return results;
    }

    public IEnumerable<LendingInstitutionRate>
GetASpecificLendingInstitutionsRate(String lendingRateIdentifier)
    {
        var results = from g in this.context.LendingInstitutionRate
            where g.PartitionKey == "LendingRates" && g.ID ==
lendingRateIdentifier
            select g;

        return results;
    }

    public void AddLendingInstitutionRate(LendingInstitutionRate newItem)
    {
        this.context.AddObject("LendingInstitutionRate", newItem);
        this.context.SaveChanges();
    }

    // Use to add multiple lending institutions in a batch
    // Call Save() after adding the institutions
    public void AddLendingInstitutionRates(LendingInstitutionRate newItem)
    {
        this.context.AddObject("LendingInstitutionRate", newItem);
    }

    public void Save()
```



```

    {
        this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
    }

    public void DeleteLendingInstitutionRate(LendingInstitutionRate item)
    {
        this.context.DeleteObject(item);
        this.context.SaveChanges();
    }

    public void DeleteLendingInstitutionRates()
    {
        foreach (var p in this.Select())
        {
            this.context.DeleteObject(p);
        }
        this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
    }
}

```

code snippet 076576 Ch08_Code.zip/LendingDataSource.cs

Because the preceding code is essentially the same code you used previously, we won't step through it again. You are just making the same methods available to the WCF service.

Notice in the preceding code that you must fully replace the placeholder value {your accountname} (including braces) with your storage account name and the placeholder {youraccountkey} with your storage account access key.

18. Right-click on the LendingRateRoleJSON project and choose Add ➞ Reference. Add the following reference from the .NET tab: System.Data.Services.Client. Click OK.

With the coding complete for the service, the last part to wire up is the web.config file. Open the web.config file for the service and you'll notice that the endpoint and service information in the <system.serviceModel> have been updated. In addition, the <behavior> element in the <endpointBehavior> element has <enableWebScript /> so that the service can send and receive JSON.

19. Add the following XML fragment just under the <serviceBehaviors> element:

```

<behavior name="LendingRateJSONServiceBehavior">
  <serviceMetadata httpGetEnabled="true" />
  <serviceDebug includeExceptionDetailInFaults="true" />
</behavior>

```

20. Find the <service name="LendingRateRoleJSON.LendingRateJSONService"> element and modify it to include the behaviorConfiguration attribute, as follows:

```

<service name="LendingRateRoleJSON.LendingRateJSONService"
  behaviorConfiguration="LendingRateJSONServiceBehavior">

```

This ties the specific service behavior to the service.

The last step is to modify the `contract` attribute on the service. Rather than `contract="LendingRateRoleJSON.LendingRateJSONService"`, you need it to be the interface, not the service. Here the interface behaves like the contract for the service, as observed in the interface previously.

21. Change the contract to the following:

```
contract="LendingRateRoleJSON.ILendingRateJSONService"
```

When completed, your `web.config` `system.serviceModel` should look like Figure 8-11.

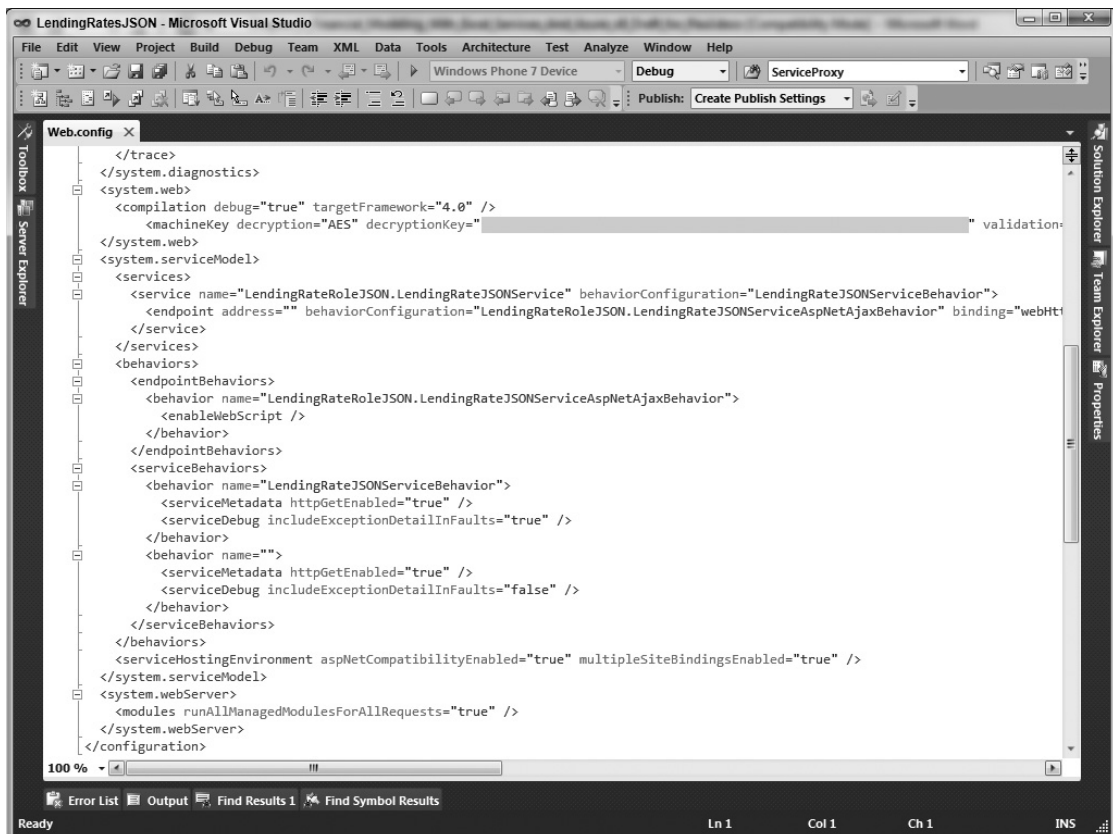


FIGURE 8-11

Once you have built your service, take the steps necessary to deploy the service to your Azure account in a staging environment. Note the DNS name for your deployment, as you will need this for your client application. Following are four code snippets that you'll use to create four files. The first is the `LendingRates_TestHarness.html` file. From this HTML page you'll call methods in the second file, `LendingServiceProxy.js`, to retrieve the desired information from the AJAX-enabled WCF service. The retrieved data will populate drop-down lists for end user selection. The purpose

here is to test your WCF service. The two additional files, `LendingStyles.css` and `Rate.htm`, help enhance the presentation of the lending rate information on the web page.

Note that two other JavaScript files are referenced: `jquery.js` and `jquery-templates.js`. This example used the jQuery JavaScript Library v1.3.2 and jTemplates 0.7.5, so you'll need to reference these versions or later. Information about these files and links to download them are available at jquery.org.



Available for
download on
Wrox.com

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script src="jquery.js" type="text/javascript"></script>
    <script src="jquery-jtemplates.js" type="text/javascript"></script>
    <script src="LendingServiceProxy.js" type="text/javascript"></script>
    <link href="LendingStyles.css" rel="stylesheet" type="text/css" />
    <title>Lending Institution Rate Selection</title>
  </head>

  <body>
    <div>
      <div id="Institutions">
        <div id="InstitutionSelection">
          <b>Lending Agency:</b>
          <select id='lendinginstdd' style='width:175px'
            onchange='dropdownlending_onchange(this)'></select>
          <br /><br />
          <b>Lending Program:</b>
          <select id='lendingratesdd' style='width:175px'
            onchange='dropdownlendingrates_onchange(this)'></select>
          <br /><br />
        </div>
        <div id="RateDisplay">
          
          <div id="SingleRate">
          </div>
        </div>
      </div>
    </div>
  </body>

  <script type="text/javascript">

    var proxy = new LendingServiceProxy();

    // Code executes after page loading is complete
    $(document).ready(function ()
    {
      // Retrieve all lending institutions
      $("#RetrievingImg").removeClass("Hidden");
      proxy.GetLendingInstitutions(institutionsRetrieved,
        serviceDefaultErrorHandler);
    });
```

```
// institutionsRetrieved()
// called when the service returns a list of lending institutions
// loads lendinginstdd dropdown list
function institutionsRetrieved(results) {
    if (results) {
        if (results.length > 0) {
            var list = document.getElementById('lendinginstdd');
            // Clear options before loading
            list.selectedIndex = 0;
            list.options.length = 0;
            for (n in results) {
                var opt = document.createElement("option");
                opt.text = results[n];
                opt.value = results[n];
                list.options.add(opt);
            }
        }
        else {
            alert("no lending institutions found");
        }
    }
    $("#RetrievingImg").addClass("Hidden");
}

// ratesRetrieved()
// called when the service returns a list of lending institution rates
// loads lendingratesdd dropdown list
function ratesRetrieved(results)
{
    if (results) {
        if (results.length > 0) {
            var list = document.getElementById('lendingratesdd');
            // clear options before loading
            list.selectedIndex = 0;
            list.options.length = 0;
            for (n in results) {
                var opt = document.createElement("option");
                opt.text = results[n].InstrumentRateDisplayName;
                opt.value = results[n].ID;
                list.options.add(opt);
            }
        }
        else {
            alert("no lending rates found for this institution");
        }
    }
    $("#RetrievingImg").addClass("Hidden");
}

// loadexcelwebpart()
// called when the service returns a specific lending institution rate
// loads the Excel Web Access web part
function loadexcelwebpart(results) {
    if (results) {
```

```

        //alert(results.LendingInstitution);
        var templateItem = $("#SingleRate").setTemplateURL("Rate.htm", null,
{ filter_data: false });
        templateItem.processTemplate(results);
    }
    else {
        alert("no specific lending rate found for this institution");
    }
}

// dropdownlending_onchange()
// Event handler for changes with the selection drop down
function dropdownlending_onchange(event) {
    if (event == null || event.selectedIndex < 0)
        return;
    var listIndex = event.options[event.selectedIndex].value;
    // If first item in the list not an actual selection.
    if (listIndex == "null")
        return;
    // Retrieve the lending rate programs offered by lending institution
    var lendingInstitutionName = event.options[event.selectedIndex].text;
    $("#RetrievingImg").removeClass("Hidden");
    proxy.GetALendingInstitutionsRates(lendingInstitutionName, ratesRetrieved,
serviceDefaultErrorHandler);
}

// dropdownlendingrates_onchange()
// Event handler for changes with the selection drop down
function dropdownlendingrates_onchange(event) {
    if (event == null || event.selectedIndex < 0)
        return;
    var listIndex = event.options[event.selectedIndex].value;
    // If first item in the list not an actual selection.
    if (listIndex == "null")
        return;
    // Retrieve the lending rate programs offered by lending institution
    var lendingRateID = event.options[event.selectedIndex].value;
    proxy.GetASpecificLendingInstitutionsRate(lendingRateID, loadexcelwebpart,
serviceDefaultErrorHandler);
}

// serviceDefaultErrorHandler()
// called when service returns an error
function serviceDefaultErrorHandler(xhr, context, method)
{
    alert(xhr.statusText);
    if (!$("#RetrievingImg").hasClass("Hidden"))
    $("#RetrievingImg").addClass("Hidden");
}
</script>
</html>

```

code snippet 076576 Cb08_Code.zip/LendingRates_TestHarnes.html

As soon as the HTML page is loaded, the `GetLendingInstitutions` method in the proxy is called. Then subsequent methods are called on the proxy in response to the `onchange` event as the end users make their selections in the drop-down lists.

The following `LendingServiceProxy.js` file handles the calls between the client and the service. Parameters are passed where appropriate, and the returned data is parsed and loaded into the response object for use. Note that the method and data parameter names passed into the `_doAjax` method must be identical to the method and data parameter names used in the WCF service `ILendingRateJSONService` interface.



Available for
download on
Wrox.com

```
// Proxy for calling WCF JSON service in Windows Azure
// constructor for the LendingServiceProxy
LendingServiceProxy = function()
{
    this._baseUrl =
    "http://YourAzureGUID.cloudapp.net/LendingRateJSONService.svc/"
    ;
};

LendingServiceProxy.prototype =
{
    GetLendingInstitutions: function (success, error) {
        this._doAjax("GetLendingInstitutions", null, success, error);
    },

    GetALendingInstitutionsRates: function (lendingInstitutionName, success, error)
    {
        var data = { lendingInstitutionName: lendingInstitutionName };
        this._doAjax("GetALendingInstitutionsRates", data, success, error);
    },

    GetASpecificLendingInstitutionsRate: function (lendingRateIdentifier, success,
    error) {
        var data = { lendingRateIdentifier: lendingRateIdentifier };
        this._doAjax("GetASpecificLendingInstitutionsRate", data, success, error)
    },

    _defaultErrorHandler: function(xhr, status, error)
    {
        alert(xhr.statusText);
    },

    _doAjax: function(method, data, fnSuccess, fnError)
    {
        if (!data) data = {};

        if (!fnError) fnError = this._defaultErrorHandler;

        $.ajax({
            type: "GET",
            url: this._baseUrl + method,
            data: data,
            contentType: "application/json; charset=utf-8",
            dataType: "json",
```

```

    success: fnSuccess,
    error: fnError,
    dataFilter: function(data)
    {
        var response;

        if (typeof (JSON) !== "undefined" && typeof (JSON.parse) === "function")
            response = JSON.parse(data);
        else
            response = val("(" + data + ")");

        if (response.hasOwnProperty("d"))
            return response.d;
        else
            return response;
    }
    });
}
};

```

code snippet 076576 Ch08_Code.zip/LendingServiceProxy.js

Following is a simple CSS file, `LendingStyles.css`:



Available for
download on
Wrox.com

```

body {background-color:#D3D7DA;}

h3 {margin-top: 4px;}
li {margin-bottom:5px;}

#InstitutionSelection {border-width:thin; width:200px;
background-color: #E1E1FF; font-size:13px; font-weight:bold; float:left; }
#InstitutionSelection a {text-decoration:none; color:Black;}

#RateDisplay {float:left; margin-left:20px;}

#SingleRate {width:400px; padding-right:4px; background-color: #FFF8E6;}

.Hidden {display:none; visibility:hidden;}

```

code snippet 076576 Ch08_Code.zip/LendingStyles.css

`LendingStyles.css` provides a simple layout style for displaying the lending institutions and their rates for end user viewing and interaction.

The final file to add to enhance the test harness user presentation is the `Rate.htm` file.



Available for
download on
Wrox.com

```

<h3>
    {<T.LendingInstitution>
    <br />
    APR: <b>{<T.APR>}</b>
    <br />
    Name: <b>{<T.Lock>}</b>
    <br />

```

```

Instrument: <b>{$T.Instrument}</b>
<br />
Instrumentname: <b>{$T.InstrumentRateDisplayName}</b></h3>
<p>
</p>

```

code snippet 076576 Ch08_Code.zip/Rate.htm

In the `LendingRates_TestHarness.html` page's JavaScript function `loadexcelwebpart`, the `Rate.htm` file is used as a template to display the additional information for the user-selected lending institution and interest rate. This additional information includes the APR and the number of days that the user can lock the interest rate.

Place the `LendingRates_TestHarness.html`, `LendingServiceProxy.js`, `LendingStyles.css`, and `Rate.htm` files in a common directory such as `TestHarness`, with the `jquery.js` and `jquery-templates.js` files. There is also a `.gif` file available in the `.zip` file for this chapter that you can use to simulate progress while the data is being retrieved from the WCF service, but it is not required for successful execution. Double-click the `LendingRates_TestHarness.html` file in your folder to open in a browser. Your code should successfully execute, with a result like Figure 8-12.

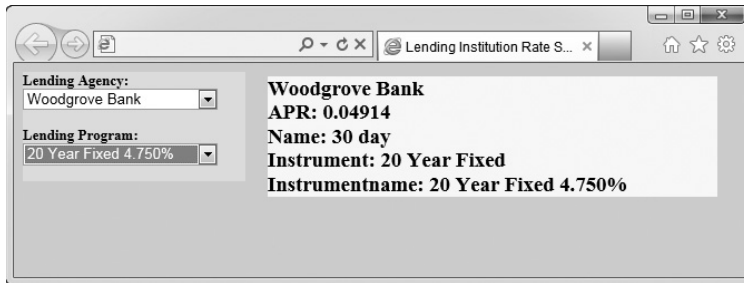


FIGURE 8-12

EXTENDING AN EXCEL FINANCIAL MODEL TO LEVERAGE AZURE TABLE DATA

You now have the fundamentals in place for leveraging Windows Azure table data directly from an application using WCF Data Services. You have also seen how to set up an Azure WCF service that serves as a public-facing endpoint for your Azure table data and returns JSON to calling programs. In the next two sections you will put both bits of knowledge to practical use by extending an Excel financial model to benefit from Azure table data, regardless of whether the model is running on the client or on SharePoint under Excel Services.

This scenario extends the mortgage financial model. Up to this point, when using the model, the real estate agent provides three input parameters: the mortgage amount, the term in years, and the fixed interest rate. In this case, the agent needs to know these values. Of course, knowing the mortgage amount and the number of years for which the buyer wants the loan is a simple matter. What is not known is the potentially large number of interest rates being offered by different lending institutions

on a given day. These rates change daily and can vary widely across lending institutions. That's where the Windows Azure Table storage comes in.

To frame this example, suppose you develop an application that crawls the Web, collects lending institution rate data, and places it in an Azure table with an entity structure like the one you have been using all along. Then, once you have the data collected, you can make it available to your financial model so that real estate agents can simply make appropriate lending rate selections and the model will be updated appropriately. This way, agents no longer need to look up daily rates from diverse sources. The authoritative data is available to them directly within their working context. That's a productivity gain and a huge win for end users!

Creating an Excel Client Add-In

Figure 8-13 shows a mortgage financial model. You will be building a financial model like this for your Excel client add-in. In the Excel client, you will be accessing the table data directly using WCF Data Services. End users will retrieve the lending rate table data with a button click, and then select a lending institution and the year term with its corresponding rate. The APR, the number of days for which the borrower can lock the rate, and the points they will need to pay for the loan at this rate are retrieved from the table and placed in the model below the computed monthly payment for reference.

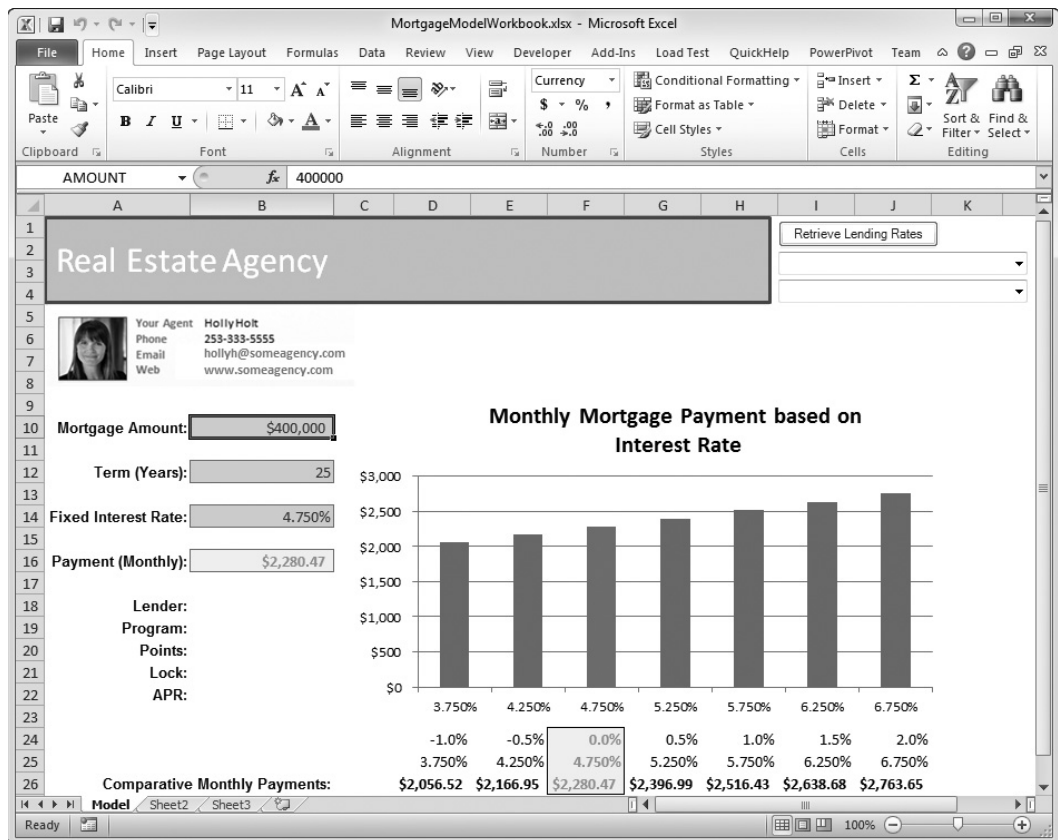


FIGURE 8-13

This is considered a document-level solution, as you want the add-in to load only when the mortgage financial model workbook is opened.

- 1. Open Visual Studio 2010 as Administrator.
- 2. Select File ⇨ New Project.
- 3. In the New Project dialog, click Office 2010 then the Excel 2010 Workbook project template, provide a name (e.g., **ExcelMortgageAzure**) and a location for the project, and then click OK.
- 4. In the Visual Studio Tools for Office Project Wizard, click Create a New Document, provide a name for the document (e.g., **ExcelMortgageAzure**), and then click OK.
- 5. To begin the model, enter text, starting in cell A2, and modify the cell for bold font with the text adjusted to the right so that your sheet looks similar to Figure 8-14.

If you haven't used Excel before, it uses the concept of *named ranges*. Any cell or collection of cells can have an arbitrary name associated with it. This named range can then be used within formulas and accessed programmatically. In order to programmatically place data into each cell in column B that corresponds to its respective label, you need to provide a named range for each of these cells.

	A	B	C
1			
2	Mortgage Amount:		
3			
4	Term (Years):		
5			
6	Fixed Interest Rate:		
7			
8	Payment (Monthly):		
9			
10	Lender:		
11	Program:		
12	Points:		
13	Lock:		
14	APR:		

FIGURE 8-14

- 6. Click cell B2. Then click in the area just left of the *fx*, type **AMOUNT**, and press Enter. Your result should look like Figure 8-15.
- 7. Complete creating named ranges for each of the cells; TERM, FIR, skip Payment (Monthly) for now, LENDER, INSTRUMENT, POINTS, LOCK, and APR, respectively.
- 8. Because Payment is a computed value, you use a formula to calculate this. Click cell B8 and type **=ROUNDUP(-PMT(FIR/12,TERM*12,AMOUNT,,0),2)**.

Notice in this formula that the named ranges you created are being used. As you type the formula, Excel provides an IntelliSense experience so that you can choose the valid values available. When you finish entering the formula and press Enter, you will see a #NUM! error in the cell. This is normal because you have not yet entered values in the other named range cells.
- 9. To validate that your formula is correct, enter **400000** for the Mortgage Amount, **25** for the Term (Years), and **0.0475** for the Fixed Interest Rate. You may need to widen column B2 for the computed value to show, but it should be \$2,280.47.

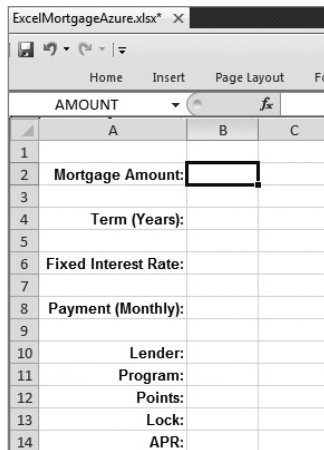


FIGURE 8-15

Your financial model is now complete with respect to a real estate agent needing to know and manually enter the input values. Now let's take it to the next level.

First, as with the previous sample, you'll add classes for the lending institution rate, the data context, and the data source. Because you have stepped through this before, these instructions are abbreviated.

10. Add references to `Microsoft.WindowsAzure.StorageClient` and `System.Data.Services.Client` to the project. If you are prompted to retarget to .NET Framework 4, confirm the prompts to do so.
11. In the Solution Explorer, right-click on the project and add a new class named `LendingInstitutionRate.cs`.
12. Replace all the code with the following:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure.StorageClient;

namespace ExcelMortgageAzure
{
    public class LendingInstitutionRate:TableServiceEntity
    {
        public string LendingInstitution { get; set; }
        public string ID { get; set; }
        public string Instrument { get; set; }
        public int Year { get; set; }
        public double Rate { get; set; }
        public double Points { get; set; }
        public double APR { get; set; }
        public string Lock { get; set; }
        public string InstrumentRateDisplayName { get; set; }

        public LendingInstitutionRate(string ID)
    }
}
```

```
        {
            PartitionKey = "LendingRates";
            RowKey = ID;
        }

        public LendingInstitutionRate()
        {
            PartitionKey = "LendingRates";
        }
    }
}
```

code snippet 076576 Ch08_Code.zip/LendingInstitutionRate.cs

13. Add a new class named **LendingDataContext.cs**.

14. Replace all the code with the following:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

namespace ExcelMortgageAzure
{
    public class LendingDataContext:TableServiceContext
    {

        public LendingDataContext(string baseAddress, StorageCredentials credentials)
            : base(baseAddress, credentials)
        {

        }

        public IQueryable<LendingInstitutionRate> LendingInstitutionRate
        {
            get
            {
                return
                    this.CreateQuery<LendingInstitutionRate>("LendingInstitutionRate");
            }
        }
    }
}
```

code snippet 076576 Ch08_Code.zip/LendingDataContext.cs

15. Add a new class named **LendingDataSource.cs**.

16. Replace all the code with the following:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;

namespace ExcelMortgageAzure
{
    public class LendingDataSource
    {
        private static CloudStorageAccount storageAccount;
        private LendingDataContext context;

        static LendingDataSource()
        {
            try
            {
                storageAccount =
                CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;
                AccountName={youraccountname};AccountKey={youraccountkey}");

                CloudTableClient c = new CloudTableClient(
                    storageAccount.TableEndpoint.AbsoluteUri,
                    storageAccount.Credentials);

                if (!c.DoesTableExist("LendingInstitutionRate"))
                {
                    CloudTableClient.CreateTablesFromModel(typeof(LendingDataContext),
                        storageAccount.TableEndpoint.AbsoluteUri,
                        storageAccount.Credentials);
                }
            }
            catch (Exception e)
            {
                System.Windows.Forms.MessageBox.Show(e.Message);
            }
        }

        public LendingDataSource()
        {
            this.context = new LendingDataContext(
                storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);

            this.context.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1));
        }

        public IEnumerable<LendingInstitutionRate> Select()
        {
            var results = from g in this.context.LendingInstitutionRate
```

```
        where g.PartitionKey == "LendingRates"
        select g;

    return results;
}

public void AddLendingInstitutionRate(LendingInstitutionRate newItem)
{
    this.context.AddObject("LendingInstitutionRate", newItem);
    this.context.SaveChanges();
}

// Use to add multiple products in a batch
// call Save() after adding the products
public void AddLendingInstitutionRates(LendingInstitutionRate newItem)
{
    this.context.AddObject("LendingInstitutionRate", newItem);
}

public void Save()
{
    this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
}

public void DeleteLendingInstitutionRate(LendingInstitutionRate item)
{
    this.context.DeleteObject(item);
    this.context.SaveChanges();
}

public void DeleteLendingInstitutionRates()
{
    foreach (var p in this.Select())
    {
        this.context.DeleteObject(p);
    }
    this.context.SaveChanges
(System.Data.Services.Client.SaveChangesOptions.Batch);
}
}
```

code snippet 076576 Ch08_Code.zip/LendingDataSource.cs

- 17.** Click the Save All icon on the Visual Studio toolbar.
- 18.** Right-click on the project and select Properties.
- 19.** If you were not previously prompted to target the .NET Framework 4, on the Application tab, select .NET Framework 4 from the drop-down below Target framework. When prompted, respond Yes, but make sure you saved your work first.

20. Press F6 to confirm that your project builds. As you did previously, make sure you fully replace the placeholder values {youraccountname} and {youraccountkey} with your appropriate values.

You now have the financial model and data access pieces in place so that you can access Azure table data. Now you'll add the UI elements for the user to interact with to load the named ranges with the retrieved values from the table.
21. In the Solution Explorer, expand the ExcelMortgageAzure.xls node and double-click on Sheet1.cs to open the workbook sheet.
22. Open the toolbox and place a Windows Forms button on the sheet in the area of cell D2. Name the button **buttonRetrieveRates** and change its Text property to **Retrieve Lending Rates**.
23. Double-click on the button to create the click event handler.
24. Just prior to the Sheet1_Startup method, add the following class-level variable:

```
List<LendingInstitutionRate> liRates = new List<LendingInstitutionRate>();
```

25. In the buttonRetrieveRates_Click method, add the following code:



Available for
download on
Wrox.com

```
//Instantiate the data source
LendingDataSource ds = new LendingDataSource();

//Retrieve Azure data from table storage and save into a list for reuse
foreach (var lir in ds.Select())
{
    LendingInstitutionRate tempLir = new LendingInstitutionRate();
    tempLir.ID = lir.ID;
    tempLir.LendingInstitution = lir.LendingInstitution;
    tempLir.Instrument = lir.Instrument;
    tempLir.Year = lir.Year;
    tempLir.Rate = lir.Rate;
    tempLir.Points = lir.Points;
    tempLir.APR = lir.APR;
    tempLir.Lock = lir.Lock;
    tempLir.InstrumentRateDisplayName = lir.InstrumentRateDisplayName;

    liRates.Add(tempLir);
}

//Load a variable with only the lending institutions
var lendingInstitutions = (from i in liRates
    select i.LendingInstitution).Distinct();

//Load the combobox with the institution names
comboBoxLenders.DataSource = lendingInstitutions.ToArray();
comboBoxLenders.DisplayMember = "LendingInstitution";
```

code snippet 076576 Cb08_Code.zip/Sheet1.cs

Here you call the `Select` method on the data source object to retrieve all the entities from table storage and cache them in a `List` object. You then use a LINQ query to select `.Distinct` on the institution name and use this to load the `ComboBox` drop-down with the lending institution names.

26. Return to the `ExcelMortgageAzure.xlsx` tab and place a `ComboBox` from the toolbox just below the button. Change its `Name` property to `comboBoxLenders`. Delete the value in its `Text` property.
27. Double-click the `ComboBox` to create its event handler.
28. Place the following code inside the method:



Available for
download on
Wrox.com

```
//On selection change selects only the institution's
//rates from the list
var lendingInstitutionRates = from i in liRates
    where i.LendingInstitution ==
        comboBoxLenders.SelectedValue.ToString()
    orderby i.LendingInstitution,
        i.Instrument, i.Year, i.Rate
    select i;

//Load the rates combobox for the lending institution
comboBoxRates.DataSource = lendingInstitutionRates.ToArray();
comboBoxRates.DisplayMember = "InstrumentRateDisplayName";
comboBoxRates.ValueMember = "ID";
```

code snippet 076576 Ch08_Code.zip/Sheet1.cs

The selection change event for the `Lenders ComboBox` queries the `liRates` list and selects all the lending institutions that match the value of the end user's selection. Matching objects in the list are selected, sorted, and used as the `DataSource` for the `comboBoxRates` object.

29. Return to the `ExcelMortgageAzure.xlsx` tab and place a `ComboBox` from the toolbox just below the previous `ComboBox`. Change its `Name` property to `comboBoxRates`. Delete the value in its `Text` property.
30. Double-click the `ComboBox` to create its event handler.
31. Place the following code inside the method:



Available for
download on
Wrox.com

```
//Select the specific rate for the institution and
//apply the data appropriately to
//the named ranges on the spreadsheet.
var lendingInstitutionRates = from i in liRates
    where i.ID ==
        comboBoxRates.SelectedValue.ToString()
    select i;

//Load the named ranges
foreach (var lir in lendingInstitutionRates)
{
    this.TERM.Cells.Value = lir.Year;
```



```

this.FIR.Cells.Value = lir.Rate;
this.LENDER.Cells.Value = lir.LendingInstitution;
this.POINTS.Cells.Value = lir.Points;
this.INSTRUMENT.Cells.Value = lir.InstrumentRateDisplayName;
this.LOCK.Cells.Value = lir.Lock;
this.APR.Cells.Value = lir.APR;
}
    
```

code snippet 076576 Ch08_Code.zip/Sheet1.cs

Lastly, when end users select the specific rate they want to apply to the financial model, this `OnSelectionChanged` event fires. In this method, the specific rate object is retrieved from the `liRates` list. Its property values are used to set the values for each of their corresponding named ranges in the Excel sheet via the Excel object model. Excel automatically recalculates the sheet when the values change in the cells.

32. Press F5 to run the project. Your financial model should look similar to Figure 8-16.

	A	B	C	D	E	F	G	H
1								
2	Mortgage Amount:	400,000.00			Retrieve Lending Rates			
3								
4	Term (Years):	20			Fabrikam Loans			
5								
6	Fixed Interest Rate:	4.25%			20 Year Fixed 4.250%			
7								
8	Payment (Monthly):	\$2,476.94						
9								
10	Lender:	Fabrikam Loans						
11	Program:	20 Year Fixed 4.250%						
12	Points:	2.75						
13	Lock:	40 day						
14	APR:	4.74%						
15								
16								

FIGURE 8-16

As you can see from this simple financial model, the opportunities are vast for building rich add-ins that reach out to external sources for data; and the Office client applications offer deep object models for developers to tap into the benefits of developing Office client solutions.

Using the ECMAScript Object Model with the Excel Web Access Web Part

Excel workbooks running on the desktop are powerful and handy tools, but workbooks tend to be replicated widely throughout an organization, and this can be problematic. For example, it's common to send an e-mail with an Excel document attached. The problem with this is that workbooks tend to morph as individuals add their own customizations, and then one begins to wonder if these broadly distributed documents still maintain integrity with the original computations in the model. And what if the original model is updated? One would need to broadly distribute the updated model again.

This is where Excel Services under SharePoint provides tremendous value. The author of the model can upload it to SharePoint, where the model becomes broadly available across the organization, from a single location, via the Web. From here, end users can open the document in either the Excel client or the new Excel Web App in SharePoint 2010.

However, there is one caveat; the Excel Web App does not support the same notion of add-ins that can be implemented in the client. Therefore, the add-in that you built in the previous section will not access the Azure table data when running under the Excel Web App. The button to retrieve the data and the two drop-down lists are simply not accessible in the Excel Web App.

Therefore, to provide end users with the same functionality to easily manipulate the mortgage financial model in the Excel Web App, you need to employ a couple of other nice features in SharePoint: the Excel Web Access web part and the Content Editor web part, in combination with the new ECMAScript object model for Excel. This is also where your JSON-enabled WCF service comes into play. You will call this web service from your ECMAScript code in the Content Editor web part. First, however, you need to set up everything on the SharePoint side.



To keep this exercise straightforward, you are not going to create a Visual Studio Tools for Office (VSTO) deployment package for your Excel document-level add-in. That discussion is beyond the scope of this chapter. Here you will simply upload a saved version of your Excel mortgage financial model to SharePoint.

In the following steps, you will upload your financial model workbook to SharePoint and use a Content Editor web part to access Azure table data via a WCF service and manipulate the financial model in an Excel Web Access web part.

1. Open your ExcelMortgageAzure project in Visual Studio and press F5.
2. When Excel opens, chose File ⇨ Save As and save it to any location on your disk with the name **ExcelMortgageAzureSP.xlsx**. Close Excel.
3. On your SharePoint site, navigate to any document library and upload the ExcelMortgageAzureSP.xlsx file.

When you view the document in the Excel Web App, you can see that the Web App recognizes that unsupported features are present in the document — that is, those not available in the browser. Click on Details in the information bar and you can see the types of features that are not supported, as shown in Figure 8-17. This restricts end users from editing the document in the browser, so they cannot manipulate the model.

To get past this restriction, you will create a SharePoint web part page that contains an Excel Web Access web part and a Content Editor web part. You will then add ECMAScript code to the Content Editor web part to access the Azure table data and manipulate the document with the Excel ECMAScript object model.

4. In SharePoint, select Site Actions ⇨ View All Site Content ⇨ Create ⇨ Page ⇨ Web Part Page ⇨ Create.

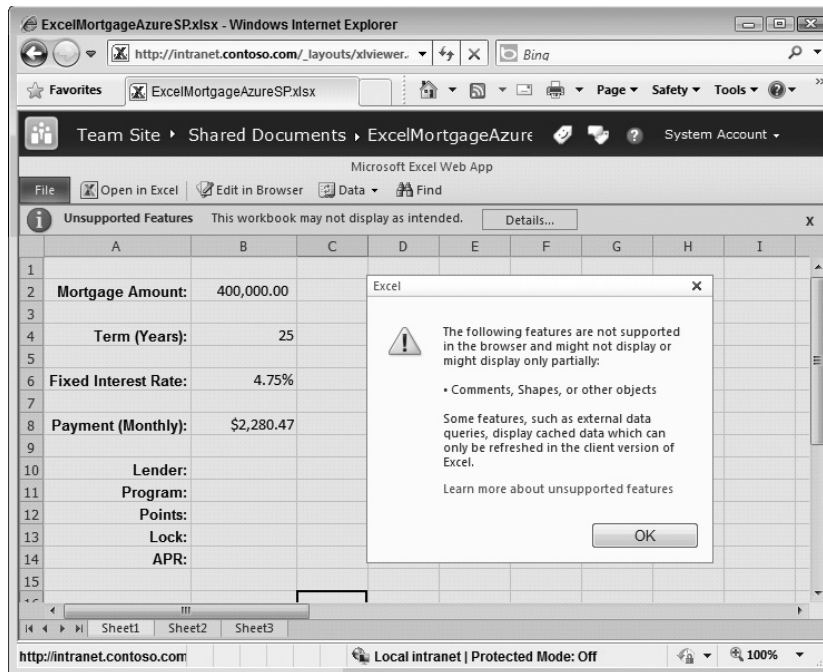


FIGURE 8-17

5. For the name, enter **MortgageModel**. Select Header, Left Column, Body for the template, and then choose Site Pages for the document library. Click Create.
6. In the Body container, click Add a Web Part.
7. Select Business Data under Categories, and then select Excel Web Access under Web Parts. Click Add.
8. Click the link “Click here” to open the tool pane.
9. In the tool pane, click the ellipses (...) beside the Workbook text box.
10. In the Select an Asset — Webpage dialog, click on the document library to which you uploaded your model and select ExcelMortgageAzureSP.xlsx. Click OK.
11. In the tool pane, set the Type of Toolbar to None, click the checkbox for Typing and Formula Entry, change Height to 375, and set the Chrome Type to None. Click OK.
12. In the left column, click Add a Web Part.
13. Click Media and Content under Categories, and Content Editor under Web Parts. Click Add.
14. Hover over the Content Editor title, click the drop-down arrow, and select Edit Web Part.
15. Set the Title to Select Lending Institution, the width to 200, and the Chrome to Title Only. Click OK.
16. In the SharePoint ribbon, choose Stop Editing.

Although you cannot interactively access the table data at this point, you can give your model a quick test under Excel Services in the Excel Web Access web part. In the web part, provide values for the mortgage amount, years, and interest rate to see the model adjust. It's good to see the model working at this level, and the following steps will bring this solution on par with the client add-in you built. Some of the previous code used in your test harness to call the Azure WCF service will look familiar to you here, but it is modified with the Excel ECMAScript object model to push retrieved data values into the document cells.

17. Add the following code snippet file to the directory where you saved your `LendingRates_TestHarness.html` file:



Available for
download on
Wrox.com

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script src="/Shared%20Documents/MortgageAssetsJS/jquery.js"
type="text/javascript"></script>
    <script src="/Shared%20Documents/MortgageAssetsJS/jquery-jtemplates.js"
type="text/javascript"></script>
    <script src="/Shared%20Documents/MortgageAssetsJS/LendingServiceProxy.js"
type="text/javascript"></script>
    <link href="/Shared%20Documents/MortgageAssetsJS/LendingStyles.css"
rel="stylesheet" type="text/css" />
    <title>Lending Institution Rate Selection</title>
  </head>

  <body>
    <div>
      <div id="Institutions">
        <div id="InstitutionSelection">
          <b>Lending Agency:</b>
          <select id='lendinginstdd' style='width:175px'
            onchange='dropdownlending_onchange(this)'></select>
          <br /><br />
          <b>Lending Program:</b>
          <select id='lendingratesdd' style='width:175px'
            onchange='dropdownlendingrates_onchange(this)'></select>
          <br /><br />
        </div>
        <div id="RateDisplay">
          
          <div id="SingleRate">
          </div>
        </div>
      </div>
    </div>
  </body>

  <script type="text/javascript">

    // Script level variables
```

```

var xlWebPart;
var proxy = new LendingServiceProxy();
var APR = null;
var Instrument = null;
var LendingInstitution = null;
var Lock = null;
var Points = null;
var Rate = null;

// Set the page event handlers for onload and unload.
if (window.attachEvent) {
    window.attachEvent("onload", Page_Load);
}
else {
    // For some browsers window.attachEvent does not exist.
    window.addEventListener("DOMContentLoaded", Page_Load, false);
}

// Load the page.
function Page_Load() {
    Ewa.EwaControl.add_applicationReady(GetEwa);
}

// Get handle to EWA instance, set event handlers and initialize content
function GetEwa() {
    xlWebPart = Ewa.EwaControl.getInstances().getItem(0);

    // Retrieve all lending institutions
    $("#RetrievingImg").removeClass("Hidden");
    proxy.GetLendingInstitutions(institutionsRetrieved,
serviceDefaultErrorHandler);
}

// institutionsRetrieved()
// called when the service returns a list of lending institutions
// loads lendinginstdd dropdown list
function institutionsRetrieved(results) {
    if (results) {
        if (results.length > 0) {
            var list = document.getElementById('lendinginstdd');
            // Clear options before loading
            list.selectedIndex = 0;
            list.options.length = 0;
            for (n in results) {
                var opt = document.createElement("option");
                opt.text = results[n];
                opt.value = results[n];
                list.options.add(opt);
            }
        }
        else {
            alert("no lending institutions found");
        }
    }
}

```

```
        $("#RetrievingImg").addClass("Hidden");
    }

    // ratesRetrieved()
    // called when the service returns a list of lending institution rates
    // loads lendingratesdd dropdown list
    function ratesRetrieved(results)
    {
        if (results) {
            if (results.length > 0) {
                var list = document.getElementById('lendingratesdd');
                // Clear options before loading
                list.selectedIndex = 0;
                list.options.length = 0;
                for (n in results) {
                    var opt = document.createElement("option");
                    opt.text = results[n].InstrumentRateDisplayName;
                    opt.value = results[n].ID;
                    list.options.add(opt);
                }
            }
            else {
                alert("no lending rates found for this institution");
            }
        }
        $("#RetrievingImg").addClass("Hidden");
    }

    // loadexcelwebpart()
    // called when the service returns a specific lending institution rate
    // loads the Excel Web Access web part
    function loadexcelwebpart(results) {
        if (results) {
            APR = results.APR;
            Instrument = results.Instrument;
            LendingInstitution = results.LendingInstitution;
            Lock = results.Lock;
            Points = results.Points;
            Rate = results.Rate;
            // load web part named ranges
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!FIR',
setFIRNamedRangeValue);
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!LENDER',
setLENDERNamedRangeValue);
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!INSTRUMENT',
setINSTRUMENTNamedRangeValue);
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!POINTS',
setPOINTSNamedRangeValue);
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!APR',
setAPRNamedRangeValue);
            xlWebPart.getActiveWorkbook().getRangeA1Async('\''Sheet1\'!LOCK',
setLOCKNamedRangeValue);
        }
        else {
```

```

        alert("no specific lending rate found for this institution");
    }
}

// dropdownlending_onchange()
// Event handler for changes with the selection drop down
function dropdownlending_onchange(event) {
    if (event == null || event.selectedIndex < 0)
        return;
    var listIndex = event.options[event.selectedIndex].value;
    // If first item in the list not an actual selection.
    if (listIndex == "null")
        return;
    // Retrieve the lending rate programs offered by lending institution
    var lendingInstitutionName = event.options[event.selectedIndex].text;
    $("#RetrievingImg").removeClass("Hidden");
    proxy.GetALendingInstitutionsRates(lendingInstitutionName, ratesRetrieved,
    serviceDefaultErrorHandler);
}

// dropdownlendingrates_onchange()
// Event handler for changes with the selection drop down
function dropdownlendingrates_onchange(event) {
    if (event == null || event.selectedIndex < 0)
        return;
    var listIndex = event.options[event.selectedIndex].value;
    // If first item in the list not an actual selection.
    if (listIndex == "null")
        return;
    // retrieve the lending rate programs offered by lending institution
    var lendingRateID = event.options[event.selectedIndex].value;
    proxy.GetASpecificLendingInstitutionsRate(lendingRateID, loadexcelwebpart,
    serviceDefaultErrorHandler);
}

// serviceDefaultErrorHandler()
// called when service returns an error
function serviceDefaultErrorHandler(xhr, context, method)
{
    alert(xhr.statusText);
    if (!$("#RetrievingImg").hasClass("Hidden"))
        $("#RetrievingImg").addClass("Hidden");
}

function setFIRNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = Rate * 100;
    range.setValuesAsync(valueArray);
}

function setLENDERNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();

```

```
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = LendingInstitution;
    range.setValuesAsync(valueArray);
}

function setINSTRUMENTNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = Instrument;
    range.setValuesAsync(valueArray);
}

function setPOINTSNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = Points;
    range.setValuesAsync(valueArray);
}

function setAPRNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = APR * 100;
    range.setValuesAsync(valueArray);
}

function setLOCKNamedRangeValue(asyncResult) {
    var range = asyncResult.getReturnValue();
    var valueArray = new Array(1);
    valueArray[0] = new Array(1);
    valueArray[0][0] = Lock;
    range.setValuesAsync(valueArray);
}

</script>
</html>
```

code snippet 076576 Ch08_Code.zip/LendingRates.html

The major difference between this code snippet and the test harness you previously worked with is that this code includes the use of the Excel ECMAScript object model for manipulating the Excel workbook in the web part.

First, you need to initialize your connection to the Excel Web Access web part. This is an established pattern. You add an `.add_applicationReady()` event listener so that when the Excel Web Access web part loads and is ready on the page, a callback to `GetEwa` is made. In the `GetEwa` callback method, you can get a reference to a specific instance of an Excel Web Access web part on the

page. In your code, you then refer to the object, in this case `xlWebPart`, to interact with that specific web part.

For your purposes, you only want to interact with the web part once the end user selects a specific lending rate from the drop-down. At that time, you need to push each of the appropriate data values into the grid so that the mortgage model can recalculate accordingly, and display the other data values for end user reference. To do this, you use the `.getRangeAsync()` method on the active workbook and pass in the name of the sheet to access, the named range value, and a callback method.

For example, when you want to set the cell value for the FIR (fixed interest rate) named range, you pass in the sheet name (`sheet1`), the named range (`FIR`), and then a callback (`setFIRNamedRangeValue`). When the cell has been selected in the grid, the callback method is called. The rate data value is placed in the array structure that Excel expects, and the `range.setValuesAsync()` method is called, passing in the array. The value is updated in the web part, and the sheet is recalculated automatically. The pattern for getting and setting the values for the additional named range cells is the same.

With your code complete, now you need to get it into the Content Editor web part you added to your SharePoint web page previously. However, first you need to get the appropriate files you used locally uploaded to a SharePoint document library. This could be the same one where your Excel document is located.

1. On your SharePoint site, navigate to the document library of your choice and upload the `LendingRates.html`, `jquery.js`, `jquery-jtemplates.js`, `LendingServiceProxy.js`, and `LendingStyles.css` files.
2. Once the files are uploaded, open the `LendingRates.html` file and modify the `<script src>` tag with the path to your JavaScript files, and the `<link href>` tag with the path to the CSS file, and then save the `LendingRates.html` file back to the document library.
3. Right-click the `LendingRates.html` file and choose Copy Shortcut.
4. Navigate to your MortgageModel site page.
5. Click on the Page tab and select Edit on the ribbon. Hover over the Content Editor web part title bar, click the drop-down arrow on the right, and then select Edit Web Part.
6. In the Content Link text box, paste the URL you copied in step 3 to the `LendingRates.html` file into the text box, and then choose OK.
7. Choose Save & Close from the SharePoint ribbon.

You should see the first drop-down immediately load with the list of lending institutions. Using the drop-downs on the left, select a lending agency and a lending program. Your financial model should update with a result similar to what is shown in Figure 8-18.

You have come full circle. What was only a client-based solution has now also become a web-based solution that many end users can use, even if they don't have Excel on their desktop. It's an excellent strategy to start thinking about how you can better utilize Excel as a service in order to leverage the business logic in Excel documents in web-based solutions.

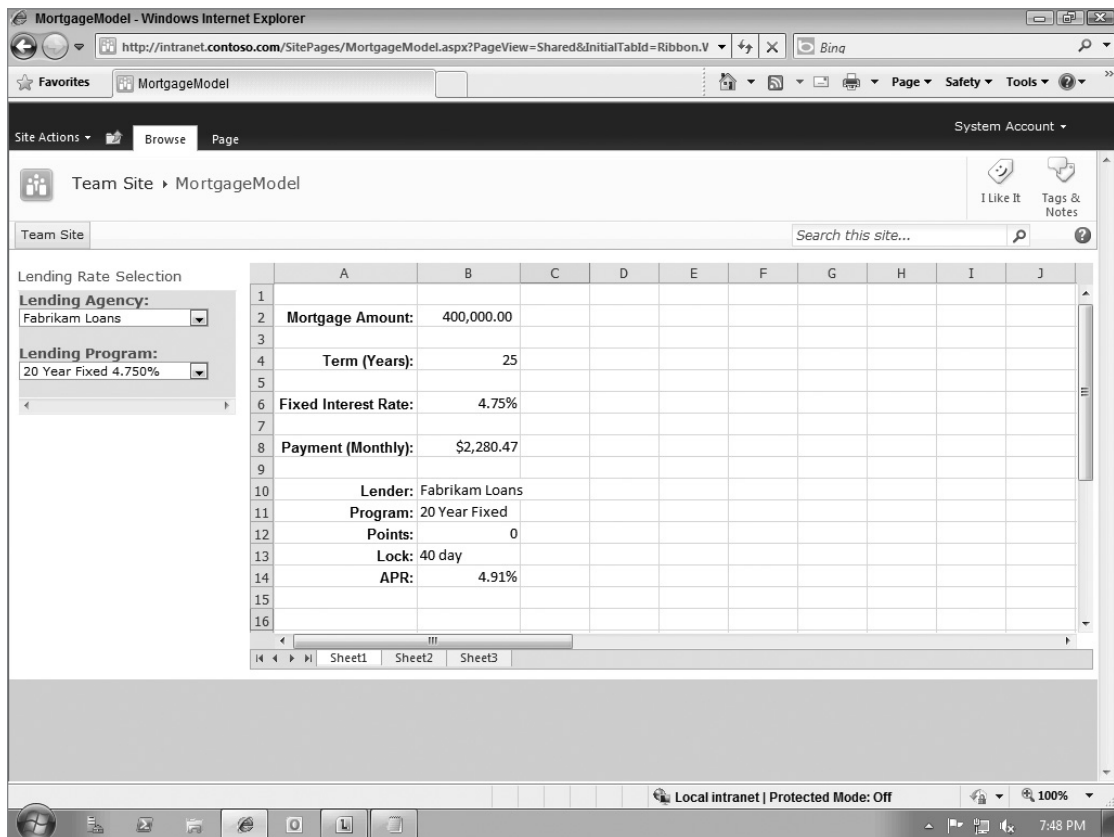


FIGURE 8-18

SUMMARY

There are millions of Office users worldwide, and all the tools are in place for developers to deliver the next generation of cloud-based solutions on Office and SharePoint. In this chapter, you used an Office 2010 project template in Visual Studio 2010 to create an Excel document-level add-in that brought Windows Azure Table data directly into the financial model. Then you learned how to build a JSON-enabled WCF service and how to use the Excel ECMAScript object model in a Content Editor web part to provide end users with the same functionality when the model is uploaded to Excel Services in SharePoint 2010. If you have not previously developed for Office, consider incorporating the Office clients into your next SharePoint solution.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- **Office Developer Center on MSDN** — <http://msdn.microsoft.com/office>
- **Windows Azure Table Service API** — <http://msdn.microsoft.com/en-us/library/dd179423.aspx>
- **Microsoft Excel Team Blog** — <http://blogs.office.com/b/microsoft-excel/>
- **Office 2010 Developer Training Course on MSDN** — <http://msdn.microsoft.com/en-us/Office2010DeveloperTrainingCourse.aspx>
- **Office 2010 Developer Training Kit Download** — <http://go.microsoft.com/?linkid=9702452>



Creating a Training Application in SharePoint Online

WHAT'S IN THIS CHAPTER?

- Building a simple video-based training application in the cloud
- Using the Windows Azure Blob storage service to complement SharePoint Online's storage
- Using JavaScript and jQuery to make AJAX calls to SharePoint web services

We live in an era in which online videos represent the biggest percentage of traffic across the entire Internet. Cisco predicts that videos will account for more than 50% of Internet traffic by the end of 2012. This is due in part not only because of the number of videos, but also the quality of the videos, as there has been a recent explosion of high definition (HD) video content online.

Fueling the fire is the “consumerization of IT” — that is, people expect the convenient technology they use at home to be available at the workplace, too, and vice versa. Watching YouTube videos and live HD streaming of sporting events using Silverlight have become de facto video standards established by users.

This trend has provided businesses, which are always looking for ways to cut costs, with an opportunity. Organizations often slash training costs before other costs, as in-person training programs are expensive, and often require travel and time away from office. However, information workers still need the new knowledge they gain from these programs and seminars to improve their skills and productivity in a rapidly evolving workplace. Moreover, many companies have mandatory training programs, such as safety and business conduct training, that are essential for new employees who need to be integrated into the organization.

This combination of cost effectiveness and user popularity has led many organizations to adopt the use of online, video-based training programs for employees. This enables the attendees to learn at their own pace and without the hassle of travel, while providing a significant cost savings to the company. In this chapter, you'll build a simple corporate training application that leverages Windows Azure for scalable video storage and SharePoint Online's platform capabilities for faster application development. The application will provide social collaborative capabilities and an easy-to-use experience for corporate users.

OVERVIEW OF THE TRAINING APPLICATION

In this chapter, you will be building a simple application in SharePoint Online that holds a list of online training videos and provides a nice user experience for playing the videos. It will also use the social capabilities of SharePoint for user interaction. The application will be built so that it runs entirely in the cloud using SharePoint Online and Windows Azure. This enables end users to go through the training videos whenever they choose and have access to the Internet.

Because the objective here is to demonstrate how you can connect the technical building blocks, the application will have an extremely minimal set of features. Of course, it will also be extensible, so you can use your ingenuity to turn it into a full-scale training solution.



Although this chapter focuses on SharePoint Online, the application can be developed and deployed for on-premises SharePoint environments and hybrid deployments.

THE SOLUTION ARCHITECTURE

The solution you'll be building is a training library in SharePoint Online using custom lists that are available in SharePoint. You will further customize the list to include videos. The custom list will also use the "I Like It" and "Tags & Notes" features of SharePoint to hold social information, and you'll display the social tags within the custom list using JavaScript and jQuery. The videos will be encoded to adaptive streaming format and stored in the Windows Azure Blob storage service for maximum efficiency. The end result looks like the view shown in Figure 9-1. The custom list can be converted to a template so that you can provision additional training libraries with just a few clicks.

Understanding the SharePoint Online Platform

The version of SharePoint Online we'll be using in this chapter is shipped as part of Office 365 and is built on the SharePoint 2010 platform. Office 365 is a subscription-based service that includes SharePoint Online, Exchange Online, Office Web Apps, Lync Online, and Office Professional Plus. For more information about Office365, visit www.office365.com.

In simple terms, SharePoint Online is SharePoint 2010 running in the cloud, hosted and maintained by Microsoft for you. It is a multi-tenant hosted instance of SharePoint with the same set of basic user functionality. Because it is in a multi-tenant environment, developers don't have access to the physical box. Instead, they build for the SharePoint sandbox and other client-side artifacts, enabling you to work securely in your SharePoint Online instance without jeopardizing other tenants that are hosted on your server. This is also a best practice for developing your SharePoint 2010 on-premises solutions so that you can easily migrate them from on-premises SharePoint to SharePoint Online and vice versa.

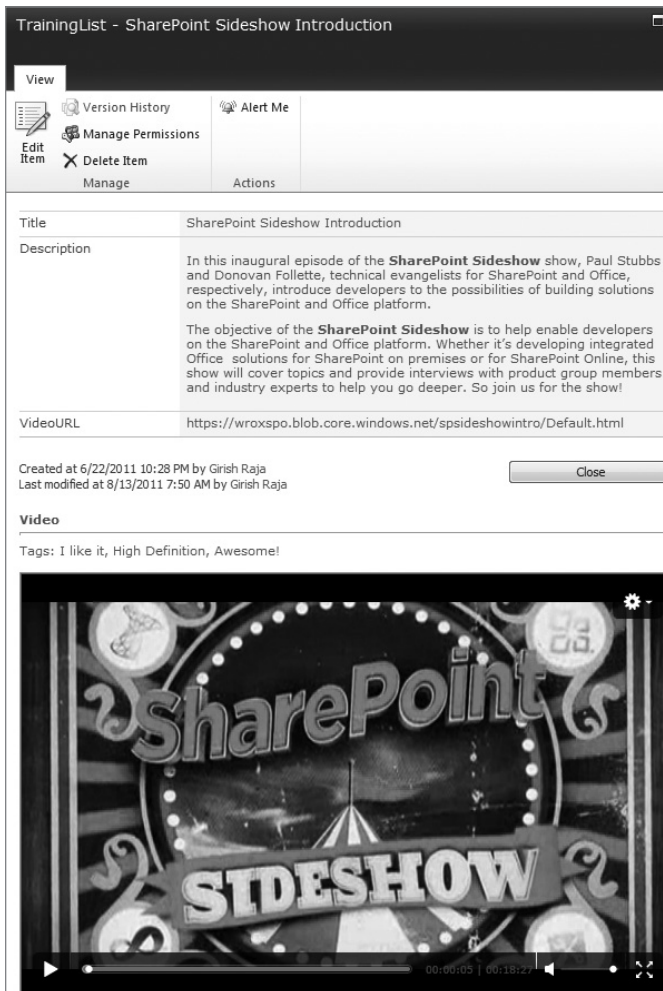


FIGURE 9-1

Scalable Storage in the Cloud with Azure

With Software as a Service (SaaS) products like Office 365, you often get finite and pre-allocated storage that you can use for your files. Of course, adding additional storage is possible, but the increments are often at a premium cost and are not scalable both in terms of cost and storage efficiency.

In our training scenario, video files can take several gigabytes (sometimes terabytes) of storage, making storing the videos directly in SharePoint very expensive and inefficient. One alternative is to store videos in a Windows Azure storage service — specifically, the Blob service, which is well suited for text and binary data.

With the Windows Azure storage services, you get built-in fault tolerance because all your content is replicated three times. Furthermore, there is no limit to the amount of storage you can use, so Windows Azure truly provides infinite elasticity for storage. You pay for only what you use. With Azure storage, you can optionally add additional benefits like content distribution networks (CDNs), which distribute your files geographically so that they are closer to a user's location across the globe, thereby dramatically boosting performance.

BUILDING THE VIDEO LIBRARY

In this section, you will build various components of the training application. First, you'll create the list definition specifying various columns to store the training data. Next, you'll convert the training videos to smooth streaming format and upload them to Windows Azure Blob storage. You'll then create a video player within the list using JavaScript. Finally, you'll embed social data from SharePoint into the video player using jQuery.

To complete the activities in this chapter, you need the following tools and online accounts:

- **SharePoint Online Account from Office 365** — You can sign up at www.office365.com.
If you don't have a SharePoint Online account, you can instead use an on-premises SharePoint installation, such as the Information Worker Virtual Machine, at www.microsoft.com/downloads/en/details.aspx?FamilyID=751fa0d1-356c-4002-9c60-d539896c66ce.
- **An active Windows Azure subscription** — www.microsoft.com/windowsazure/offers/
- **Expression Encoder 4 Pro (optional)** — www.microsoft.com/expression/products/EncoderPro_Overview.aspx
- **Adaptive Streaming with Windows Azure Blobs Uploader** — <http://archive.msdn.microsoft.com/streamingazure>
- **jQuery 1.6.1** — http://docs.jquery.com/Downloading_jQuery
- **spjs-utility.js** — <http://sharepointjavascript.wordpress.com/2010/05/28/get-or-set-value-for-sharepoint-field-in-newform-editform-and-dispform-get-only-in-dispform>

- **SPServices 0.6.1** — <http://spservices.codeplex.com>

SPServices is a jQuery library that abstracts SharePoint's web services and makes them easier to use within JavaScript and jQuery.

- **Sample videos** — You can use your own sample videos or download creative commons videos available for free at www.bigbuckbunny.org/index.php/download or <http://channel9.msdn.com>.

Creating the Data Model in SharePoint Online

The training application will be a self-contained custom list within SharePoint. Therefore, you need to first create the data model within SharePoint to store the metadata for videos:

1. Create a new Site Collection within SharePoint Online and browse to its home page.
2. Click List in the left navigation bar, and then click the Create button.
3. Choose Custom List in the dialog that appears and enter **TrainingList** for the name (see Figure 9-2). Click the Create button, which will create the list and redirect the page to the default view.

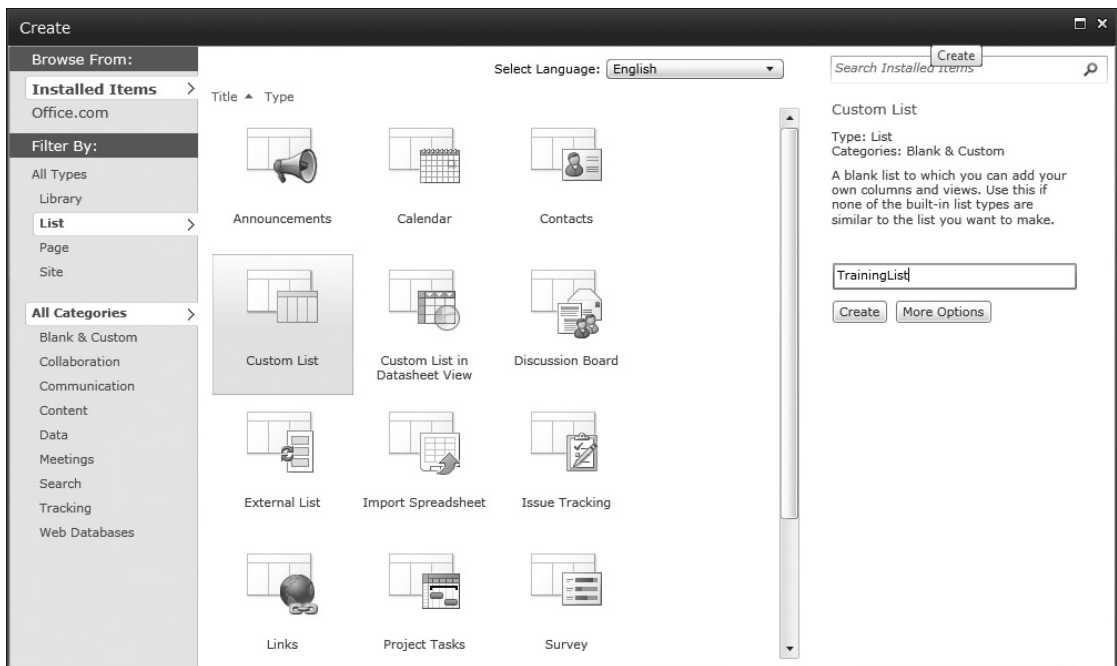


FIGURE 9-2

4. Under the List ribbon menu, click List Settings.

5. In the List Settings page, create two new columns for the list, Description and VideoURL, with data types shown in Figure 9-3.

By default, SharePoint includes an Attachment column, which can be used for storing relevant files, such as student handbooks or reference materials.

Columns		
A column stores information about each item in the list. The following		
Column (click to edit)	Type	Required
Title	Single line of text	✓
Description	Multiple lines of text	
VideoURL	Single line of text	
Created By	Person or Group	
Modified By	Person or Group	
Create column		

FIGURE 9-3

Although the list is pretty simple at this point, you can add additional fields based on your metadata requirements. The goal here is to have self-contained metadata for the list of videos. Also, instead of creating new columns, you could add them from existing site columns, if you have the appropriate types already created in your site column gallery.

Uploading Videos to Windows Azure

This section describes the process of uploading a video to the Windows Azure Blob service. As mentioned earlier, the videos will be encoded to adaptive/smooth streaming format before being uploaded to Azure storage.

1. Download the Adaptive Streaming with Windows Azure Blobs Uploader from <http://archive.msdn.microsoft.com/streamingazure> and extract the binaries.
2. Within the EncoderPlugin subfolder, open an elevated command prompt and enter **install.cmd** to install the plugin for Microsoft Expression Encoder.
3. Open Expression Encoder and choose Silverlight Project in the initial dialog.
4. On the main menu, click File ⇨ Import, and then choose the video file you would like to upload to Windows Azure. The video will appear in the Source pane at the bottom of Expression Encoder.
5. Right-click the video and select Apply Preset ⇨ Encoding for Silverlight ⇨ IIS Smooth Streaming ⇨ VC-1 IIS Smooth Streaming - HD 720p VBR, as shown in Figure 9-4.

In this case, the resolution of the source video is 1280×720 , so the encoding profile is high-definition, 720-pixel-wide variable bitrate resolution (VBR) for optimum clarity and file size. You can choose other smooth streaming encoding profiles based on your input video.

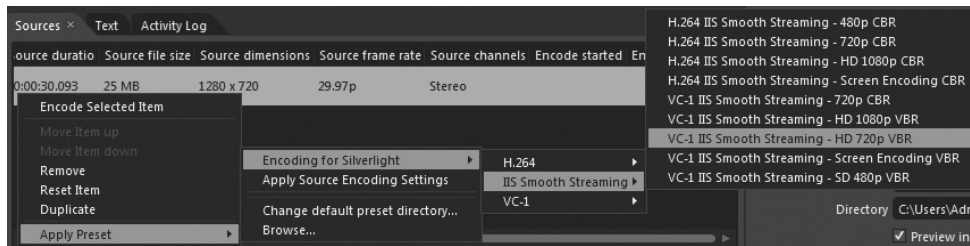


FIGURE 9-4

Expression Encoder also generates a nice Silverlight video player for the video. If you look in the right panel of Encoder, you'll see a section titled **Templates** where you can choose from several templates that are available for the video player, as shown in Figure 9-5. For this example, we'll go with the default Expression template.

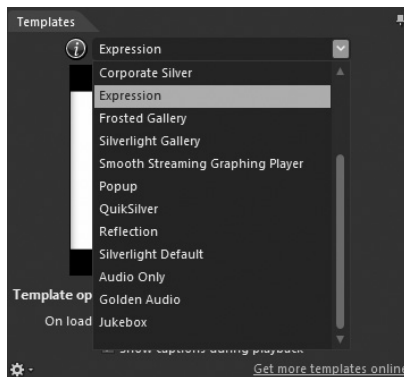


FIGURE 9-5



The screenshots used in this chapter were taken using Microsoft Expression Encoder 4 Pro. If you have a different version of Expression Encoder, the screen layout may differ slightly.



Smooth streaming is a technology that dynamically detects local bandwidth and CPU conditions, and seamlessly switches, in near real time, the video quality of a media file that a player receives. Consumers with high-bandwidth connections can experience high definition (HD) quality streaming, whereas users with lower bandwidth speeds receive the appropriate stream for their connectivity, enabling consumers across the board to enjoy a compelling, uninterrupted streaming experience and obviating the need to cater to the lowest common denominator quality level within the audience base.

(continues)

(Continued)

During the process of encoding, the video/audio source is cut into many short segments (chunks) and encoded to the desired delivery format. The “adaptive” part of the solution comes into play when the video/audio source is encoded at multiple bit rates, generating multiple chunks of various sizes for each 2 to 4 seconds of video. Because web servers usually deliver data as fast as network bandwidth allows, the client can easily estimate user bandwidth and decide to download larger or smaller chunks ahead of time. Also, if a user fast forwards or skips during the video playback, only the corresponding chunk is downloaded, so the user sees the video with almost no buffering.

For a deeper technical overview of smooth streaming, read the white paper at www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=03d22583-3ed6-44da-8464-b1b4b5ca7520.

6. Within the Publish window in the right panel, select Publish to Windows Azure Adaptive Streaming Publisher, which is enabled because of the add-in installed earlier.
7. Check the “Publish after encode” checkbox.
8. In the Settings pane, enter your Windows Azure storage service account name, account access key, and a unique path where the videos will be published (see Figure 9-6).

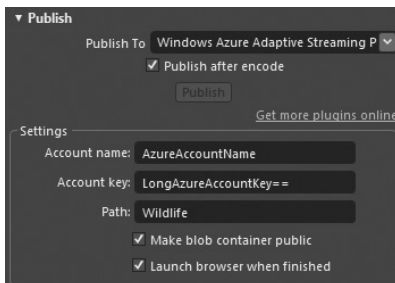


FIGURE 9-6



For illustrative purposes, our blob container is set to public access, meaning anyone with the resulting URL can access the video across the Internet. Blobs can also be stored in private containers, which can then be made available to a restricted audience using a shared authentication key. If your training materials include sensitive or confidential information, you might want to store the videos in private containers.

Also, ensure that you enter a valid unique path within Expression Encoder. Paths are also referred to as “containers” in the Windows Azure Blob storage nomenclature. For a list of valid container names, see <http://msdn.microsoft.com/en-us/library/dd135715.aspx>.

9. If you're unsure of your Windows Azure storage account credentials, log in to the Windows Azure Platform Management Portal at <http://windows.azure.com> using your subscription. Under the Storage Accounts section, either create a new storage account or select an existing storage account to use. The name of the account will be displayed in the Properties pane. Click View Access Keys in the ribbon to see the Primary access key, which is usually a long string of random characters. You can click on the Copy to Clipboard button, as shown in Figure 9-7, to copy the key and paste it within the Account key textbox in Expression Encoder.

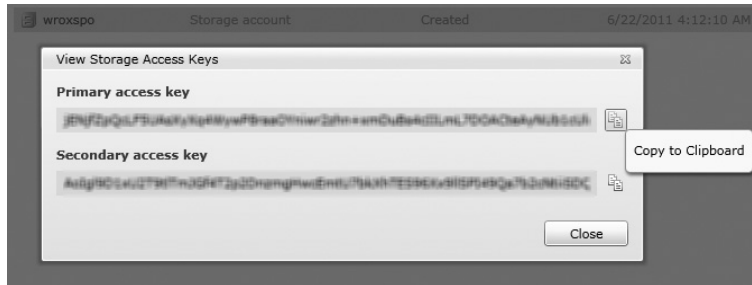


FIGURE 9-7

10. At this point, you can save the encoding job, if you like, by selecting File ⇨ Save Job in the main menu within Expression Encoder.
11. The encoding job is now ready. Click the Encode button in the Sources pane to begin encoding (see Figure 9-8).

The encoding process takes a few minutes to complete, varying according to the size of the video and the performance of the machine on which you're running the encoder.

Once the encoding process is complete, a browser window will open, playing the video.

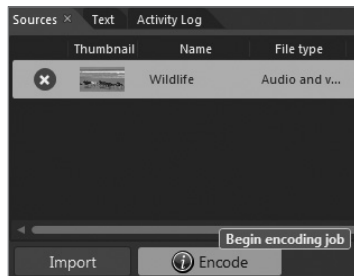


FIGURE 9-8

12. The next step is to get the URL of the video player stored in Windows Azure, which is an easy task. For example, if your storage account name were “wroxspo” and the path you chose were “wildlife” (refer to Figure 9-6), the video URL would be <https://wroxspo.blob.core.windows.net/wildlife/Default.html>.

You can also get the URL of the video player by checking your Azure storage account using tools like the Azure Storage Explorer (<http://azurestorageexplorer.codeplex.com>) if you want to obtain the URL from your storage account later (see Figure 9-9).

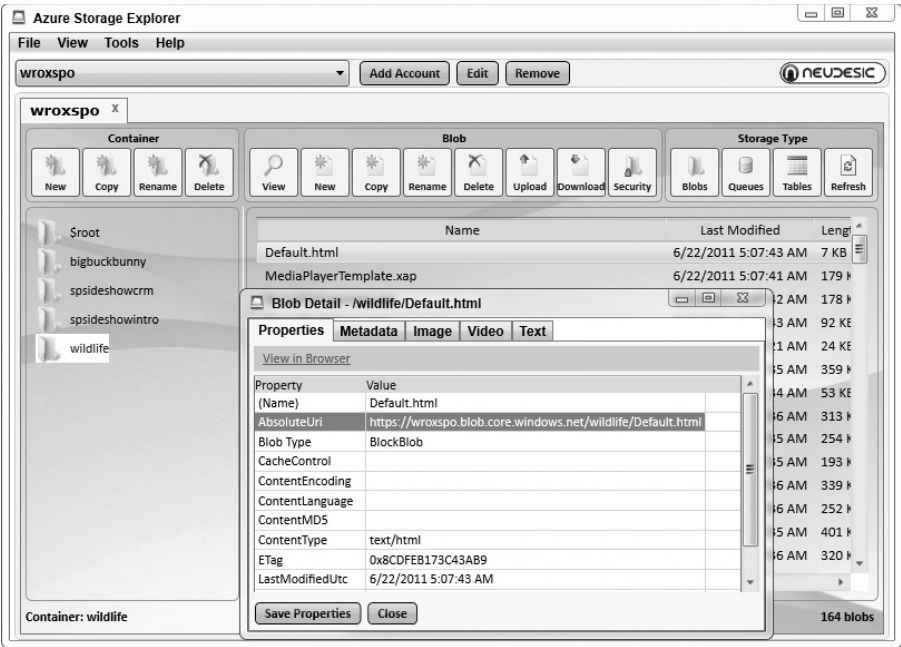


FIGURE 9-9

13. Repeat steps 3 through 12 for each video you'd like to encode and upload to Windows Azure Blob service.

If you don't want to use Expression Encoder, you could alternatively perform the preceding steps using AdaptiveStreamingAzure.exe, from <http://archive.msdn.microsoft.com/streamingazure>, to get the video uploaded to Windows Azure storage.

These days, it is also pretty common to have live streaming videos for training instead of on-demand content. If that is the case, you could try using Windows Azure Live Smooth Streaming Project (<http://azlivestreaming.codeplex.com>) and follow basically the same approach to obtain the live streaming URL and proceed with the rest of the chapter.

CREATING AN INTEGRATED USER EXPERIENCE

In this section, you'll customize the custom training list in SharePoint Online to include the videos from Azure. You will create a simplistic UI so that a training attendee can easily view a list of available videos and quickly select and play one. You'll also use the built-in social features of SharePoint Online and its API to pull relevant information. The interesting approach here is the use of JavaScript, jQuery, and AJAX to render information on the SharePoint Online page. This pattern of client-side

HTML and JavaScript development is now very popular, and its use will become even more critical as adoption of HTML5 grows within the developer community.

Creating the Video Player

Use the following steps to populate the custom training list with the videos you have uploaded to Azure:

1. Return to the SharePoint Online page, to the custom TrainingList that you created and configured earlier.
2. Click Add New Item to add a new video to the list.
3. Enter the title, a brief description, and the URL of the video you uploaded to the Blob service (from the previous section), as shown in Figure 9-10. You can also add attachments to the list item — for example, to include handouts or other reference manuals as part of the video listing. Once you have added everything you need for the video item, click the Save button.

The screenshot shows a SharePoint form titled "TrainingList - Wildlife in HD". The form has a toolbar with icons for Save, Cancel, Paste, Copy, Delete Item, Attach File, and Spelling. Below the toolbar, there are several fields: "Title *" with the value "Wildlife in HD", "Description" with a text area containing "The official wildlife video that is featured in Windows 7. The video is in HD and is about 15 seconds in length, this is a perfect sample video if you want to show off your HD computer or TV. Footage: Small World Productions, Inc; Tourism New Zealand | Producer: Gary F. Spradling | Music: Steve Ball", "VideoURL" with the value "https://wroxspo.blob.core.windows.net/wildlife/Default.html", and "Attachments" with two files: "Wildlife Q and A.docx" and "Wildlife Handout.pdf". At the bottom of the form, there are "Save" and "Cancel" buttons.

FIGURE 9-10

4. Repeat the preceding two steps for as many videos as you'd like to include as part of the training list.
5. At this point, you need some supporting JavaScript libraries for this list. Create a new document library within the site called **JavaScript** and upload the following three JavaScript files within the document library:
 - **jQuery 1.6.1** — http://docs.jquery.com/Downloading_jQuery
 - **spjs-utility.js** — <http://sharepointjavascript.wordpress.com/2010/05/28/get-or-set-value-for-sharepoint-field-in-newform-editform-and-dispform-get-only-in-dispform>
 - **SPServices 0.6.1** — <http://spservices.codeplex.com>

The resulting document library should look like Figure 9-11.

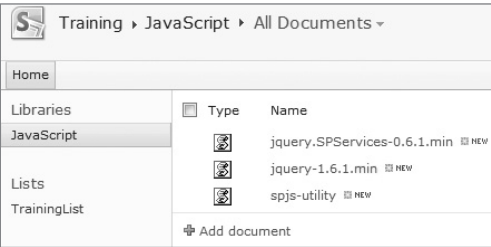


FIGURE 9-11

6. After adding the JavaScript files, return to the TrainingList and click the List Tools ribbon menu. Under the List category, there is a Customize List section. Within that section, click the Form Web Parts drop-down and select Default Display Form, as shown in Figure 9-12.



FIGURE 9-12

7. When the page is in edit mode, click Add a Web Part in the middle of the page.
8. Add an HTML Form web part and move it below the TrainingList web part by dragging and dropping it underneath it.
9. Click the Web Part Menu and select Edit Web Part, as shown in Figure 9-13.

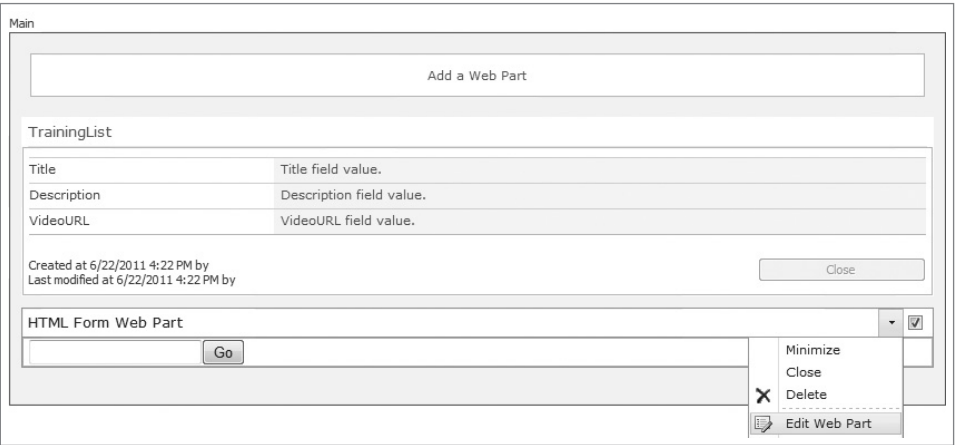


FIGURE 9-13

10. Click the Source Editor Button to bring up the Text Editor dialog.
11. Delete the text within the Text Editor and enter the following code:

```
<B>Video</B> <HR />
<BR>
<iframe id="videoFrame" width="100%" height="384px"> </iframe>
```



```

<script type="text/javascript"
    src="../../JavaScript/jquery-1.6.1.min.js"></script>
<script type="text/javascript"
    src="../../JavaScript/spjs-utility.js"></script>
<script type="text/javascript">
fields = init_fields_v2();
var currentVideoURL = getFieldValue('VideoURL',true);
document.getElementById('videoFrame').src = currentVideoURL;
</script>

```

code snippet 076576 Ch09_Code.zip/TrainingListIFrameScript.js

The preceding code snippet first creates an iframe window within the web part. You then use `spjs-utility.js` to obtain the value of the VideoURL field in the display form. Then you take that value and set it as the source of the inline frame (`<iframe>` tag).

This enables you to take the video from Windows Azure Storage and play the video within the Default Display Form. Although you could build a custom Silverlight-based player, this example leverages the existing player generated by Expression Encoder.

12. Click the Save button, followed by OK in the Web Part Properties pane.
13. Under the Page ribbon, click Stop Editing to save your changes to the form.
14. Open one of the items in the TrainingList. You should see the video displayed below the form in an iframe (see Figure 9-14). You can play the video by clicking the play button in the middle, and you can pause, skip forward, and play the video in full screen, if you like.

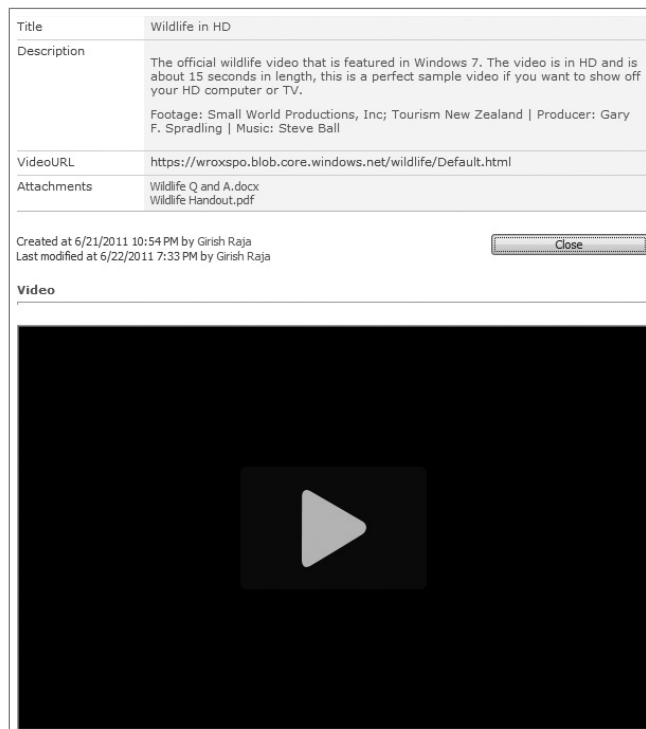


FIGURE 9-14

The beauty of this setup is that from an end user's perspective, the video appears to be playing natively from SharePoint Online, whereas behind the scenes it is actually playing from Windows Azure Blob storage service.



You can also include videos from websites such as YouTube. To include YouTube videos, ensure that you include the appropriate “embed” URL so that the video plays within the iframe. Also, your web browser might display a warning if your video is being rendered from a HTTP site. This is because SharePoint Online uses the protected HTTPS mode by default; so if any of the page content (such as iframe content) is rendered in unsecured HTTP mode, the browser warns users so that they are not duped by unsecured content. To be safe, it is better to use HTTPS URLs. Therefore, the YouTube link you want to use might look like `https://www.youtube.com/embed/s12Jb5Z2xaE`, which looks like Figure 9-15.

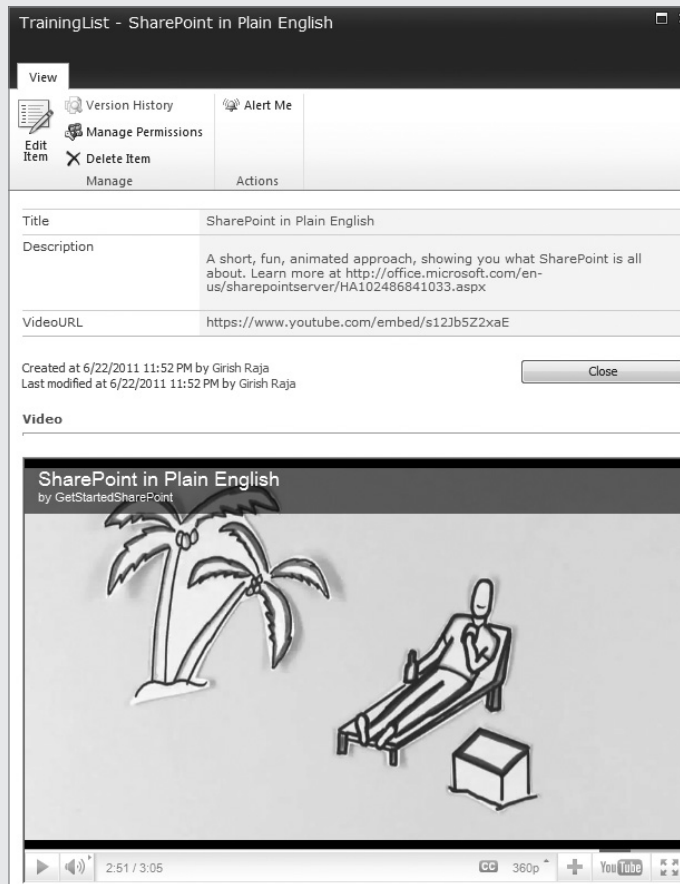


FIGURE 9-15

Enabling Social Experiences in Training

Training is rarely an isolated experience for an individual. Historically, it is clear that collective intelligence is certainly beneficial for the individuals involved, and therefore very productive for the organization itself. When the people involved are not physically located within the same room or setting, however, it can be difficult for them to socialize and collaborate on learning activities. Fortunately, these days, an increasing number of people are getting used to remotely socializing and collaborating with their friends and family through online tools. With the advent of Facebook and Twitter, this has been greatly accelerated.

SharePoint Online brings the concept of personal social networking to the enterprise. With the revamped My Site, profile pages, I Like It button, and a host of other new social features, SharePoint Online has become *the* social collaboration platform for businesses. SharePoint exposes these social features using the Social Data Service. This web service is easy to consume in .NET applications; however, it would be difficult to use within jQuery. This is where the `SPServices` library comes into play, as it is a wrapper around SharePoint web services (including the Social Data Service), providing a simpler way to use the SharePoint web services from jQuery. You can read more about the Social Data Service at <http://msdn.microsoft.com/en-us/library/ee590739.aspx>.

In this section, you'll use the built-in I Like It button and the Tags & Notes feature of SharePoint Online to enable users to post and share comments about the videos in the TrainingList. You'll then use the Social Data Service to retrieve the data and display it along with the video using JavaScript and jQuery.

1. Open the custom TrainingList that you created and configured earlier.
2. Select an item in the list and click the I Like It button on the ribbon, as shown in Figure 9-16.

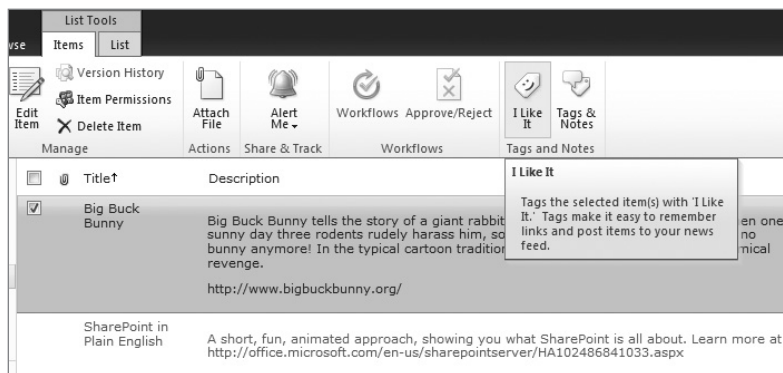


FIGURE 9-16

3. Click the Tags & Notes button and enter a few appropriate tags for the video, separated by semicolons (see Figure 9-17). You can also add notes about the video under the Note Board. Once you have entered all the tags you need, click the Save button.
4. Back in TrainingList, click the List Tools ribbon menu. Under the List category ⇨ Customize List section, click the Form Web Parts drop-down, and select Default Display Form (refer to Figure 9-12).

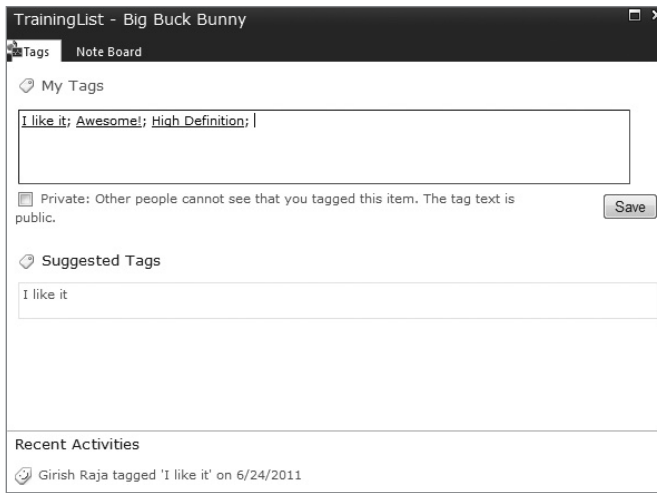


FIGURE 9-17



Given that you have already added some custom script to the page, you might see an error stating “Server Error in ‘/’ Application” within the web part in customization mode. You can safely ignore that error.

5. Open the Web Part menu for the HTML Form web part and select Edit Web Part (refer to Figure 9-13).
6. Click the Source Editor button to bring up the Text Editor dialog. Delete the text within the Text Editor and enter the following code:

```
<B>Video</B> <HR />
<BR>
<iframe id="videoFrame" width="100%" height="384px"> </iframe>
<script type="text/javascript"
    src="../../JavaScript/jquery-1.6.1.min.js"></script>
<script type="text/javascript"
    src="../../JavaScript/spjs-utility.js"></script>
<script type="text/javascript"
    src="../../JavaScript/jquery.SPServices-0.6.1.min.js"></script>
<script type="text/javascript">
fields = init_fields_v2();
var currentVideoURL = getFieldValue('VideoURL',true);
document.getElementById('videoFrame').src = currentVideoURL;
$(document).ready(function() {
    $.SPServices({
        operation: "GetTags",
        // Need URL like https://myinstance.sharepoint.com/TrainingTest
        //                               /Lists/TrainingList/DispForm.aspx?ID=2
```



Available for
download on
Wrox.com

```

url: window.location.href.substring(0,window.location.href.indexOf('&')),
dataType: "xml",
    completefunc: function (xData, Status) {
var textToInsert='';
    $(xData.responseXML).find("Name").each(function() {
        textToInsert = textToInsert + $(this).text() + ', ' ;
    });
    $('#videoTags').text('Tags: ' +
        textToInsert.substring(0, textToInsert.length-2));
    });
});
</script>

```

code snippet 076576 Ch09_Code.zip/TrainingListSocialScript.js

In the preceding code snippet, you are extending the code you wrote earlier by including the `SPServices` library. You then use jQuery to make an AJAX web call to the Social Data Service to get the tags for the current open video. The resulting XML is then parsed, and the tags are displayed above the video.

7. Click the Save button, followed by OK, in the web part's Properties pane.
8. Under the Page ribbon, click Stop Editing to save your changes to the form.
9. Open the item in the TrainingList. You should see the tags displayed right above the video (see Figure 9-18).

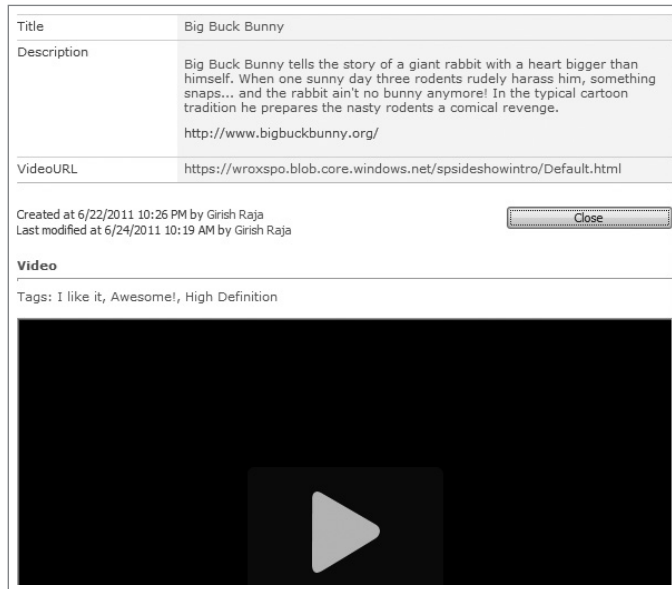


FIGURE 9-18

This exercise is just the tip of the iceberg in terms of what you can do with the Social Data service in SharePoint. The main objective of this section was to get you started using the Social Data service. For more methods available through this API, check out <http://spservices.codeplex.com/wikipage?title=SocialDataService>.

Congratulations! You have now created a nice end-to-end training application in SharePoint Online.

SUMMARY

SharePoint Online, when combined with Windows Azure, provides an amazing platform for developing applications in the cloud. In this chapter, you created a customized training list library in SharePoint Online using simple list customizations and client-side development only. A key takeaway here for developers is that you can create a variety of artifacts and customizations in SharePoint Online using just HTML, JavaScript, and jQuery, and making AJAX web service calls. The Windows Azure Blob storage service complements such development patterns by providing scalable and limitless storage, along with easy-to-use tools and APIs for surfacing the blobs within SharePoint. You can use the Blob service for storing any binary data from SharePoint, including the video example you used in this chapter. Using these code samples as a template and your ingenuity, we are sure you will come up with more interesting social training solutions for SharePoint Online.

ADDITIONAL REFERENCES

- *Beginning SharePoint 2010 Development* — Steve Fox (Wrox, 2010)
- **Steve Marx's blog article** — <http://blog.smarx.com/posts/smooth-streaming-with-windows-azure-blobs-and-cdn>
- **spjs-utility.js** — <http://sharepointjavascript.wordpress.com/2010/05/28/get-or-set-value-for-sharepoint-field-in-newform-editform-and-dispform-get-only-in-dispform>
- **SPServices jQuery Library for SharePoint Web Services** — <http://spservices.codeplex.com/wikipage?title=SocialDataService>
- **jQuery Project** — <http://jquery.org>
- **Girish Raja's blog** — <http://blogs.msdn.com/girishr>

10

Managing Customer Information in SharePoint Online

WHAT'S IN THIS CHAPTER?

- Building a SharePoint Online dashboard that brings various customer-related information from Microsoft Dynamics CRM Online
- Understanding CRM Online and how to interact with its web services
- Using Silverlight and Windows Azure to bridge SharePoint Online and CRM Online

Customers are the key to any successful business. In our current global economy, gaining new customers, while at the same time keeping existing customers happy, is more important than ever. As a result, customer relationship management (CRM) systems have evolved rapidly in the last decade. What were once nice-to-have tools are now must-have tools for businesses worldwide. If you are new to CRM, it is essentially an application to manage the sales pipeline, marketing activities, and customer service requests of a business.

Traditionally, businesspeople have maintained their customer information in a simple list within Microsoft Excel, but their needs quickly outgrow such simple lists. Software as a service-based (SaaS) CRM systems are already one of the major applications in the cloud computing space, and many analysts forecast that the number of customers using such online CRM systems will grow significantly in the next few years.

Although most CRM systems are capable of handling all business functions (such as ordering, billing, and inventory), many organizations still use multiple line-of-business (LOB) applications to manage such functions. Integrating data from these disparate LOB applications and creating a unified view of the information is critical if business users are to analyze and collaborate on the information it provides. This is where SharePoint comes into the picture, as it is an indispensable platform for building such integrated solutions.

In this chapter, you will build a simple SharePoint Online application that provides a unified view of data from CRM Online. The SharePoint application can then be extended to include data from other LOB applications.

OVERVIEW OF MICROSOFT DYNAMICS CRM

Microsoft Dynamics CRM is an application built by Microsoft as part of the Microsoft Dynamics family of business applications. Dynamics CRM offers a full suite of functionality in the sales, marketing, and customer service areas. Similar to SharePoint, Dynamics CRM is available in three deployment models: traditional on-premises software, partner-hosted systems, and the Microsoft-hosted Microsoft Dynamics CRM Online (aka CRM Online). This chapter focuses only on CRM Online, especially the R6 update of CRM Online released in January 2011, and currently available in 40 markets and 41 languages worldwide.



Although this chapter covers only CRM Online, you should be able to use the CRM code from this chapter in other deployment models, as well. All three types of CRM deployment have the same underlying code base, and the developer extensibility options are similar across all three.

From a developer's perspective, CRM Online is very similar to SharePoint in that it is a web application built on .NET, has industry-standard web service endpoints, and uses SQL Server as the underlying storage system, with a variety of middle-tier components, including plug-ins, workflows, and so on. The CRM UI uses components from SharePoint and Office, such as the navigation pane and ribbon, as shown in Figure 10-1.

Figure 10-2 shows an architectural overview of the CRM platform and the various components contained within it. The gear symbols represent customization points for developers. For more information on the CRM architecture, check out the CRM developer center, at <http://msdn.microsoft.com/dynamics/crm>.



Developers have built a variety of LOB applications on top of the CRM platform. To learn more about CRM and how developers can extend it, check out the whitepaper titled "Building Business Applications with Microsoft Dynamics CRM 2011," available at www.microsoft.com/downloads/en/details.aspx?FamilyID=42b24fb7-b0c7-408a-ae65-90884616285f.

CRM Online is built on .NET Framework 4 and offers a couple of WCF-based web services to interact with the data. This enables developers to use a variety of techniques to interact with CRM's web services, as shown in Figure 10-3. In this chapter, we'll be interacting with CRM Online web services from a Windows Azure proxy using late-bound connectivity to the SOAP endpoint with `Microsoft.Xrm.Sdk` assemblies.

As mentioned earlier, the API model is the same for both on-premises software and CRM Online. The only difference is how you authenticate to the web services.

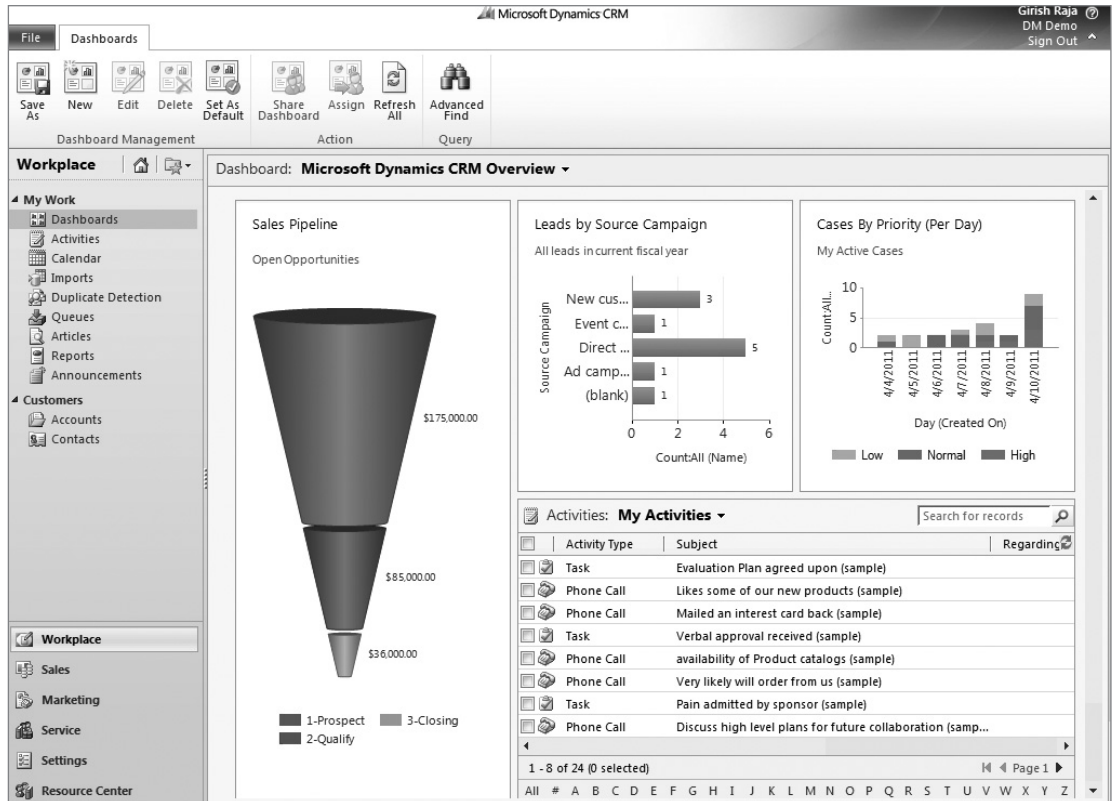


FIGURE 10-1

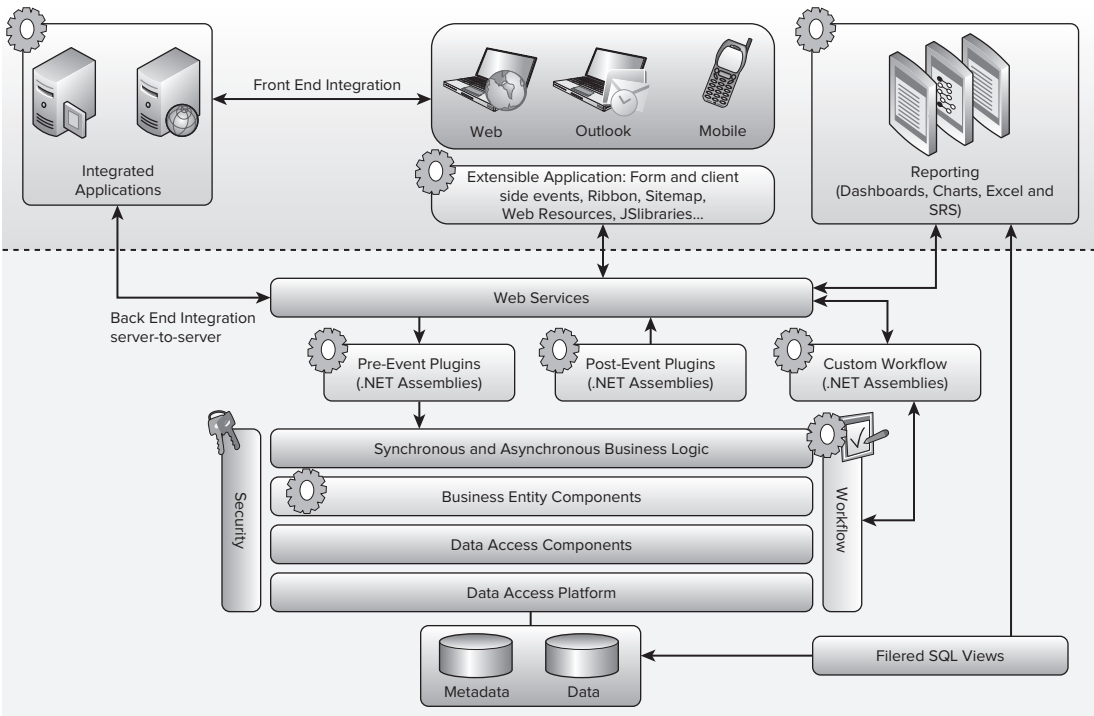


FIGURE 10-2

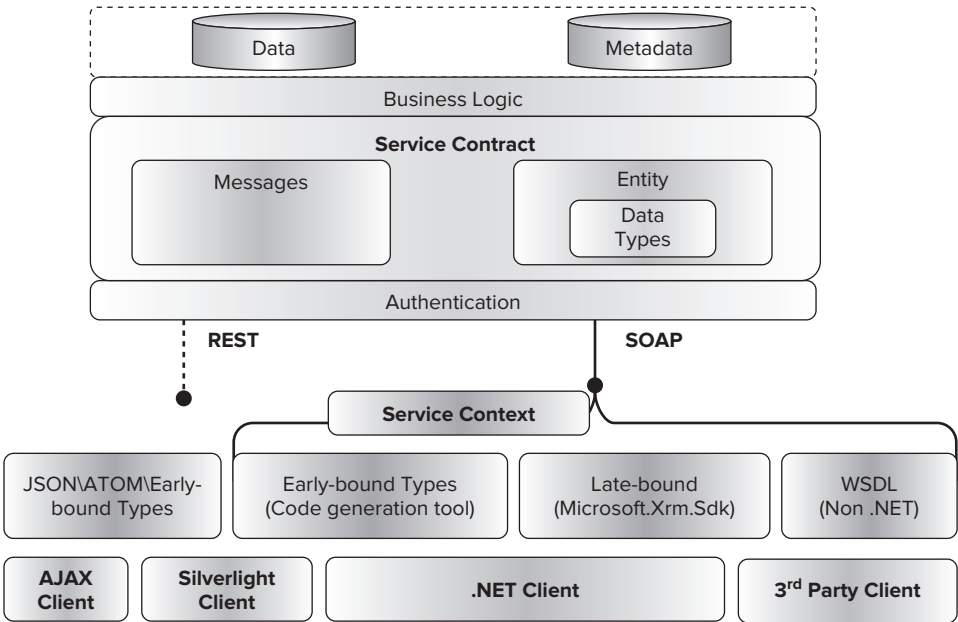


FIGURE 10-3

THE SOLUTION ARCHITECTURE

In this chapter, the objective is to build an account-management dashboard within SharePoint Online that integrates customer-related information from a variety of sources, such as CRM Online, document libraries, and a custom list within SharePoint. Figure 10-4 shows the result of following the steps described in this chapter.

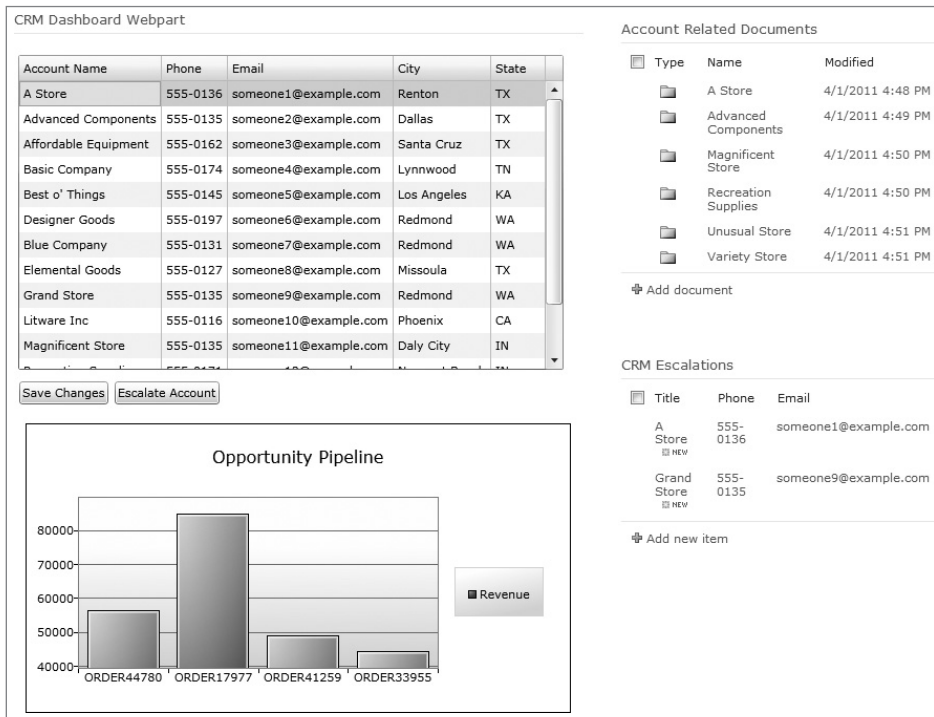


FIGURE 10-4

One of the major components of the dashboard that we'll be building in this chapter is the Silverlight-based custom web part, which displays account information in a grid and the sales pipeline in a nice chart. Figure 10-5 shows the high-level architecture of the Silverlight web part.

The customer data is stored within CRM Online as two different entities: an Account entity and an Opportunity entity. You can think of "entities" in CRM Online as analogous to tables within a SQL Server database.

The web part running on the SharePoint Online page uses a custom Silverlight 4 control. The Silverlight control calls a thin WCF proxy web service running on Windows Azure. The WCF proxy web service, in turn, calls the CRM Online web services to read and write customer data. You can also enable caching on the proxy to improve web part performance. Windows Azure helps with scalability, distribution, and composition of multiple data sources, if necessary. The WCF proxy uses .NET Framework version 4 so that you can use the `Microsoft.Xrm.Sdk` assemblies, which are in .NET 4 as well. The WCF proxy also enables cross-domain web service calls to be made from the

Silverlight control running on SharePoint Online. Caching can be optionally enabled on the proxy to further improve data load performance for the SharePoint Online users.

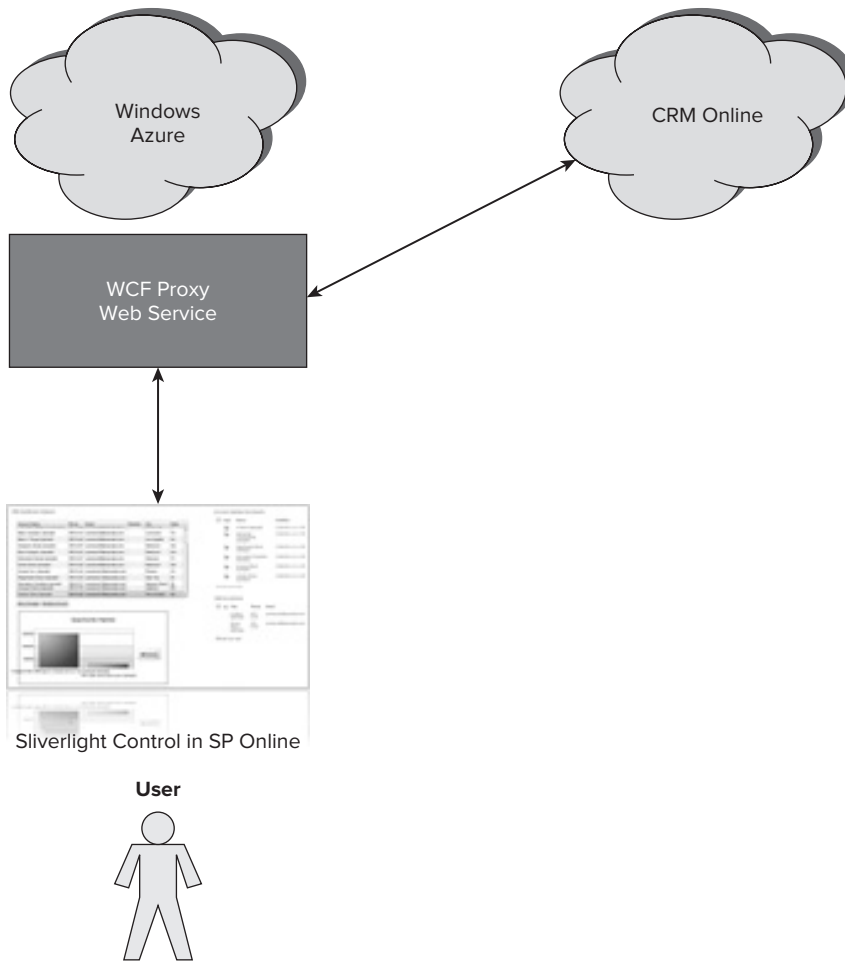


FIGURE 10-5

Using Silverlight with SharePoint

There is a good reason why many developers are building custom UI components for SharePoint using Silverlight. As a platform, SharePoint 2010 natively supports Silverlight out of the box. Because of Silverlight's tight integration with SharePoint, and hence support within Visual Studio, it is easy for developers to build and deploy such applications. Given the sandbox restrictions in SharePoint Online, which don't allow you to host custom server-side web parts, Silverlight web parts that execute on the client make perfect sense.

One of the main reasons we are using Silverlight within this solution is because of Silverlight's capability to seamlessly call external web services (Windows Azure WCF proxy, in this case) in an asynchronous fashion. The client object model within the SharePoint API also has a Silverlight version, making it easy for developers to use Silverlight controls as artifacts within SharePoint. Furthermore, Silverlight, along with Expression Blend, makes it easy for developers and designers to collaborate when creating user interfaces.

Security and Authentication

Any cloud-based system must take into careful consideration the security it employs for both the data it stores and the data in transit. This section provides an overview of the important security measures developers must consider before deploying this solution within their environment. A detailed, step-by-step process for implementing security is beyond the scope of this book, of course.

Data in transit can be secured by using HTTPS (HTTP with SSL/TLS). SharePoint Online and CRM Online have security measures to protect data that is stored; and HTTPS is enabled by default to secure the connection on their end. If an intermediary Azure proxy web service doesn't have SSL enabled, the end-to-end web calls are not fully secure, and the browser might warn users of mixed-mode errors. However, if you enable HTTPS for the proxy as well, you shouldn't see such browser errors with this solution.

Authentication for SharePoint Online can be done using Office 365's built-in authentication methods. In this solution, we are going to assume that not all SharePoint Online users are CRM Online users, so we will be using one set of CRM Online credentials as "Service" credentials for all SharePoint Online users. If every SharePoint Online user is also a CRM user, then, ideally, authentication can be flowed across systems through claims-based federated authentication.



The underlying code in CRM Online uses claims-based authentication using Microsoft's Windows Identity Framework (WIF), although at the time of this writing, Windows Live ID is the only supported authentication mechanism for CRM Online.

BUILDING THE DASHBOARD

In this section, you will build various components of the dashboard. The build process is split into the following three steps:

1. Connect document libraries to CRM Online.
2. Build the Windows Azure proxy.
3. Create the Silverlight grid and chart.

You will then bring together the components to create the dashboard. In order to complete the activities in this chapter, you will need the following developer tools and online accounts:

- **A CRM Online account** — Sign up for a 30-day trial at <http://crm.dynamics.com>.
- **SharePoint Online from Office 365** — Sign up at www.office365.com.
- **Visual Studio 2010 SP1** — If you don't have SP1, you have to download Silverlight Tools for Visual Studio separately.
- **Windows Azure SDK v1.4 and Tools for Microsoft Visual Studio 2010** — You can download this at www.microsoft.com/downloads/en/details.aspx?FamilyID=7a1089b6-4050-4307-86c4-9dadaa5ed018.
- **Silverlight 4 Toolkit for Charts** — Available at <http://silverlight.codeplex.com/releases/view/43528>.
- **CRM 2011 SDK** — Download at www.microsoft.com/downloads/en/details.aspx?FamilyID=420f0f05-c226-4194-b7e1-f23ceaa83b69.
- **Windows Identity Foundation v3.5 runtime** — <http://go.microsoft.com/fwlink/?LinkId=202021>.

Connecting Document Libraries to CRM Online

Once you have provisioned your CRM Online and SharePoint Online subscriptions, the next step is to connect them together for document libraries. CRM Online now enables you to use the document-management capabilities of SharePoint directly from within Microsoft Dynamics CRM. You can store and manage documents in the context of a CRM record on the SharePoint platform. This is made possible by a SharePoint solution called the List component, which the CRM team ships as part of CRM Online.



For more information on this topic and a step-by-step configuration guide to connecting the document libraries to CRM Online, refer to the CRM 2011 SDK topic titled “Integrate SharePoint with Microsoft Dynamics CRM,” at <http://msdn.microsoft.com/en-us/library/gg334768.aspx>. A step-by-step configuration guide can be found at <http://blogs.msdn.com/b/crm/archive/2010/10/08/crm-with-sharepoint-integration-introduction.aspx>.

Once you have configured the connection between SharePoint document libraries and CRM Online, you have completed the first component required for the dashboard. When you browse to the Documents section for a record within CRM (on the left in Figure 10-6), you will see it as native CRM functionality, but the documents are being stored and managed in SharePoint (on the right in Figure 10-6).

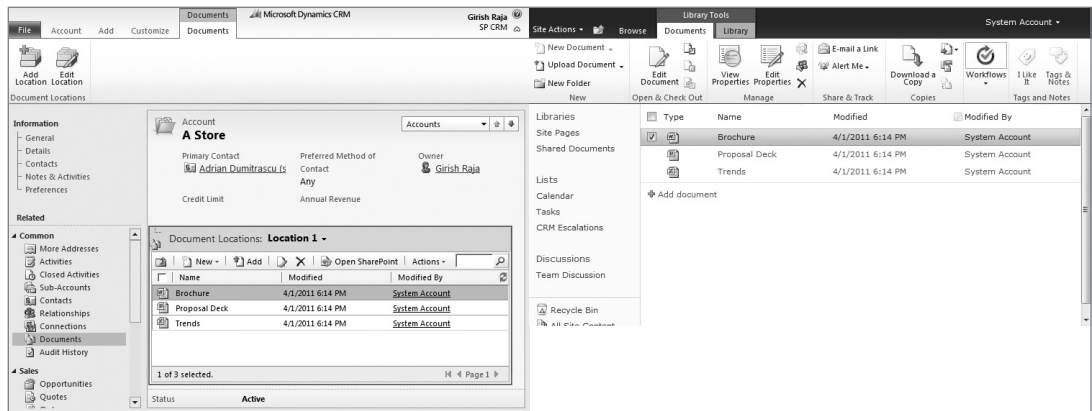


FIGURE 10-6

Building the Windows Azure Proxy

Now that you've configured the connection to document libraries, you can start building the Silverlight part of the dashboard. The Silverlight control is a typical three-tiered application (refer to Figure 10-5), and you'll be building it from the bottom up. Given that the CRM Online and SharePoint Online platforms have already taken care of the data model, you can now build the middle-tier — the WCF web services proxy running in Windows Azure.

1. Open Visual Studio 2010 and click File ⇨ New ⇨ Project. In the New Project dialog, choose Other Project Types ⇨ Visual Studio Solutions ⇨ Blank Solution to create a blank solution. Enter the name of the solution as **WroxCRMOnlineWebpart** and click OK (see Figure 10-7).

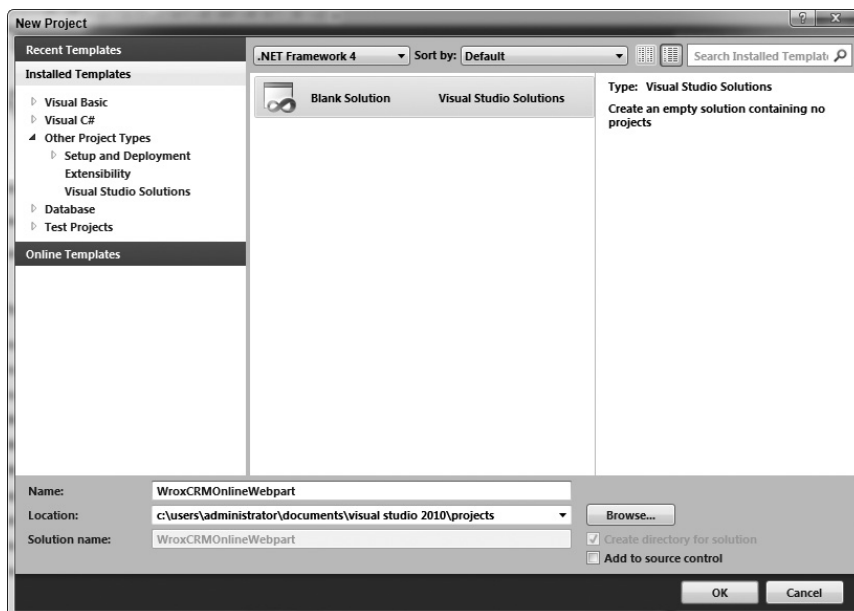


FIGURE 10-7

2. Right-click on the solution in Solution Explorer and choose Add New Project to add a new Windows Azure Project within the solution. Name it **AzureCRMProxy** (see Figure 10-8). Click OK.

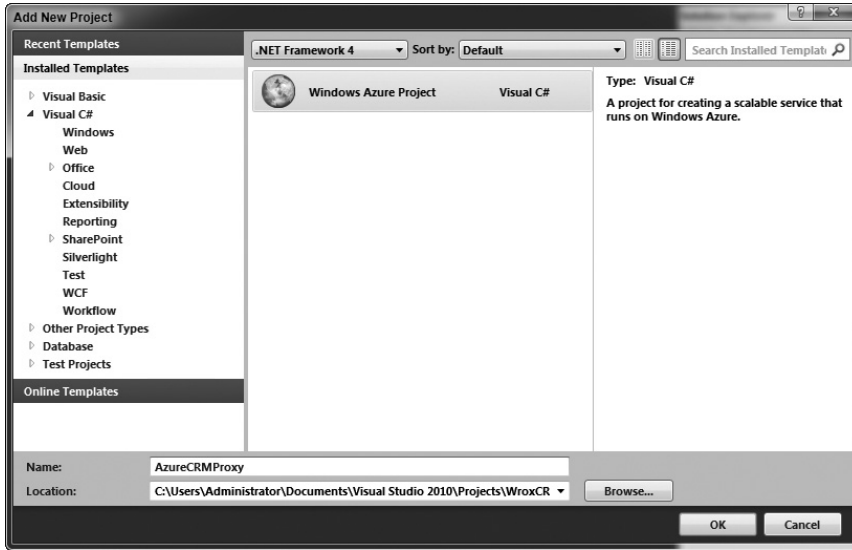


FIGURE 10-8

3. Add a WCF Service Web Role in the New Windows Azure Project dialog.
4. Rename the WCF Service Web Role to **WCFCRMProxyRole**, as shown in Figure 10-9.

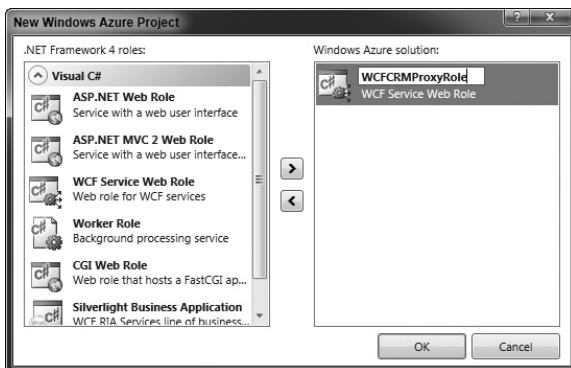


FIGURE 10-9

5. Visual Studio 2010 will create a default WCF service called **Service1**. Within the file **Service1.svc.cs**, use the **Rename** command on the Visual Studio Refactor menu, as shown in Figure 10-10, to change the class name to **CRMProxyService**.

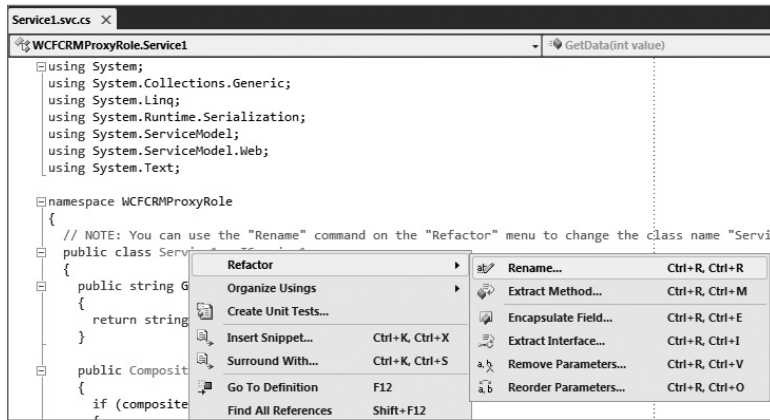


FIGURE 10-10

6. Similarly, rename the appropriate interface class to `ICRMProxyService`.

It's a good practice to change the filenames to appropriate class names. After you have made the changes from steps 5 and 6, your Visual Studio Solution Explorer should look like Figure 10-11.

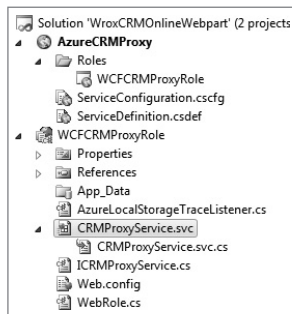


FIGURE 10-11

7. Right-click on the `WCFCRMProxyRole` project, select `Add Reference`, and then click the `Browse` tab to add the `microsoft.xrm.sdk.dll` and `microsoft.crm.sdk.proxy.dll` files. These assemblies, located within the CRM 2011 SDK installation folder under the `sdk\bin` directory, simplify the amount of code needed to interact with the CRM Online WCF web services.
8. Similarly, add a reference to `Microsoft.IdentityModel.dll` from the folder `C:\Program Files\Reference Assemblies\Microsoft\Windows Identity Foundation\v3.5`. If you don't see this file, ensure that you have installed the Windows Identity Foundation v3.5 runtime.
9. Within Visual Studio Solution Explorer, right-click on `Microsoft.IdentityModel.dll` and select `Properties`. Within the `Properties` window, set the value to `Copy Local` to `True`.

10. Also add a reference to `System.Security` (version 4.0.0.0) under the .NET tab in the Add Reference dialog.
11. Now you need to include a couple of helper source code files from the CRM SDK. Browse to the CRM 2011 SDK installation subfolder `sdk\samplecode\cs\helpercode` and copy the files `deviceidmanager.cs` and `myorganizationcrmsdktypes.cs`. Right-click on the `WCFCRMProxyRole` project, select `Open Folder in Windows Explorer`, and paste the two files within the open folder.
12. Within Visual Studio Solution Explorer, select `Show All Files`. The two new copied files will be displayed in Solution Explorer. Right-click on both the files and choose `Include in Project`.
13. To define the `DataContract` for the data you'll need and the `OperationContract` for operations you'll be performing with the data, open the `ICRMProxyService.cs` file and add the following code:



Available for
download on
Wrox.com

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WCFCRMProxyRole
{
    [ServiceContract]
    public interface ICRMProxyService
    {
        [OperationContract]
        List<AzureAccount> GetAccounts();
    }

    // Note that the CRM datatypes defined in myorganizationcrmsdktypes.cs are not
    // compatible with Silverlight client
    [DataContract]
    public class AzureOpportunity
    {
        private string _Name;
        [DataMember]
        public string Name
        {
            get { return _Name; }
            set { _Name = value; }
        }

        private decimal _EstimatedRevenue;
        [DataMember]
        public decimal EstimatedRevenue
        {
            get { return _EstimatedRevenue; }
            set { _EstimatedRevenue = value; }
        }
    }
}
```

```
    }  
}  
  
[DataContract]  
public class AzureAccount  
{  
    [DataMember]  
    public List<AzureOpportunity> Opportunities;  
  
    private string _AccountID;  
    [DataMember]  
    public string AccountID  
    {  
        get { return _AccountID; }  
        set { _AccountID = value; }  
    }  
  
    private string _AccountName;  
    [DataMember]  
    public string AccountName  
    {  
        get { return _AccountName; }  
        set { _AccountName = value; }  
    }  
  
    private string _MainPhone;  
    [DataMember]  
    public string MainPhone  
    {  
        get { return _MainPhone; }  
        set { _MainPhone = value; }  
    }  
  
    private string _Email;  
    [DataMember]  
    public string Email  
    {  
        get { return _Email; }  
        set { _Email = value; }  
    }  
  
    private string _City;  
    [DataMember]  
    public string City  
    {  
        get { return _City; }  
        set { _City = value; }  
    }  
  
    private string _State;  
    [DataMember]  
    public string State  
    {  

```

```

        get { return _State; }
        set { _State = value; }
    }
}
}

```

code snippet 076576 Ch10_Code.zip/ICRMProxyService.cs

14. With the contracts defined, you need the WCF endpoint URL for your instance of CRM Online. You can get it by logging into CRM Online and choosing Settings ⇨ Customizations ⇨ Developer Resources. Your URL will be displayed under Organization Service, as shown in Figure 10-12.

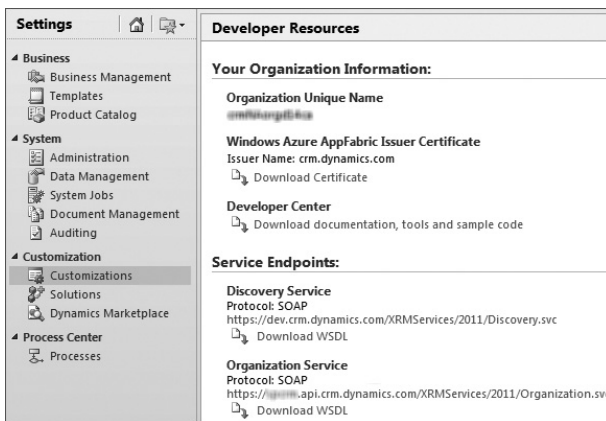


FIGURE 10-12

15. Open the `web.config` file to enter the WCF endpoint URL (referred to as Organization URI), and enter the credentials that you'll be using to call CRM Online web services. The endpoint URL and the credentials referred to here will be used in the code to retrieve data from CRM Online.



Available for
download on
Wrox.com

```

<appSettings>
  <add key="OrganizationUri"
    value="https://yourcrm.api.crm.dynamics.com/XRMServices/2011/Organization.svc"/>
  <add key="WLIDUsername" value="yourwindowsliveemail@live.com"/>
  <add key="WLIDPassword" value="yourpassword"/>
</appSettings>

```

code snippet 076576 Ch10_Code.zip/WCFCRMProxyRole/Web.config



In this chapter, we have entered credentials in plain text within the configuration file for readability and illustrative purposes, but you should use encrypted configuration sections or claims-based/federated authentication in real-world implementations.

- 16.** Open the `CRMProxyService.svc.cs` file and replace its contents with the following code to declare various variables that will be needed:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Client;
using System.ServiceModel.Description;
using System.Configuration;

namespace WCFCRMProxyRole
{
    public class CRMProxyService : ICRMProxyService
    {
        public Uri OrganizationUri;
        public Uri HomeRealmUri = null;
        public ClientCredentials Credentials = null;
        public ClientCredentials DeviceCredentials = null;

        private OrganizationServiceProxy _serviceProxy;
        private IOrganizationService _service;

        public List<AzureAccount> GetAccounts()
        {
        }
    }
}
```

code snippet 076576 Ch10_Code.zip/CRMProxyService.svc.cs

- 17.** Add the following code within the `GetAccounts()` method. In this code segment, we create a reference to the CRM Online web service and retrieve all the accounts from CRM Online and their corresponding sales pipeline. We are using LINQ statements to query CRM Online.



```
...
// Retrieve the CRMOnline Organization from the web.config settings
OrganizationUri = new Uri(ConfigurationManager.AppSettings["OrganizationUri"]);
ClientCredentials Credentials = new ClientCredentials(); ;
Credentials.UserName.UserName = ConfigurationManager.AppSettings["WLIDUsername"];
Credentials.UserName.Password = ConfigurationManager.AppSettings["WLIDPassword"];
DeviceCredentials = Microsoft.Crm.Services.Utility.DeviceIdManager.
    LoadOrRegisterDevice();

// Connect to the Organization service.
// The using statement assures that the service proxy will be properly disposed.
using (_serviceProxy = new OrganizationServiceProxy(OrganizationUri,
    HomeRealmUri,
    Credentials,
```

```
DeviceCredentials))
{
    // This statement is required to enable early-bound type support.
    _serviceProxy.EnableProxyTypes();

    _service = (IOrganizationService)_serviceProxy;
    OrganizationServiceContext orgContext = new
        OrganizationServiceContext(_service);

    List<AzureAccount> azureAccounts = new List<AzureAccount>();
    // Get the list of Accounts from CRMOnline
    azureAccounts = (from a in orgContext.CreateQuery<Account>()
        select new AzureAccount
        {
            AccountID = a.Id.ToString(),
            AccountName = a.Name,
            City = a.Address1_City,
            Email = a.EmailAddress1,
            MainPhone = a.Telephone1,
            State = a.Address1_StateOrProvince,
            Opportunities = new List<AzureOpportunity>()
        }).ToList();

    List<AzureOpportunity> azureOpps = new List<AzureOpportunity>();
    // Get the list of corresponding opportunities from CRMOnline
    var allCRMOpps = (from accountId in
        (from a in azureAccounts
            select Guid.Parse(a.AccountID))
        join o in orgContext.CreateQuery<Opportunity>()
        on accountId equals o.CustomerId.Id
        where o.StateCode.Value == OpportunityState.Open
        select new Opportunity
        {
            CustomerId = o.CustomerId,
            Name = o.Name,
            EstimatedValue = o.EstimatedValue
        });

    // Associate the opportunities to account objects
    foreach (Opportunity opp in allCRMOpps)
    {
        AzureAccount currentAccount = azureAccounts.FirstOrDefault(acc =>
            acc.AccountID.Equals(opp.CustomerId.Id.ToString()));
        currentAccount.Opportunities.Add(new AzureOpportunity()
        {
            Name = opp.Name,
            EstimatedRevenue = opp.EstimatedValue.Value
        });
    }
    return azureAccounts;
}
...
```

code snippet 076576 Ch10_Code.zip/CRMProxyService.svc.cs

18. In order for the Silverlight control to be able to call this WCF web service, add a file called `clientaccesspolicy.xml` within the root of the WCF web role with the following contents:



```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="SOAPAction">
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

code snippet 076576 Ch10_Code.zip/clientaccesspolicy.xml

19. The WCF service is now ready to go! You can right-click on the `AzureCRMProxy` project and select Publish to deploy the service to Windows Azure. For more information on publishing to Windows Azure, refer to <http://msdn.microsoft.com/en-us/library/ff683672.aspx>. You can also right-click on the `WCFCRMProxyRole` project and select Debug to run the service locally within ASP.NET development server.



If you publish to Azure with an instance count of 1, you will see a warning in the Azure portal. In order to maintain the uptime guaranteed by Windows Azure SLA, you need to deploy a minimum of two instances. For more information on Windows Azure SLA, see <http://www.microsoft.com/windowsazure/sla>.

It's a good idea to test this service to ensure it works as expected. You can write a custom command-line application to do the testing, but an easier approach is to use the `WcfTestClient.exe` included with Visual Studio. `WcfTestClient.exe` can be found within the folder `C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\`. For more information on `WcfTestClient.exe`, see <http://msdn.microsoft.com/en-us/library/bb552364.aspx>.

Creating the Silverlight Grid and Chart

With the middle-tier complete, you can now build the user interface components in Silverlight 4:

1. Right-click on the Visual Studio solution and select Add New Project.
2. In the Add New Project dialog, select Silverlight Application from the list of Installed Templates, name the project **CRMSLParts**, as shown in Figure 10-13, and then click OK.

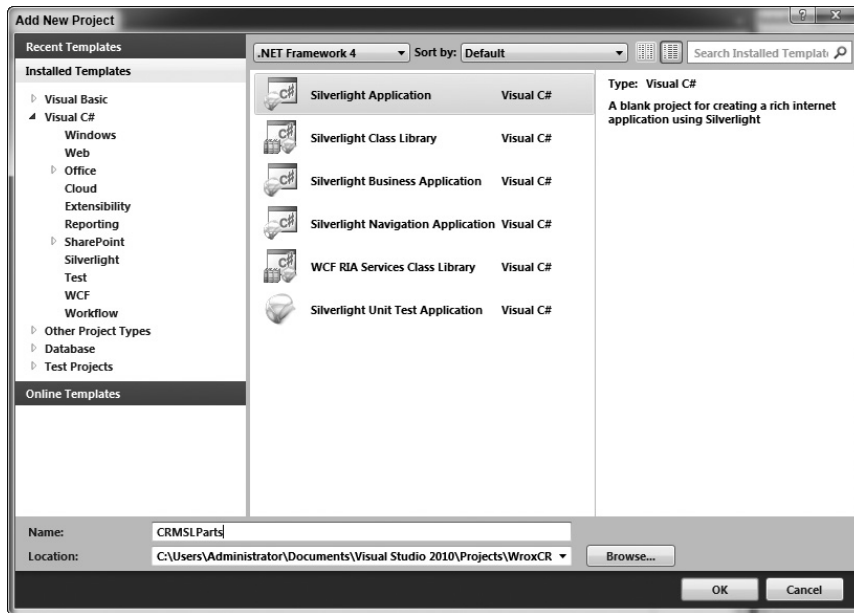


FIGURE 10-13

3. In the New Silverlight Application dialog, select New Web Project to host the test page for the Silverlight control, and then click OK (see Figure 10-14).

Although we won't be using this website as part of the solution, it is very useful for testing and debugging the Silverlight control.

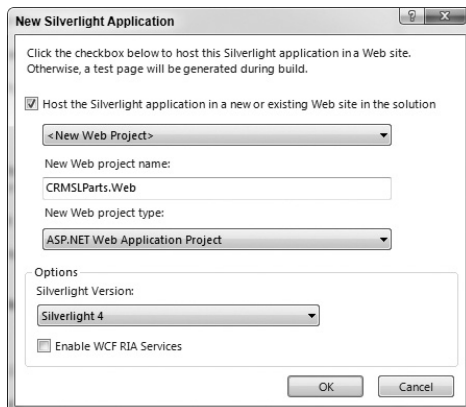


FIGURE 10-14

4. Within the Solution Explorer in Visual Studio, right-click on the CRMSLParts project and select Add Service Reference to refer to the WCF proxy service you built earlier. Enter CRMPProxy as the namespace for the reference. As shown in Figure 10-15, you're adding the proxy from the local emulator, but you could enter the corresponding URL from Windows Azure if you already have it deployed.



You could change the URL referred to in step 4 at a later time by right-clicking on CRMProxy within Solution Explorer ⇨ Configure Service Reference and entering the updated URL.

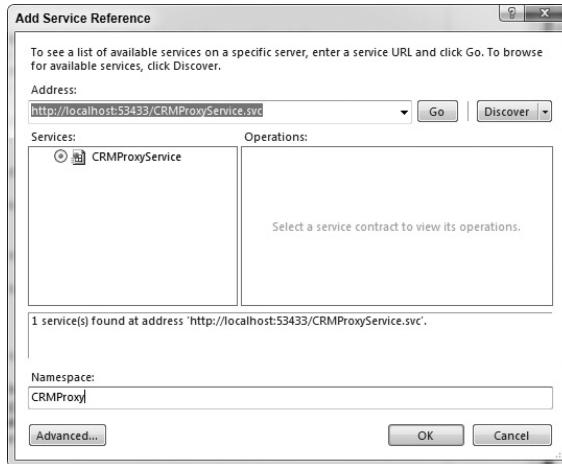


FIGURE 10-15

5. To the same project, also add references to the following four assemblies: `System.Windows.Controls.Data.dll` and `System.Windows.Controls.DataVisualization.Toolkit.dll` from the Silverlight 4 Toolkit, and `Microsoft.SharePoint.Client.Silverlight.dll` and `Microsoft.SharePoint.Client.Silverlight.Runtime.dll` from the SharePoint SDK.

You can find Silverlight 4 Toolkit assemblies within the folder `C:\Program Files (x86)\Microsoft SDKs\Silverlight\v4.0\Toolkit\Apr10\Bin`. The SharePoint SDK assemblies are located in `C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin\`.

6. The next step is to write the code to define the classes required to represent the customer data within the Silverlight project. Create a new folder called `Model` within the project and add a new C# class called `Account.cs` to it. Enter the following code within that file:



Available for
download on
Wrox.com

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.ComponentModel;

using System.Collections;
```

```
using System.Collections.Generic;

namespace CRMSLParts
{
    public class Account : INotifyPropertyChanged
    {
        public List<Opportunity> AccOps;
        private bool _IsDirty = false;
        public bool IsDirty
        {
            get { return _IsDirty; }
            set { _IsDirty = value; }
        }

        private string _AccountID;
        public string AccountID
        {
            get { return _AccountID; }
            set { _AccountID = value; NotifyPropertyChanged("AccountID"); }
        }

        private string _AccountName;
        public string AccountName
        {
            get { return _AccountName; }
            set { _AccountName = value; NotifyPropertyChanged("AccountName"); }
        }

        private string _MainPhone;
        public string MainPhone
        {
            get { return _MainPhone; }
            set { _MainPhone = value; NotifyPropertyChanged("MainPhone"); }
        }

        private string _Email;
        public string Email
        {
            get { return _Email; }
            set { _Email = value; NotifyPropertyChanged("Email"); }
        }

        private string _City;
        public string City
        {
            get { return _City; }
            set { _City = value; NotifyPropertyChanged("City"); }
        }

        private string _State;
        public string State
        {
            get { return _State; }
            set { _State = value; NotifyPropertyChanged("State"); }
        }
    }
}
```

```

protected void NotifyPropertyChanged(string PropertyName)
{
    if (null != PropertyChanged)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(PropertyName));
        this.IsDirty = true;
    }
}

public event PropertyChangedEventHandler PropertyChanged;
}
}

```

code snippet 076576 Ch10_Code.zip/Account.cs

As shown here, you are designing it so that the data can be updated within the Silverlight code. Although we don't include steps to update CRM when the data changes in Silverlight, it should be easy to accomplish.

7. Similar to step 6, add the `Opportunity.cs` class to the model folder and enter the following code:



```

using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace CRMSLParts
{
    public class Opportunity
    {
        private string _OpportunityID;
        public string OpportunityID
        {
            get { return _OpportunityID; }
            set { _OpportunityID = value; }
        }

        private string _Name;
        public string Name
        {
            get { return _Name; }
            set { _Name = value; }
        }

        private decimal _EstimatedRevenue;
        public decimal EstimatedRevenue
        {
            get { return _EstimatedRevenue; }
        }
    }
}

```

```

        set { _EstimatedRevenue = value; }
    }

}
}

```

code snippet 076576 Ch10_Code.zip/Opportunity.cs

- 8.** To add the associated actions upon the data, add `AccountManager.cs` with the following code:



Available for
download on
Wrox.com

```

using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.ObjectModel;
using System.Collections.Generic;

using CRMSLParts.CRMProxy;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Collections;

namespace CRMSLParts
{
    public class AccountManager
    {
        public ObservableCollection<Account> Accounts { get; set; }
        public ObservableCollection<Opportunity> Opportunities { get; set; }

        public AccountManager()
        {
            Accounts = new ObservableCollection<Account>();
            Opportunities = new ObservableCollection<Opportunity>();
            FillData();
        }

        private void FillData()
        {
            CRMProxyServiceClient service = new CRMProxyServiceClient();
            service.GetAccountsCompleted += new
                EventHandler<GetAccountsCompletedEventArgs>
                (service_GetAccountsCompleted);
            service.GetAccountsAsync();
        }
    }
}

```

```

void service_GetAccountsCompleted(object sender,
                                GetAccountsCompletedEventArgs e)
{
    IEnumerable<AzureAccount> crmAccounts =
        (IEnumerable<AzureAccount>)e.Result;

    foreach (AzureAccount account in crmAccounts)
    {
        Account currentAccount = new Account()
        {
            AccountID = account.AccountID,
            AccountName = account.AccountName,
            City = account.City,
            Email = account.Email,
            MainPhone = account.MainPhone,
            State = account.State,
            AccOpps = new List<Opportunity>(),
        };
        foreach (AzureOpportunity opp in account.Opportunities)
        {
            currentAccount.AccOpps.Add(new Opportunity()
            {
                Name = opp.Name,
                EstimatedRevenue = opp.EstimatedRevenue
            });
        }
        Accounts.Add(currentAccount);
    }
}

public void GetOpportunityPipeline(string accountID)
{
    Opportunities.Clear();
    Account acc = FindAccountByID(accountID);
    foreach (Opportunity opp in acc.AccOpps)
    {
        Opportunities.Add(opp);
    }
}

public Account FindAccountByID(string accountID)
{
    foreach (Account acc in Accounts)
    {
        if (acc.AccountID.Equals(accountID,
            StringComparison.OrdinalIgnoreCase))
            return acc;
    }
    return null;
}

public void UpdateData()
{
    // Use Account.IsDirty to determine accounts that have changed

```

```

    // Call the Proxy WebService to update the data back into CRMOnline
  }
}
}

```

code snippet 076576 Ch10_Code.zip/AccountManager.cs

After you've added the `Account.cs`, `Opportunity.cs`, and `AccountManager.cs` files, your Silverlight project structure should look like what is shown in Figure 10-16.

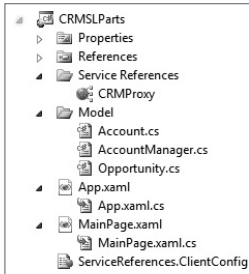


FIGURE 10-16

9. To create the user interface, double-click to open the `MainPage.xaml` file and view it in XAML view.
10. Enter the following **bolded** parameters within the `<UserControl>` XAML tag:



Available for
download on
Wrox.com

```

<UserControl x:Class="CRMSLParts.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:my="clr-namespace:CRMSLParts"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="400"
xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls
.Data"
xmlns:chartingToolkit="clr-namespace:System.Windows.Controls.DataVisualization
.Charting;assembly=System.Windows.Controls.DataVisualization.Toolkit">

```

code snippet 076576 Ch10_Code.zip/MainPage.xaml

11. Within the `<UserControl>` tag, enter the following snippet to define the view source:

```

<UserControl.Resources>
  <my:AccountManager x:Key="accountManagerViewSource" />
</UserControl.Resources>

```



The WPF Designer in Visual Studio might show an unhandled exception at this point, but you can safely ignore it.

- 12.** Replace the <Grid> tag in XAML with the following code, where you'll be creating the data grid, buttons, and chart. These controls will be laid out within the grid using a set of stack panels:



Available for
download on
Wrox.com

```
<Grid x:Name="LayoutRoot" Background="White">
    <StackPanel Orientation="Vertical" Margin="15">
        <data:DataGrid AutoGenerateColumns="False"
            DataContext="{Binding Path=Accounts,
            Source={StaticResource accountManagerViewSource}}"
            ItemsSource="{Binding}" SelectionChanged="dg_SelectionChanged"
            HorizontalAlignment="Left" Margin="5" Height="300" Name="dg"
            IsReadOnly="False" HeadersVisibility="Column"
            Grid.Row="0" Grid.Column="0" >
            <data:DataGrid.Columns>
                <data:DataGridTextColumn Binding="{Binding AccountName}"
                    Header="Account Name" />
                <data:DataGridTextColumn Binding="{Binding MainPhone}"
                    Header="Phone" />
                <data:DataGridTextColumn Binding="{Binding Email}"
                    Header="Email" />
                <data:DataGridTextColumn Binding="{Binding City}" Header="City" />
                <data:DataGridTextColumn Binding="{Binding State}"
                    Header="State" />
            </data:DataGrid.Columns>
        </data:DataGrid>
        <StackPanel Orientation="Horizontal" Margin="3" Grid.Row="1">
            <Button Margin="3" Content="Save Changes" x:Name="btnSave"
                Click="btnSave_Click" />
            <Button Margin="3" Content="Escalate Account" x:Name="btnPublishToSP"
                Click="btnPublishToSP_Click" />
        </StackPanel>
        <chartingToolkit:Chart HorizontalAlignment="Left" Margin="12,12,0,0"
            Name="chart1" Title="Opportunity Pipeline"
            VerticalAlignment="Top" Height="276" Width="520">
            <chartingToolkit:Chart.Series>
                <chartingToolkit:ColumnSeries Title="Revenue"
                    ItemsSource="{Binding}"
                    IndependentValueBinding="{Binding Name}"
                    DependentValueBinding="{Binding EstimatedRevenue}" />
            </chartingToolkit:Chart.Series>
        </chartingToolkit:Chart>
    </StackPanel>
</Grid>
```

code snippet 076576 Ch10_Code.zip/MainPage.xaml

- 13.** Right-click the SelectionChanged within the <DataGrid> tag to automatically generate the C# event handler within the file MainPage.xaml.cs, and then enter the following code:



Available for
download on
Wrox.com

```
private void dg_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    AccountManager dataObject = this.Resources["accountManagerViewSource"]
                                as AccountManager;
    dataObject.GetOpportunityPipeline((dg.SelectedItem as Account).AccountID);
}
```

```
private void btnPublishToSP_Click(object sender, RoutedEventArgs e)
{
    // To be implemented.
}
private void btnSave_Click(object sender, RoutedEventArgs e)
{
    // To be implemented.
}
```

code snippet 076576 Ch10_Code.zip/MainPage.xaml.cs

This code enables the chart to show the corresponding opportunity data when an account is selected in the grid.



For readability, we have written some of the business logic code in MainPage.xaml.cs. In real-world implementations, it is recommended to follow the Model-View-ViewModel (M-V-VM) pattern.

- 14.** Within the MainPage class, add the following bolded code to wire up the initial Loaded event:



Available for
download on
Wrox.com

```
public MainPage()
{
    InitializeComponent();
    Loaded += new RoutedEventHandler(MainPage_Loaded);
}

void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    AccountManager dataObject = this.Resources["accountManagerViewSource"]
                                as AccountManager;
    ColumnSeries column = chart1.Series[0] as ColumnSeries;
    column.ItemsSource = dataObject.Opportunities;
}
```

code snippet 076576 Ch10_Code.zip/MainPage.xaml.cs

- 15.** The Silverlight control is almost ready. You can test it by right-clicking on the CRMSPartParts. Web project and choosing Debug ➔ Start New Instance to verify.

The next step is to wire the Silverlight control with the SharePoint web part.

- 16.** Right-click on the solution and add a new project. Choose Empty SharePoint Project and name it CRMSPWebPartProj, as shown in Figure 10-17, and then click OK.
- 17.** In the dialog that appears, enter the URL of the SharePoint site you will be using and deploy it as a sandboxed solution. Click OK.



The URL can point to your on-premises development environment, but because you're creating it as a sandboxed solution, it works with both SharePoint 2010 Online and SharePoint On-Premises.

18. Right-click on the SharePoint project and select Add ⇄ New Item. In the Add New Item dialog, create a new module named XAPModule (see Figure 10-18), and click Add.

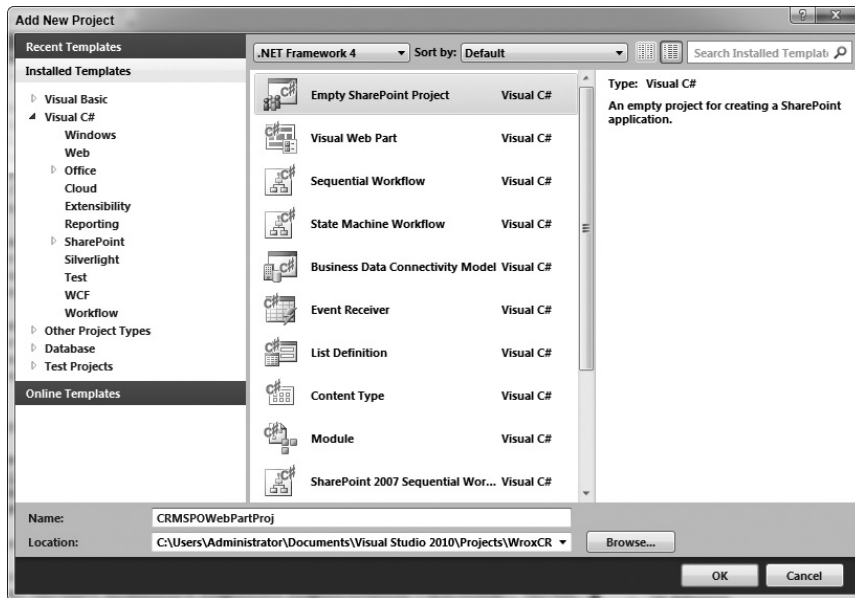


FIGURE 10-17

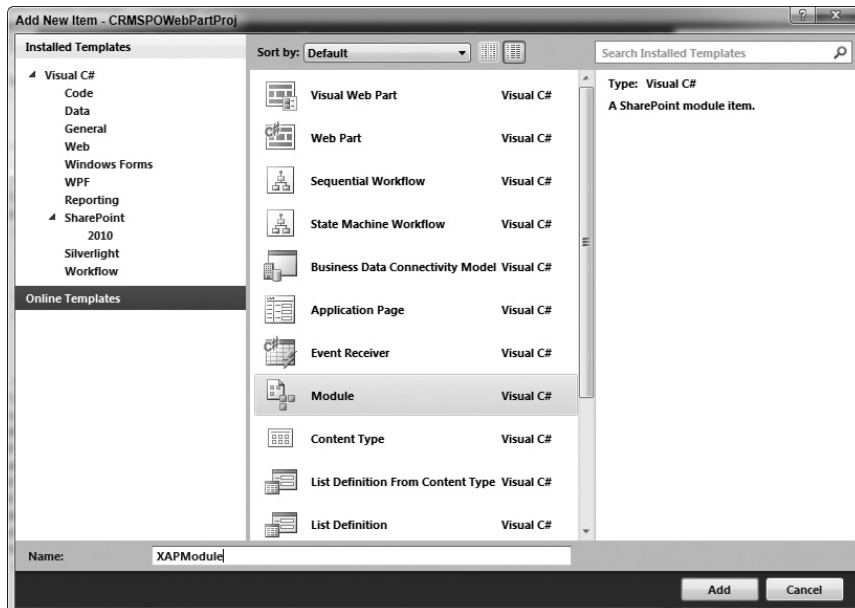


FIGURE 10-18

19. Delete the `Sample.txt` file within the `XAPModule` folder in Solution Explorer and add the following contents to the `Elements.xml` file:



Available for
download on
Wrox.com

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Name="XAPModule" Url="XAP_Bin">
    <File Path="CRMSLParts.xap" Url="CRMSLParts.xap" />
  </Module>
</Elements>
```

code snippet 076576 Ch10_Code.zip/Elements.xml

20. Right-click on `XAPModule` in Solution Explorer, select **Add** ⇨ **Existing Item**, and then browse to the folder containing the XAP output of the Silverlight project you created earlier. The folder can be found within the solution under `WroxCRMOnlineWebpart\CRMSLParts\Bin\Debug`.
21. Find the file `CRMSLParts.xap` within that folder and instead of clicking the **Add** button, click the downward arrow next to the button and select **Add As Link**, as shown in Figure 10-19.

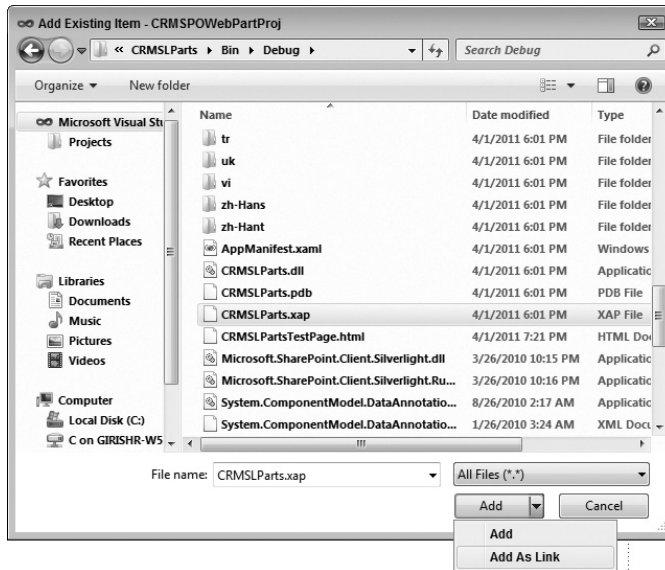


FIGURE 10-19



This step enables the SharePoint web part to automatically refer to the latest version of the Silverlight control if you make any changes to it later.

- 22.** Create a new web part called **CRMDashboard**. To do so, right-click on the project and select Add New Item, and then navigate to the SharePoint 2010 node and select Web Part. In the `CRMDashboard.cs` file, add the following code.

Note that you'd have to replace the reference to `intranet.contoso.com` within this code to your SharePoint instance that you chose earlier in step 17.



```
using System;
using System.ComponentModel;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;

namespace CRMSPOWebPartProj.CRMDashboard
{
    [ToolboxItemAttribute(false)]
    public class CRMDashboard : WebPart
    {
        protected override void CreateChildControls()
        {
            string slStreamCode = "<div id=\"slApp\"/>" +
                "<script language=\"JavaScript\" type=\"text/javascript\">" +
                "var slSPDIV = document.getElementById('slApp');" +
                "slSPDIV.appendChild(slSPDIV);" +
                "slSPDIV.innerHTML =" +
                "'<object data=\"data:application/x-silverlight,\" " +
                " type=\"application/x-silverlight\" width=\"740\" height=\"800\">" +
                "<param name=\"source\" " +
                "value=\"http://intranet.contoso.com/XAP_Bin/CRMSLParts.xap\"/> " +
                "<param name=\"initParams\" value=\"MS.SP.url=" +
                SPContext.Current.Site.Url + "\" /></object>';" +
                "</script>";

            this.Controls.Add(new LiteralControl(slStreamCode));

            base.CreateChildControls();
        }
    }
}
```

code snippet 076576 Ch10_Code.zip/CRMDashboard.cs

At this point, you merely have a Silverlight control surfaced in a SharePoint web part. In the next step, you'll wire up this web part to connect to a SharePoint custom list.

- 23.** In your SharePoint site, create a new custom list called **CRM Escalations** with the columns Title, Phone, and Email (see Figure 10-20).

Columns	
A column stores information about each item in this list:	
Column (click to edit)	Type
Title	Single line of text
Phone	Single line of text
Email	Single line of text
Created By	Person or Group
Modified By	Person or Group
Create column	
Add from existing site columns	

FIGURE 10-20

- 24.** Go back to the CRMSPartS Silverlight project and connect it with this custom SharePoint list. Open the `MainPage.xaml.cs` and add the following code highlighted in bold:



Available for
download on
Wrox.com

```
...
using Microsoft.SharePoint.Client;

namespace CRMSPartS
{
    public partial class MainPage : UserControl
    {
        List CRMEscalations;
        Account selectedAccount;

        public MainPage()
        ...
```

code snippet 076576 Ch10_Code.zip/MainPage.xaml.cs

- 25.** Add the following code within the `MainPage` class to wire up the `btnPublishToSP_Click` handler.

Again, remember to replace the reference to `intranet.contoso.com` within this code to the SharePoint instance that you chose earlier in Step 17.



Available for
download on
Wrox.com

```
private void btnPublishToSP_Click(object sender, RoutedEventArgs e)
{
    ClientContext context;

    if (App.Current.IsRunningOutOfBrowser)
    {
        context = new ClientContext("http://intranet.contoso.com");
    }
    else
    {
        context = ClientContext.Current;
    }

    //Get the list
    CRMEscalations = context.Web.Lists.GetByTitle("CRM Escalations");
    context.Load(CRMEscalations);

    //Create the escalation list item
    ListItemCreationInformation itemCreateInfo =
        new ListItemCreationInformation();
```

```

ListItem listItem = CRMEscalations.AddItem(itemCreateInfo);
selectedAccount = dg.SelectedItem as Account;
listItem["Title"] = selectedAccount.AccountName;
listItem["Phone"] = selectedAccount.MainPhone;
listItem["Email"] = selectedAccount.Email;
listItem.Update();

//Make the call to the SharePoint server
context.ExecuteQueryAsync(
    succeededCallback,
    failedCallback);
}

void succeededCallback(object sender, ClientRequestSucceededEventArgs args)
{
    this.Dispatcher.BeginInvoke(() =>
    {
        MessageBox.Show(selectedAccount.AccountName +
            " has been escalated to the Senior Leadership Team");
    });
}

void failedCallback(object sender, ClientRequestFailedEventArgs args)
{
    this.Dispatcher.BeginInvoke(() =>
    {
        MessageBox.Show(
            string.Format("Failed - msg:{0} errcode:{1} stackTrace:{2}",
                args.Message, args.ErrorCode, args.StackTrace));
    });
}

```

code snippet 076576 Ch10_Code.zip/MainPage.xaml.cs

In the preceding snippet, you are using the client object model from the SharePoint SDK to connect CRM Online data and the custom SharePoint list through the UI. Whenever a user clicks the Escalate Account button in the UI, the code adds the customer account details from CRM Online to the SharePoint custom list.



In the preceding code snippet, we assume that the CRM Escalations list is available at the top site collection level; hence, we refer to it using the context.Web.Lists collection. If the CRM Escalations list is created in a sub-site, modify the code snippet to load context.Web.Webs and refer to the list within the context.Web.Webs[subsiteIndex].Lists collection.

26. Right-click on the SharePoint project and select Package to bundle your SharePoint web part as a .wsp solution file. If you were using the local Azure emulator up until now, you need to deploy your project to Windows Azure before you package and publish your solution. You can upload this .wsp file to the SharePoint Online solution gallery, after which it will be available for use in SharePoint Online pages.

You now have a working Silverlight web part capable of running in SharePoint Online and able to bring data from CRM Online. You can extend this coding pattern to bring data from other LOB data sources in your organization.

Bringing It All Together

At this point, you are done coding the solution. Now it's time to build a dashboard page that hosts all the web parts you have built until now.

1. Log in to your SharePoint Online site and create a new web part page with a three-column layout (header, right column, and body), as illustrated in Figure 10-21.
2. Click the Page Ribbon menu and select Edit Page.
3. Click the Add a Web Part hyperlink. This will open the Page Tools ribbon menu, as shown in Figure 10-22.
4. Choose Web Part followed by Custom (under Categories) to select the custom web part that you built in this chapter, and then click the Add button (ensuring the drop-down next to it is set to "Body") to place it in the body zone of the page.

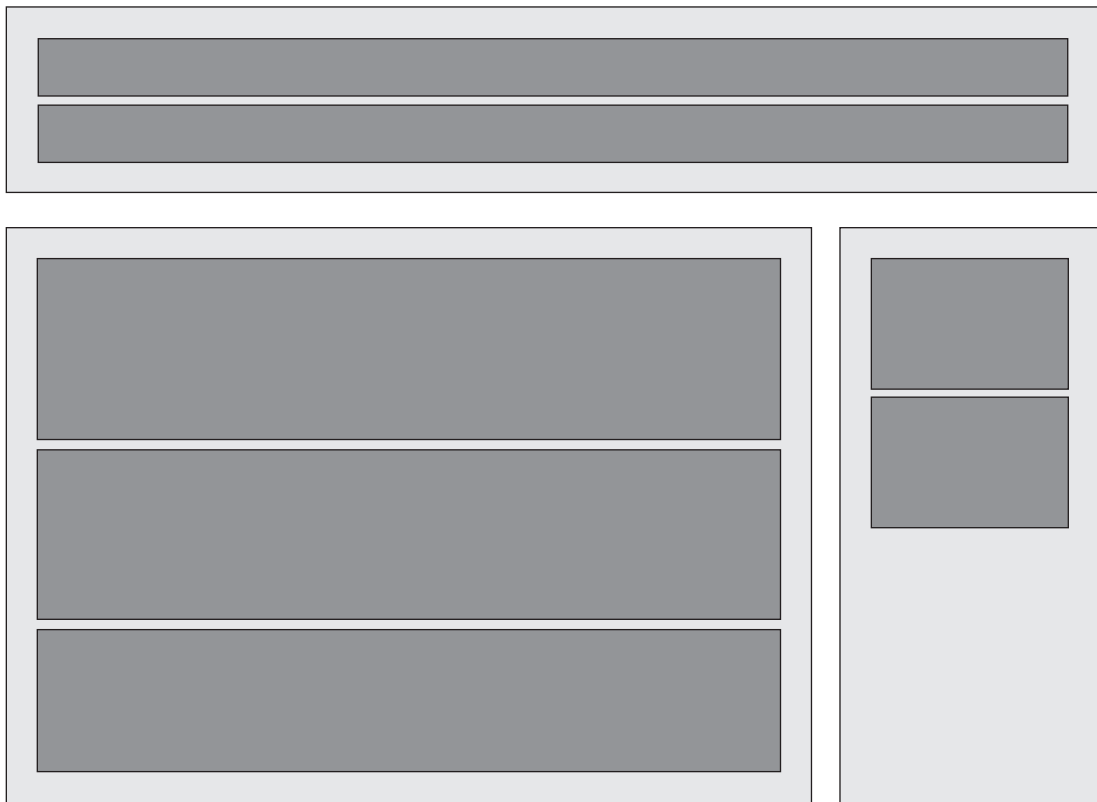


FIGURE 10-21

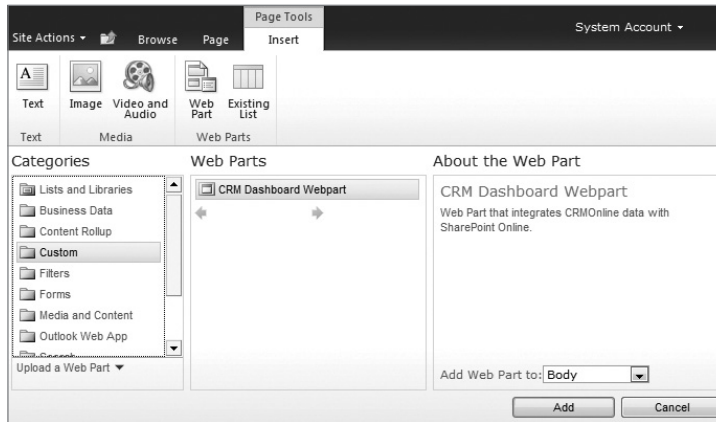


FIGURE 10-22

5. Similar to step 4, add web parts for rest of the artifacts (e.g., the document library for accounts and the CRM Escalations custom list) that you created earlier in this chapter. After you have added all the web parts to the page, the layout of the page should resemble Figure 10-23.

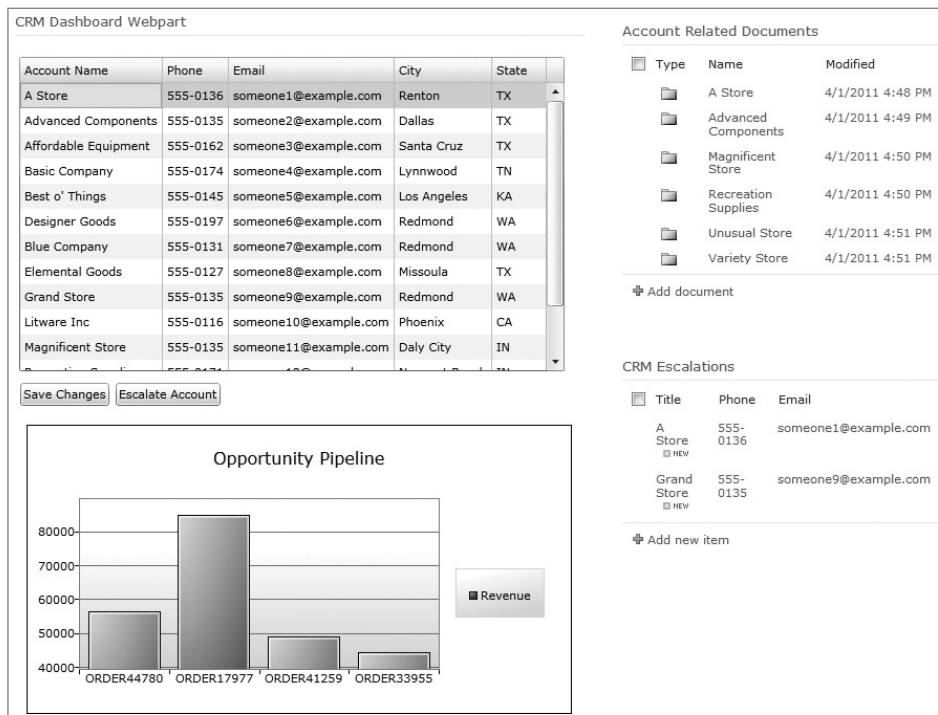


FIGURE 10-23

The Silverlight web part will display the list of accounts from CRM momentarily. Click through those accounts to see the opportunity pipeline charts changing accordingly.

6. With an account selected, you can click the Escalate Account button, which will add it to the escalations list. Refresh the page to see the new record in the CRM Escalations web part. You can also create a simple workflow on the CRM Escalations list to e-mail the escalation team, if necessary.

Congratulations! You have now created a nice end-to-end customer information dashboard in SharePoint Online.

SUMMARY

SharePoint Online provides a flexible platform that enables you to create composite applications rapidly. CRM Online complements SharePoint Online by providing an easy way for developers to extend it and create compelling integrated applications. Furthermore, the APIs of SharePoint and CRM are streamlined so that the methods described in this chapter can also be used to build similar applications for local, on-premises-based software. The simple solution we walked through in this chapter provides a good foundational pattern that you can build on to create much more complex, advanced solutions. For example, using the example described in this chapter, you could consider ways to extend this solution to include more heterogeneous LOB data sources, creating a one-stop shop for your customers and business users.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- *Beginning SharePoint 2010 Development* — Steve Fox (Wrox, 2010)
- **CRM 2011 Developer Training Course** — <http://go.microsoft.com/?linkid=9762116>
- **CRM 2011 SDK** — www.microsoft.com/downloads/en/details.aspx?FamilyID=420f0f05-c226-4194-b7e1-f23ceaa83b69
- **Girish Raja's blog** — <http://blogs.msdn.com/girishr>

11

Securing Cloud Solutions Using Claims-Based Authentication

WHAT'S IN THIS CHAPTER?

- Understanding claims-based identity
- Understanding the relationships among identity providers, federation providers, and relying party applications for building solutions
- Configuring a trust relationship between Active Directory Federation Services v2.0 (AD FS v2.0) and the Azure AppFabric Access Control service
- Building a claims-based Azure web application that is accessed from within SharePoint with single sign-on

We have all encountered claims-based identity through the course of our daily activities without even really recognizing it. It can be as simple as showing some form of identification in exchange for a stamp on the hand to a sophisticated process that formally checks and validates multiple forms of identification before providing a smart card that allows access to top-secret locations. Claims-based security is about one entity trusting another entity that issues information about a subject. Then, in turn, the first entity issues information about the subject that can be relied upon by those that choose to trust it.

To implement this claims-based pattern with technology so that it has the broadest reach and adoption possible, you must use interoperable data structures for the information and protocols for handling these data structures that are based on open standard specifications. Open, standards-based specifications can be implemented in any number of programming languages and therefore be available on any operating system platform to enable the desired level of interoperability. This is the essence of the various claims-based technology implementations available in the market today. Let's take a deeper look and see where claims-based technologies can help us in our SharePoint and cloud integration.

OVERVIEW OF CLAIMS-BASED IDENTITY

Claims-based identity is not a new concept, either in the practical world or within technology; it's just that we don't readily identify it as such in our day-to-day activities as easily as we can recognize it when it's implemented with technology. Therefore, before we delve into claims-based identity within its technical implementation, this section looks at a practical scenario you have likely encountered, which you can use to help connect the dots to understand the concepts of claims-based identity when applied in technology.

Morgan arrives at the pharmaceutical conference eager and ready to participate for the next three days. She will have a triple role within the conference. First, since her company is a conference sponsor, she will spend part of her time in the company booth in the exhibit hall talking with conference attendees. Second, she is a conference speaker for one of the sessions. And, finally, she will be attending as many other sessions as possible for personal and professional enrichment.

Her first stop is at the registration booth, where Dean asks her to present a picture ID, which can only be a valid driver's license or passport. Upon verifying her identity, the registrar provides her with a conference badge that includes her name, company name, bar code identifier, and three different colored ribbons to attach to the badge: a red exhibitor ribbon, a yellow speaker ribbon, and a white attendee ribbon. As you no doubt have surmised, each ribbon has different rights and privileges associated with it within the conference site.

Morgan tucks away her picture ID, securely attaches the ribbons to the badge, hangs it around her neck, and proceeds to the area to pick up the conference materials. At the distribution table, the person sees the white ribbon and provides a conference bag with all the materials for an attendee. Morgan then wants to see the booth area, so she visits the exhibit hall. Although it's not yet open to attendees, the exhibit hall doorkeeper sees her red exhibitor ribbon and grants her access into the hall. After a brief visit with her colleagues, Morgan heads to the speaker preparation room to add the final touches to her presentation. Here again she is granted access, as she has the required yellow speaker ribbon.

This simple scenario illustrates the components of a claims-based identity system. There are specific *identity providers* (IPs) that the conference registrar will *trust*. In this case it is either a state department of licensing agency that has verified an individual's identity and issued a driver's license, or a government that has performed the verification and issued a passport. In either case the document they issue is considered an acceptable *security token* and can be validated by checking its specific security markings that warrant its authenticity.

The security token itself contains one or more *claims* — that is, statements about the *subject*, such as name, birth date, height, weight, photograph, and so on. These are called claims because they are supposedly true statements about the subject, but they are trusted only to the extent to which the IP that issued the security token is trusted. In other words, Morgan could have showed the registrar another security token — for example, her corporate picture ID card — that had almost all the same claims as her driver's license. Even though this company-issued security token is truly authentic, and valid for Morgan in other contexts, it is not recognized by the conference. Although her company, as an IP, issued the security token with claims about her, the registrar could only accept trust statements about her when they were present in security tokens issued by state or government IPs.

A *security token service* (STS) determines the trust relationships it will have with other STSs in a claims-based identity system. *Trust relationships* can be established between an IP STS and a *federation provider* (FP) STS, sometimes called a *resource STS* (R-STS). The registrar in this scenario performs the role of the FP STS. The registrar verified that the security token provided for identification was authentic and from a trusted IP STS — and note his next action. In turn, he issued a brand-new security token: the badge, with its claims in the form of text, name, company, etc., and colored ribbons. It is this new security token that is trusted within the conference site (*security domain*) by all the different parties that will rely upon it. Essentially, the registrar accepted one security token, validated its authenticity, and then issued another security token scoped for another security domain, the conference. For Morgan, the original security token was no longer needed. Once the badge and ribbons were issued by the registrar FP STS, her driver's license was put away and only her badge was required within the conference site.

Within the conference security domain, the *relying party* (RP) applications are the doorkeepers at the various session rooms, speaker preparation rooms, and exhibit halls. The RP cares nothing about whether the conference attendee showed a driver's license or passport to the registrar. The RP needs only to trust the registrar FP STS, and relies upon it to provide the attendee with the conference badge security token, with its appropriate claims. Upon the basis of this trust, the RP can check the colored ribbon claim to grant/deny access to the appropriate venue. The additional claims on the badge are not required for access, but can be captured by handheld scanning devices to track attendance at each session and to collect leads by exhibitors in the exhibit hall.

Does this scenario sound familiar? This is claims-based identity in action, and it happens in many aspects of our daily lives; we just don't readily recognize it. Airport security is essentially the same. You check in, have your identity validated, and then receive a boarding pass. Again, you pass from one security boundary to another by having one IP security token validated by a FP STS, which issues another security token, a boarding pass, that is relied upon at the gate where you board the plane. In other words, these components of a claims-based identity system — subject, trust, IP, STS, FP, security domain, security token, and claims are parts of everyday life.

Claims-based identity, therefore, is based on the ability of a security token service to encapsulate claims about a subject within a security token structure and issue the security token. Trust relationships are established between identity provider STSs, resource STSs, and relying party applications so that authenticity of an issued security token can be verified before being trusted by the relying party.

In this chapter, you'll see the claims-based identity components called out within specific technical implementations. This can often be somewhat confusing; however, the preceding two analogies will help you roll back the technical details into a practical scenario you are familiar with.

WS-Federation vs. WS-Trust

As mentioned in the previous section, claims-based identity is not a new concept; it dates back to the early 2000s, when a number of technical implementations emerged, including SAML, Shibboleth, WS-Trust, and WS-Federation. Volumes have been written about these implementations and it is not the intent of this chapter to discuss the technical details. The focus of this chapter is to provide a practical implementation of the technology to help you see how a claims-based identity solution comes together.

As noted in the conference scenario, many situations require getting a security token from one security domain that can be trusted by another security domain so that a subject's identity can flow unimpeded to target applications and services. To accomplish this, you must use interoperable data structures for tokens and protocols that are based on open-standard specifications, not proprietary or platform-bound specifications. Open, standards-based specifications can be implemented in numerous programming languages and therefore made available on any operating system platform to enable this desired level of interoperability.

Two such standards-based implementations are WS-Trust and WS-Federation. WS-Federation provides a request/response pattern for requesting and receiving a security token from an STS via the browser through a series of redirects. Conversely, unlike this passive browser model, WS-Trust is active. It describes the pattern used for security token request/response for interactions between rich/smart client applications and web services. The solution you will build in this chapter relies on WS-Federation.

The Power of Claims-Based Identity

Any application that needs some level of personalization must be able to answer two questions. First, who are you? Second, what are you allowed to do? Answering the first question is the process of *authentication*, in which users must prove they are who they say they are — canonically, via a username and password. The second question, which isn't applicable in all applications, deals with what a subject can do within the application once authenticated. This is *authorization*. Together, these processes describe the concept of identity management, which includes a user's identity, credentials, attributes, and, potentially, roles, etc.

To provide a personalized experience and security within an application, typically the application relies on interaction with an *identity store* for authentication and to set the authorization permissions within the application. At a minimum, an identity store contains user names and passwords; generally, however, it will contain as much information about each user as necessary for an application to provide a rich personalized experience. When the user logs in to an application, the identity store is queried, credentials are validated, and user information is returned to the application to light up their personalized experience, including setting their access permissions within the application.

A common problem with identity stores is that they are tightly coupled with their associated application. Therefore, each application has an identity store that is not shared with any other application because it, too, has its own identity store. We are all well familiar with the result of this issue: Across the Web, we must manage dozens of unique user IDs and passwords for the many sites we interact with on a frequent or infrequent basis. Of course, we can't change how Web entities choose to interact with us, but as developers we can choose our strategy for implementing identity in our applications. Why follow this model of siloed, coupled identity stores with each application? Wouldn't it be much better to share a common identity store across multiple applications? Or be able to accept identities from other identity stores where users are already being managed so that you don't need to perform that function? All in all, identity management comes at a cost. Typically, this is in terms of user life-cycle management, as well as ongoing end user interactions for password resets and account question resolution.

This is where the power of claims-based identity provides one of its best benefits for application developers. By implementing a claims-based architecture in your application, you decouple authentication

and the identity store from the application, and rely only on the security token, with its claims, in your application. The end user who wants to access your application must first authenticate to an IP STS by supplying whatever credentials it requires. The IP STS issues a security token so your application only needs to validate that it was provided from a trusted IP. That way, the application can simply use the provided claims to carry out its business. This also means that any IP STS that can provide the set of claims your application relies upon can become a trusted STS to your application. This gives your application a much broader reach and you don't bear the cost of managing all the identities. Your application simply manages trust relationships with STSs, but more on that in the "Relying Party Applications, Federation Providers, and Identity Providers" section later in this chapter.

An additional benefit of externalizing authentication to an IP STS and relying solely on the claims provided is that the claims do not necessarily need to even identify the user by name. Maybe your application only needs to have a claim that verifies a user is at least 21 years old. This security token and its claims could be provided by a trusted IP STS that vouches for someone's age. This is the kind of flexibility offered by claims. Claims can be much more descriptive than the standard roles or group memberships that are provided from a directory in non-claims-based systems. Typically, claims can be derived from any number of attribute stores, not just a directory. For example, suppose you want to authorize access to a particular part of your application to someone with a specific shoe size or who drives a particular vehicle; this could be done with claims.

In summary, claims-based identity provides developers with a robust security mechanism, one that is based on open industry standards, externalizes authentication from the application, allows identities to be reused across applications and security domains by decoupling the application from its identity store, and provides a consistent architectural pattern for use across applications. These characteristics enable the development of scalable, maintainable, service-oriented solutions that need to span security domains, devices, and operating systems.

SHAREPOINT AND CLAIMS

A significant amount of work was put into SharePoint 2010 to better support claims-based authentication than its previous versions. SharePoint 2010 has two authentication modes available: Classic-Mode Authentication and Claims-Based Authentication. Classic-mode supports Windows NT and forms-based authentication methods, just as the previous versions of SharePoint did. Claims-based supports Windows NT, forms-based, and SAML 1.1/WS-Federation authentication methods. Moreover, you can now configure multiple authentication methods on a single zone. So, if you are using claims-based authentication, you could have all three authentication methods on the same zone. Then SharePoint, irrespective of how the user is authenticated, will create a claims-based identity SAML token and the SPUser object based on that token. Another great thing about this enhancement is that the URL remains the same for all the authentication types. This makes a nicer end-user experience since the URL they access inside the intranet is the same URL they access from home or when they're on the road.

There are a couple of significant reasons why enterprises want to leverage the claims-aware capabilities of SharePoint. First, they want to collaborate with their business partners through establishing federated relationships. We will not be discussing this here, but once you have the AD FS v2.0 infrastructure in place to support what this chapter discusses, you are only a few steps away from being

able to work with partners to establish federated trust relationships to allow their participation and collaboration on your SharePoint site. This is generally more of a procedural issue than a technical one once your infrastructure is all set up.

Secondly, enterprises have cloud-based or web applications in different domains other than SharePoint where they want to provide web Single Sign-on (SSO) for their end users. Many times this is due to enterprises trying to reduce usernames and passwords across heterogeneous systems, rationalizing end-user access to web applications following a merger or any other number of reasons for helping end users be more efficient in accessing web resources. This is the scenario we will address here, using the Claims Mode authentication method on a SharePoint web application so that AD FS can broker web SSO between SharePoint and an Azure cloud-based web application.

Refer to the “Additional Resources” section at the end of this chapter for guidance on how to configure SharePoint authentication using a SAML security token.

RELYING PARTY APPLICATIONS, FEDERATION PROVIDERS, AND IDENTITY PROVIDERS

Now let’s turn our attention more specifically to the relationships among the players: the RP applications, the FPs, and the IPs.

As mentioned in the conference scenario, the RP applications, the doorkeepers, only needed to trust the badge security token issued by the FP, the registrar. The FP accepted security tokens only from the two IPs it trusted: a state department of transportation and a national government. In this example, you can see the trust relationships clearly played out.

However, if each of the individual RP doorkeepers had the trust relationship with the two IPs, then at the access point to each venue within the conference, the doorkeeper would need to validate the driver’s license or passport to determine whether access should be granted. Clearly, this wouldn’t scale. It’s the same with applications; if every claims-based application needed to manage its own relationships with all the IPs it trusts, then the next application you write would also need to manage all the IP trust relationships. Again, this isn’t a feasible solution for applications within an organization; it would be a manageability nightmare and doesn’t provide the necessary level of trust relationship reuse. Hence, you can see the purpose of the FP STS role.

Rather than each RP application having direct trust relationships with IPs, you inject the FP STS into the mix. Now, the RP application has a single source to trust — only the security tokens the FP provides. The FP then becomes the “trust hub” for all your RP applications, and you can configure it to send each application the specific set of claims on which it relies.

All IP STS trust relationships are then configured with the FP STS. This indirection proves an additional benefit for the RPs trusting the FP. As with the case of the driver’s license and passport, each one could contain a different set of claims, potentially in a different format, based on the state or country, yet the registrar is able to interpret them, use additional resources to determine which ribbons to assign, and then issue a singular badge that can be interpreted precisely by the RP doorkeepers. In the technical implementation, the FP can perform this claims transformation role too.

Various IPs might provide claim values that can be sourced only from their systems. Sometimes these differ in their semantics, and therefore need to be transformed before they are understood by the RP. For example, one IP might provide a supervisor claim, whereas another IP provides a manager claim. The RP, however, needs an administrator claim. It's the FP that manages these discrepancies by accepting both the supervisor claim and the manager claim from the respective IPs, and then applying the transformation logic necessary to supply an administrator claim in the security token it issues to the RP application. Very cool!

Now about this notion of trust. How is trust established so that FPs can trust IPs, and RPs can trust FPs? In the case of establishing trust for WS-Federation, the trust is based upon the public key infrastructure (PKI). Digital certificates are issued by well-known certificate authorities (CAs) like VeriSign and Go Daddy, or organizations may have a CA service of their own as well. An X.509 digital certificate is broadly used across the Internet and is probably best known for enabling SSL. An X.509 certificate has metadata associated with it for identification, but it also has a private-public key pair. When the X.509 certificate is used by its owner to digitally sign or decrypt a document, the owner will always use the private key. Another party can use the public key to validate a digital signature on a document issued by the certificate owner or to encrypt a document destined for the certificate owner.

To apply this to our three players, an STS, whether IP or FP, requests and receives a digital certificate from a CA that uniquely identifies it and then uses the certificate's private key to digitally sign the security tokens it issues. RPs are provided the certificate's public key from their FP STS. The RP knows the security token is authentic if it can validate the digital signature. Likewise, the FP uses the public key of the IP STS to validate the security token it issued and signed with its private key. Granted, there is much more occurring under the hood than space allows discussion of here. If you want to learn more about this, you can start with the resources provided at the end of the chapter. The brief overview provided here should give you a sense of how trust relationships are established and validated.

Relying Party Applications

Our scenario involves two RP applications, SharePoint and an Azure-based web application. For SharePoint, one of its web applications has been configured to trust the Contoso Active Directory Federation Service (AD FS) as its IP. Therefore, it relies upon AD FS to authenticate end users and provide the needed security token.

The Azure-based web application is an RP as well, but it trusts the Windows Azure AppFabric Access Control Service (ACS), described in the next section, to provide its security tokens with the necessary claims.

In both cases, each RP application registers with the respective STS that will be providing its security tokens. Registering an RP application with an STS can either be a manual or automated process, depending on the STS, but, essentially, the STS must be configured so that it knows which claims the RP needs and the service endpoint URL for the RP. This way, the security token can be scoped for use specifically to that RP application. The RP also registers the certificate, with its public key, of the STS and the STS's service endpoint so that the RP can validate that the security token is from the STS it trusts.

Window Azure AppFabric Access Control Service

The Windows Azure AppFabric ACS is a federation provider STS in the cloud. It has the capability to serve as the FP hub for any number of RP applications, whether Azure or non-Azure-based web applications or web services. ACS can also manage claims transformations. Through ACS management, you can configure claims transformation rules to handle the input claims provided by an IP and in turn issue the required output claims for the RP application.

As an FP, ACS can be configured with numerous IPs, such as Windows Live ID, Google, Yahoo!, and Facebook, as well as custom WS-Federation identity providers, such as Microsoft Active Directory Federation Services 2.0, Oracle Identity Federation, CA SiteMinder Federation Security Services, IBM Tivoli Federated Identity Manager, and others. ACS can be configured so that each RP application has only specific IPs associated with it for end-user access. Therefore, although many IPs are trusted by your ACS, you still maintain discretion in terms of how you configure identities to which you'll allow access to each RP application.

Active Directory Federation Services v2.0

AD FS is a Windows-based server product that can serve as an organization's STS. It's included in your Windows Server license, so if you haven't fired it up and started building claims-based applications, you might want to do so. AD FS has been used for many years to provide web SSO across enterprise claims-based web applications, and cross-organization, federated end-user access to RP applications.

As an STS, AD FS can be used in two roles. One, it can serve in the IP role, where it authenticates end-user credentials against Active Directory and issues security tokens with claims to other STSs, such as ACS or another federation partner's STS. Two, it can serve in the FP role, where an organization's claims-based RP applications rely upon it to broker relationships with other IPs, perform claims transformations, and issue security tokens the RP application will trust.

AD FS 2.0 supports both WS-Federation and WS-Trust, but it also supports the standards-based Security Assertion Markup Language (SAML) 2.0 protocol — specifically, its Web Browser SSO Profile and HTTP POST binding. This support for the SAML 2.0 protocol offers greater opportunities to establish cross-organization federations because many organizations use federation STS products built by vendors that implement the SAML 2.0 protocol, and not WS-Federation. This is a great advantage for your claims-based applications; end users from these SAML 2.0-based enterprises can still access your claims-based applications. AD FS brokers the trust relationship and issues a security token to your application for that end user. As a result, you gain cross-domain, cross-platform, cross-organization, and cross-protocol reach for your application!

THE SOLUTION ARCHITECTURE

It is quite common to have several federated web applications to which you want to provide seamless access to your SharePoint end users. In some cases these may be your own enterprise's web applications that are hosted in Windows Azure and are RP applications to your AppFabric ACS. Provided that your SharePoint web application is wired up for claims-based authentication as an RP to AD

FS and AD FS is configured as an identity provider to your ACS, this seamless SSO access for the SharePoint end user is absolutely doable.

Therefore, this solution uses an Azure-hosted, claims-based web application that relies upon ACS as its federation provider STS. ACS has a trust relationship with Contoso AD FS as an IP. Therefore, end users from Contoso who access the web application will be required to first authenticate to Contoso AD FS before they can access the site. One additional twist is that the web application is displayed in a SharePoint Content web part (IFrame), so authentication to the Azure-hosted application should appear seamless to the end user. Figure 11-1 shows the solution architecture.

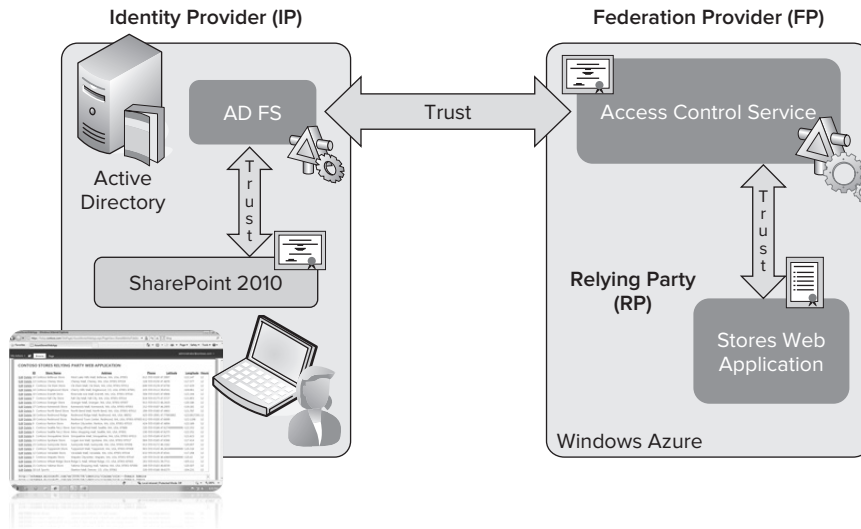


FIGURE 11-1

The following steps describe how this works:

1. The end user accesses a page on SharePoint that hosts an IFrame web part to display the Azure-based web application.
2. The claims-based Azure web application determines that the trusted security token from its FP STS is not present, so it redirects the user to the FP STS for a security token.
3. The FP STS performs *home realm discovery (HRD)* to determine which IP it trusts is the home realm for the end user seeking access to the web application. Once the home realm is identified, the end user is redirected to that IP to authenticate.
4. The IP accepts the end user's credentials and issues a security token with the appropriate claims to the FP STS.
5. The FP STS validates the security token, performs any required claims transformations, and issues a security token to the RP web application.
6. The RP web application validates the security token, extracts the claims, and uses them for site personalization and authorization.
7. The end user uses the Azure-based web application directly from within SharePoint.

ACCESSING CLAIMS-BASED AZURE SERVICES FROM SHAREPOINT

In this section you'll work through the practical steps to set up your ACS namespace, configure the trust between AD FS and ACS, build the RP application, configure the trust relationship between ACS and the RP application, and then seamlessly consume the Azure-based RP application in SharePoint with an SSO end user experience.

Setting Up an Access Control Service Namespace

The AppFabric ACS serves as the federation provider. First, you need to configure it so that you have its endpoints available for the RP application and the identity provider.

1. Log in to `windows.azure.com` and click Service Bus, Access Control & Caching.
2. Click Access Control in the left navigation bar.
3. If you do not already have an ACSv2 namespace, click New Namespace in the ribbon, fill out the dialog, and click Create.
4. When your new namespace becomes active, click it and then click Access Control Service in the ribbon.

Clicking Access Control Service in the ribbon takes you to the new ACSv2 portal where you will do the configuration for identity providers, RP applications, claims transformation rules, and so on. Note that there are a lot of links on the site where you can read more information, so don't hesitate to dig in.

Take a moment to follow the navigation links on the left and become familiar with the site. To call out a few of these links, the Application Integration link gives you quick access to the various URLs that you will need when configuring STSs and RP applications. You can come here to copy these and paste them into configuration settings. The Management service link describes how you can programmatically access parts of the portal so you don't need to always do configurations manually. For our purposes, however, you'll be working in the Trust Relationships section.

Establishing AD FS v2.0 As a Trusted Identity Provider for ACS

Setting up AD FS is beyond the scope of this book, but it is very straightforward to do with v2.0. Resources for installing AD FS v2.0 are noted at the end of the chapter. Assuming that you have your AD FS instance running, you can configure a trust relationship between AD FS and your new ACS. You establish this trust before configuring the trust between the RP application and its trust with ACS because you want to pass through, or transform, claims from the IP to the RP application.

1. In your Access Control Service portal under Trust relationships in the left navigation bar, click Identity providers.
2. Click the Add link and you should see an Add Identity Provider screen like the one shown in Figure 11-2.

Notice there are a number of preconfigured identity providers you can very easily set up to provide identities to access your RP application. Again, this gives your application great reach because these identity providers constitute millions of end users, but your goal is to wire up your AD FS as an IP.

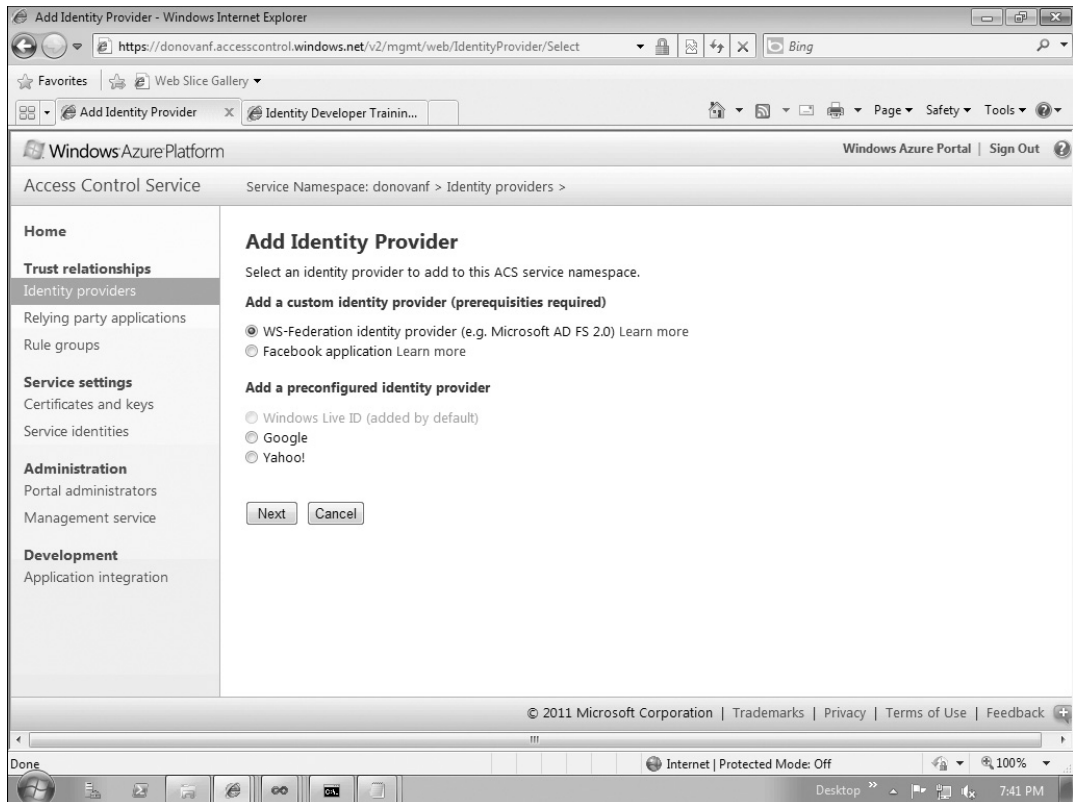


FIGURE 11-2

3. Click WS-Federation identity provider, and then click Next.
4. Enter a display name. This can be any meaningful name.
5. Using the pattern shown in the example URL, use the WS-Federation metadata URL exposed by AD FS: `https://www.contoso.com/FederationMetadata/2007-06/FederationMetadata.xml`. Swap in your AD FS server domain name and paste in the URL. However, if your AD FS server is a test virtual environment that is not exposed to the public Internet, copy and paste the URL for your federation metadata into a new browser window. Access the federation metadata to retrieve the XML. From the browser, save the file as **FederationMetadata.xml** to a convenient location. Then choose the file, click Browse, navigate to the location of the saved file, and select it.
6. For Login link text, enter something like **Login at Contoso**.
7. There is no need for an Image URL or to configure Email domain names, so click Save.

That's it — it couldn't have been easier. Your ACS now trusts your AD FS as an IP.

Because your AD FS server does not yet know about providing tokens to your ACS, you need to configure that side of the relationship too:

1. Open the AD FS v2.0 management console.
2. Expand the Trust Relationships node and click Relying Party Trusts.
3. In the right-hand Action pane, click Add Relying Party Trust to start the wizard.
4. Click Start on the Welcome page.
5. On the Select Data Source page, click “Import data about the relying party published online or on a local network.”
6. Enter the URL for your ACS WS-Federation Metadata endpoint (i.e., `https://yournamespace.accesscontrol.windows.net/FederationMetadata/2007-06/FederationMetadata.xml`). You can find this in your ACS portal by following the link Application integration. Copy and paste this URL into the wizard, and then click Next.
7. On the Specify Display Name page, keep the default display name or change this to something more meaningful to you, and then click Next.
8. On the Choose Issuance Authorization Rules page, choose “Permit all users to access this relying party,” and then click Next.

Sometimes you don't want all users in the organization to have access to an application. Therefore, AD FS allows you to designate which users you want to allow for a given RP application. This is the point in the configuration where you can make that determination.

9. On the Ready to Add Trust page, click through the tabs to see the configuration setting that will be made in AD FS for the ACS relationship, and then click Next.
10. On the Finish page, confirm that the checkbox is checked so the Edit Claim Rules dialog will be opened when you click Close. Click Close.

Notice that when you click the close button, the Edit Claim Rules dialog opens. This is where you configure the claims that you will be releasing to ACS. You do this by adding rules. You can add any number of rules, and AD FS has a claims transformation engine that can perform some very sophisticated transformations if you choose. In this case you will keep it pretty simple.

11. Click Add Rule.
12. On the Select Rule Template page, select Send LDAP Attributes as Claims, and then click Next.
13. On the Configure Rule page, provide a Claim rule name, such as **Claims for ACS**.
14. Select Active Directory as the Attribute store.
15. For Mapping LDAP attributes to outgoing claim types, pair the following selections for LDAP Attribute and Outgoing Claim Type: SAM-Account-Name to Name; Display-Name to Common Name; and Token-Groups-Unqualified Names to Role; and then click Finish.

When completed, your Edit Claim Rules dialog should look like Figure 11-3.

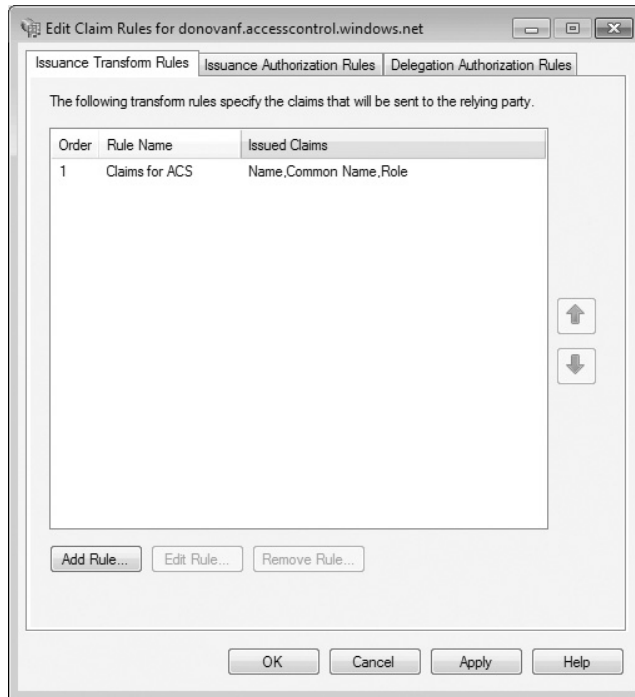


FIGURE 11-3

16. Click Apply and then OK.

Your AD FS is now configured with a new RP trust, for which your ACS is the endpoint. You might be thinking, “I’m confused; I thought my web application was the RP, and ACS was an FP.” You are correct on both counts, but let’s look at the relationships. ACS is an FP. It has relationships with one or more IPs and provides tokens to the applications that rely upon it; but to AD FS, it does not know about your RP application. It only knows about ACS, so ACS is an RP to AD FS. Although ACS is an STS that can transform and issue tokens as well, AD FS only sees it as a relationship that relies on its tokens; hence, to AD FS, ACS is an RP. Federation metadata makes setting up trust relationships very simple.

Creating a Relying Party Azure Web Application

The best way to become familiar with implementing claims-based identity in your solutions is to work through the hands-on labs in the downloadable Identity Developer Training Kit. (See the resources at the end of the chapter.) Download and install the training kit to ensure you have all the prerequisites in place for developing with Windows Identity Foundation (WIF). Here, you will build an Azure-based RP web application, but the training kit covers some of the technical details much more deeply than space allows here, and it provides a `CreateCert.cmd` file for creating a certificate for your RP application.

1. Open Visual Studio 2010 as Administrator.
2. Select File ⇨ New Project.
3. In the New Project dialog, click Cloud and Windows Azure Project, and then provide a name for the project. (*Important:* This name must be unique across *.cloud.net, so you don't want a collision with another web application with the same name on deployment.) Also provide a location for the project. Make sure .NET Framework 4 is selected, check "Create a directory for solution," and then click OK.
4. In the New Windows Azure Project dialog, click ASP.NET Web Role, and then click the center > button.
5. Hover over the Web Role in the right-hand pane, click the pencil icon to rename (e.g., `Federated_StoresApp`), and then click OK.

Now you need to create an SSL certificate that will be used for your RP application.

1. Run Visual Studio Command Prompt (2010) as Administrator.
2. Change the directory path using the `cd` command to the drive with `[IdentityTrainingKitInstallLocation]\Labs\WindowsAzureAndPassiveFederation\Source\Assets`.
3. Type `createcert.cmd yourprojectname` (lowercase is very important for this), where *yourprojectname* is the unique name for your project.
4. When prompted for a password to store with the certificate for accessing the private key, type `abc!123` which is the required password by `CreateCert.cmd`.
5. When prompted to install the certificate, click Yes.

With the certificate created and installed on your machine, you'll now configure the web application:

1. In the Solution Explorer, expand the Role folder, right-click the Web Role inside it (e.g., `Federated_StoresApp`), and select Properties.
2. Click the Certificates tab and choose Add Certificate.
3. Name the certificate the same lowercase name used for your project when creating the certificate.
4. Click the ellipses (...) at the far right of the row to select a certificate.
5. In the certificate selection prompt, select the certificate you just created, and then click OK.
6. Select the Endpoints tab.
7. Select the Endpoint1 row, and then click Remove Endpoint.
8. Click Add Endpoint.
9. For the name, enter `HttpIn`, select the https drop-down, and set the Public Port to 8080.
10. In the drop-down that says <Not set>, click your certificate.
11. Close the [Role] tab and click Yes when prompted to save.

Up to this point, you can consider this a traditional ASP.NET web application that would need to implement its own authentication and identity store. However, this application is intended to rely on claims-based authentication. Therefore, it offloads authentication to an IP and simply reacts to the tokens it receives from a trusted STS. In the following steps, you wire up the claims-based nature of the web application.

1. In the Solution Explorer, under the `Federated_StoresApp`, right-click on `Reference` and choose `Add Reference`.
2. Select the `.NET` tab, choose `Microsoft.IdentityModel`, and then click `OK`.
3. Expand the `References` node and click the `Microsoft.IdentityModel` node. If the `Properties` window is not open, press `F4`.
4. In the `Properties` window, set `Copy Local` to `True`, and `Specific Version` to `False`. This makes the assembly for WIF available to your application when it runs in the cloud.
5. Collapse the `References` node.
6. Open the `Default.aspx` page in `Design` mode.
7. From the `Toolbox`, add a `TextBox` control to the page for displaying claims, which you will enable/disable based on a specific role claim value being present.
8. Open `Default.aspx.cs` and add the following using statements:

```
using Microsoft.IdentityModel.Claims;
using System.Threading;
```

9. Add the following code inside the `Page_Load` method:

```
IClaimsPrincipal icp = Thread.CurrentPrincipal as IClaimsPrincipal;
IClaimsIdentity ici = icp.Identity as IClaimsIdentity;

string displayClaims = "";

foreach (Claim c in ici.Claims)
    displayClaims = displayClaims +
        (c.ClaimType + ":-" + c.Value + "\n");

this.TextBox1.Text = displayClaims;
this.TextBox1.Enabled = false;

//Special permissions for Manager role
if (Thread.CurrentPrincipal.IsInRole("Manager"))
{
    this.TextBox1.Enabled = true;
}
```

The first thing you'll notice is the `IClaimsPrincipal` and `IClaimsIdentity`. Of course, you are used to the `IPrincipal` and the `IIdentity` in ASP.NET from the `HttpContext.Current.User` property, but these are extensions to each. The WIF processes the token and makes the collection of claims available to your application for use. Here, the first task is to get the current principal from the thread and cast it to an `IClaimsPrincipal`. Then the `Identity` is cast to the `IClaimsIdentity`

so that you can access the claims collection for the current user. In this case, you iterate over the claims and concatenate them for display in the TextBox control. This is done to visibly surface the claims passed from the FP STS on the web page.

One of WIF's powerful capabilities is that any role claims passed in a token are automatically added to the ASP.NET Roles infrastructure. Here we're referring specifically to the URI `http://schemas.microsoft.com/ws/2008/06/identity/claim/role`. This claim is generally mapped to an Active Directory security group in AD FS. AD FS then generates a role claim for each security group the end user belongs to and sends these as role claims to the RP. Because of this, you can simply use `.IsInRole()`, which is familiar to ASP.NET developers. For example, users in the Manager role will be able to edit and delete data. If they are not in the Manager role, they will have read-only permissions. Although you are using `IsInRole` functionality to drive what the end user can and cannot do, `IsInRole` was set due to a Manager role claim being present in the token.

There are a couple more details to call out, and here again the Identity Training Kit provides that guidance. Cookie encryption in an Azure-deployed solution needs to be handled differently from a local deployment on the box. Therefore, the following code snippets are needed in your web application:

1. Open `Global.asax.cs` file and add `using` statements so that your code looks like the following:

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Text;
using Microsoft.IdentityModel.Tokens;
using Microsoft.IdentityModel.Web;
using Microsoft.IdentityModel.Web.Configuration;
```

2. Type the `OnServiceConfigurationCreated` method just after the `Application_Start` method and before the `Application_End` method:

```
void OnServiceConfigurationCreated(object sender,
ServiceConfigurationCreatedEventArgs e)
{
    //
    // Use the <serviceCertificate> to protect the cookies that are
    // sent to the client.
    //
    List<CookieTransform> sessionTransforms =
        new List<CookieTransform>(new CookieTransform[] {
            new DeflateCookieTransform(),
            new
RsaEncryptionCookieTransform(e.ServiceConfiguration.ServiceCertificate),
            new
RsaSignatureCookieTransform(e.ServiceConfiguration.ServiceCertificate) });
    SessionSecurityTokenHandler sessionHandler =
new SessionSecurityTokenHandler(sessionTransforms.AsReadOnly());
e.ServiceConfiguration.SecurityTokenHandlers.AddOrReplace(sessionHandler);
}
```

3. In the `Application_Start` method, you need to register the handler for the preceding method. Type the following code in `Application_Start`:


```
FederatedAuthentication.ServiceConfigurationCreated +=
OnServiceConfigurationCreated;
```

4. Build your application and fix any errors. Your `Global.asax.cs` file should look like what is shown in Figure 11-4.

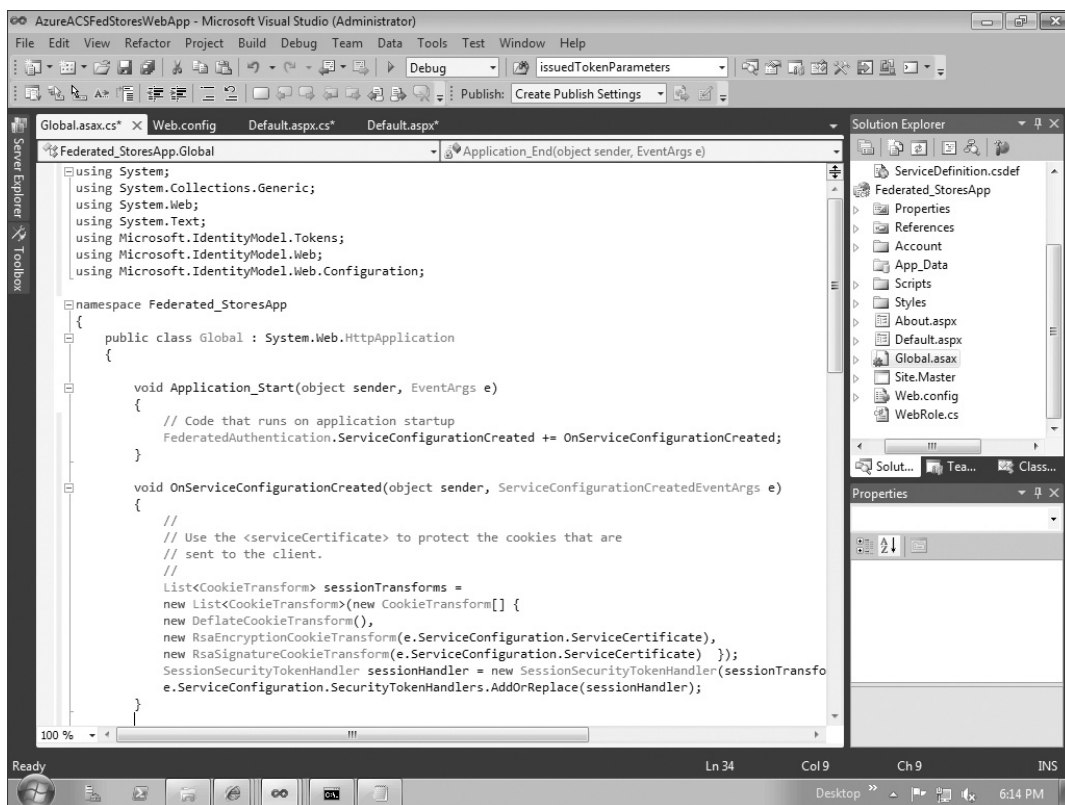


FIGURE 11-4

In this section, you started building your Azure claims-based application, but it's not ready for prime time yet. It's claims-aware because you have referenced the `Microsoft.IdentityModel`, but it is not yet aware of its STS. In the next section, you will wire it up to trust ACS for its security tokens.

Wiring the Azure Web Application to Trust ACS

With your coding complete for the application, the next step is to wire it up to the FP STS that it will trust for its security tokens. In this case, that is your AppFabric ACS.

1. If open, save and close the `Federated_StoresApp web.config` file.
2. In the Solution Explorer, right-click `Federated_StoresApp` and select `Add STS Reference`.

3. On the Welcome to the Federation Utility Wizard, enter `https://yourprojectname.cloudapp.net/` for the Application URI, where *yourprojectname* is the unique name for your Azure Web application, and then click Next.
4. Select Use an existing STS. Return to your ACS portal page and follow the navigation link to Application Integration. Copy the URL for the WS-Federation metadata endpoint and paste it into the STS WS-Federation metadata document location. Click the Test location button to confirm that you can access the `FederationMetadata.xml` document for your ACS.

When the test is successful, a browser will open and display the federation metadata document for the STS.

5. On the STS signing certificate chain validation error page, choose “Disable certificate chain validation,” as this is a development environment scenario. If you had a signing certificate that was issued by a certificate authority (CA) for a production environment, you would want to enable this option. Click Next.
6. On the Security token encryption page, choose No encryption and click Next.
7. On the Offered claims page, you will see that ACS offers two claims. Click Next.
8. Review the Summary page information. If you want to keep this summary information, note that you can only save it by copying the contents and pasting it into a Notepad document. Click Finish and then OK in the confirmation dialog.

The STS Federation Utility configures the `microsoft.identityModel` element in the `web.config` file for `Federated_StoresApp`. Open the `web.config` file and scroll down to the `microsoft.identityModel` element and browse around. The audience URI establishes the scope of the security token; it is valid for this URI only. Note the federated authentication endpoints. The issuer’s endpoint is your ACS FP, and the realm is the URL of your RP application. These are used at runtime to manage the WS-Federation redirects. Only the name and role claim are expected by the RP application. Lastly, note the trusted issuer. This is how trust is validated by the RP application. The only token it will trust is one digitally signed with the private key of the certificate identified with that specific thumbprint.

Because your deployment will be in Azure, you need to make two additions to the `microsoft.identityModel`.

9. Just prior to the `</service>` closing element tag, add the following snippet, where `[yourCorrespondingThumbprint]` is the thumbprint of the certificate you created. You can easily get this by navigating to the Certificates tab of the Web Role described in the previous section and copying the Thumbprint.

```
<serviceCertificate>
  <certificateReference x509FindType="FindByThumbprint"
    findValue="[yourCorrespondingThumbprint]" />
</serviceCertificate>
```

Finally, you need to borrow one more asset from the identity kit to help validate non-encoded XML. Because security tokens are XML, and non-encoded XML is considered a risk for ASP.NET, you need to mitigate this with a special request validation code.

10. Open Windows Explorer and navigate to `[IdentityTrainingKitInstallLocation]\Labs\WindowsAzureAndPassiveFederation\Source\Assets`.
11. Right-click and copy the `SampleRequestValidator.cs` file.
12. In Windows Explorer, navigate to the folder location of your `Federated_StoresApp` project and paste the file.
13. In the Solution Explorer, right-click `Federated_StoresApp` and select **Add** ⇨ **Existing Item** and select `SampleRequestValidator.cs`. Click **Add**.
14. Open the `web.config` file and insert the following snippet just before the `</system.web>` closing element:


```
<httpRuntime requestValidationType="SampleRequestValidator" />
```
15. Save the `web.config` file and build the solution.

Up to this point, you have described the FP STS to the RP application, but you now need to configure ACS to know about the RP application. Here again you use federation metadata to help configure the relationship.

1. Open your ACS portal and under **Trust Relationships**, click the **Relying party applications** link.
2. On the **Relying Party Applications** page, click **Add**.
3. Enter a meaningful name for your RP application, such as **Federated Web Application**.
4. For mode, choose **Import WS-Federation metadata**.
5. Choose either **URL** or **File**, depending on whether the federation metadata can be accessed via the Internet. If you need to use the **File** option, the `FederationMetadata.xml` file for your RP application is in your Visual Studio solution's `FederationMetadata` folder. You can browse to this location and select the file.
6. For the **Token** format, select **SAML 1.1**.
7. For **Authentication Settings**, choose only your AD FS as an **Identity Provider**.
8. Keep the default settings for **Create new rule group** and **Token signing**. Click **Save**.

Your **Relying Party Applications** page should look something like Figure 11-5.

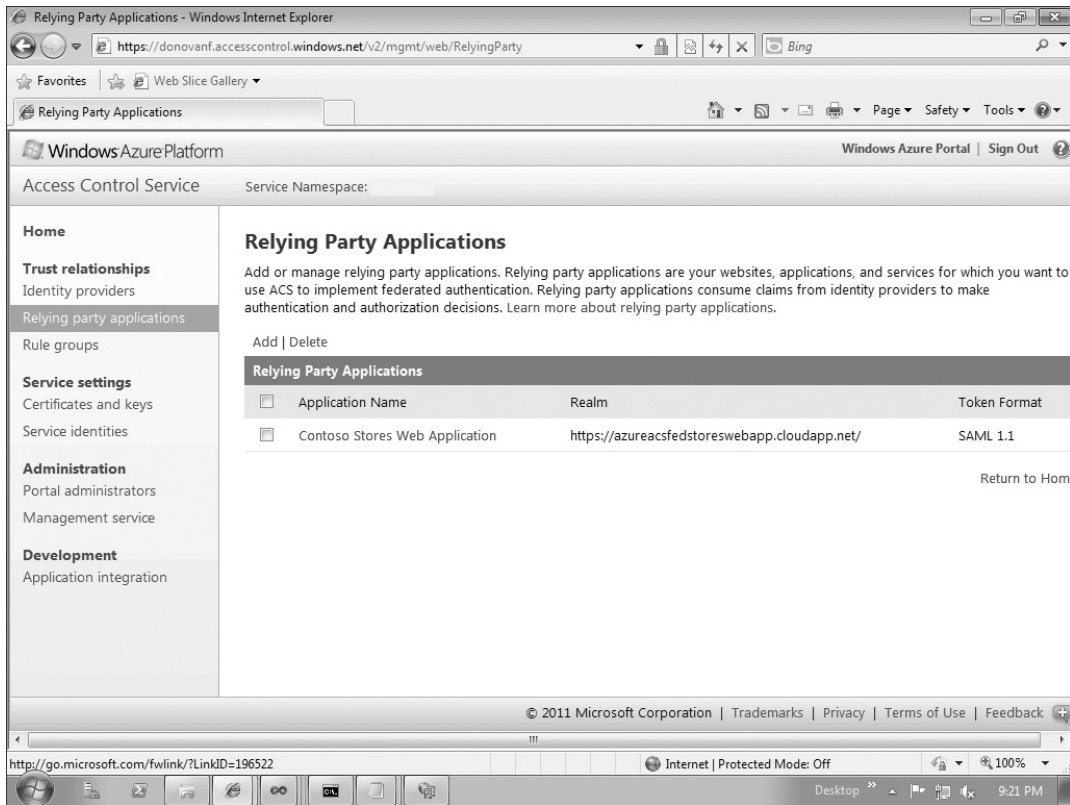


FIGURE 11-5

When you saved the Relying Party Applications page, you had to create a new rule group for the RP. By default, you can create a rule group for every RP application that will rely on ACS for its security tokens. Alternately, if you have a number of RP applications that all rely on the same set of claims being delivered to them, you can configure a rule group that contains this defined set of claims and reuse it across multiple RP applications. That provides you with a great degree of flexibility regarding how you use ACS, but for this example you'll configure a rule group that will be used specifically for your RP application. A rule group consists of the claims that you want to pass through directly from the IP as output claims or, optionally, any specific input claim can be transformed into a different output claim.

1. On the Relying Party Application page, click the link for your newly created RP application.
2. Scroll down to Rule groups and click the link that is the Default Rule Group for your RP application.
3. On the Edit Rule Group page, click Add.

4. For the Claim Issuer Identity Provider, select your AD FS.

Remember the claims you selected as output claims in your AD FS configuration (Name, Common Name, Role)? Here you configure ACS to output these to your RP application.

5. For Input claim type, choose Select type and then select the `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name` from the drop-down list, as shown in Figure 11-6.

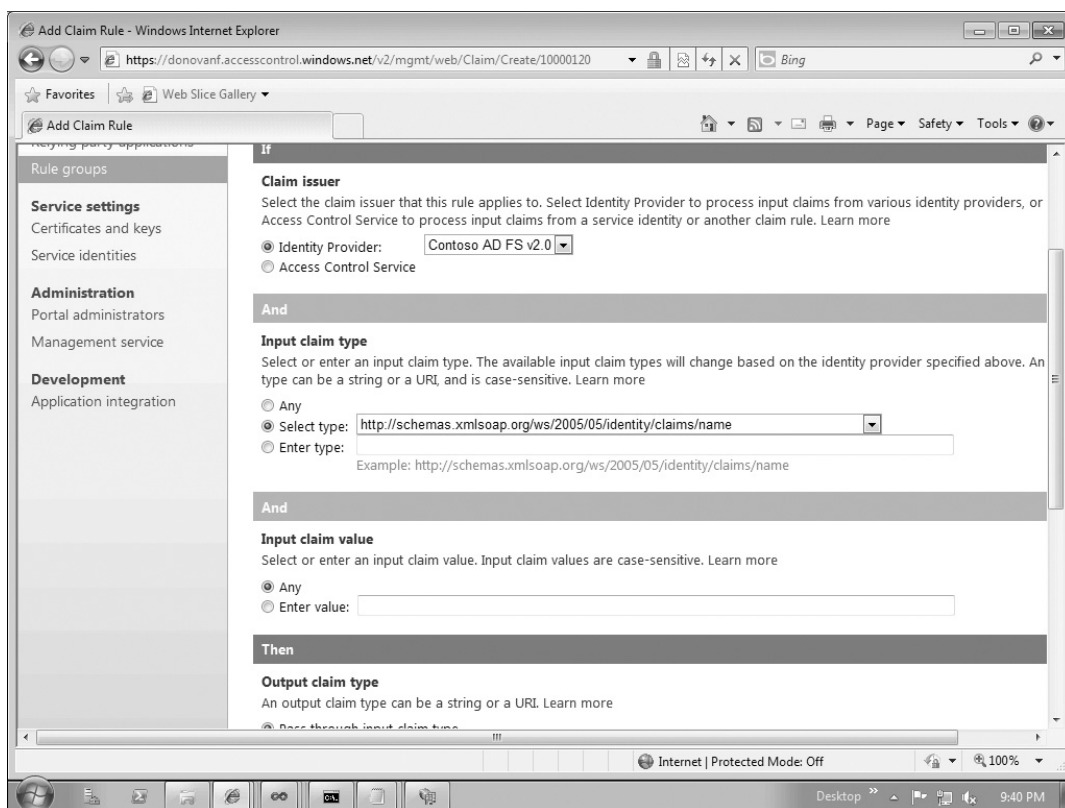


FIGURE 11-6

6. Repeat step 5 for the claims: `http://schemas.xmlsoap.org/claims/CommonName` and `http://schemas.microsoft.com/ws/2008/06/identity/claims/role`.
7. Leave all the remaining default selections at this time so that ACS will pass through the claim and its value from the IP, and then click Save.
8. Click Add again for the Common Name and role claims. Your completed set of rule group claims should look like Figure 11-7.

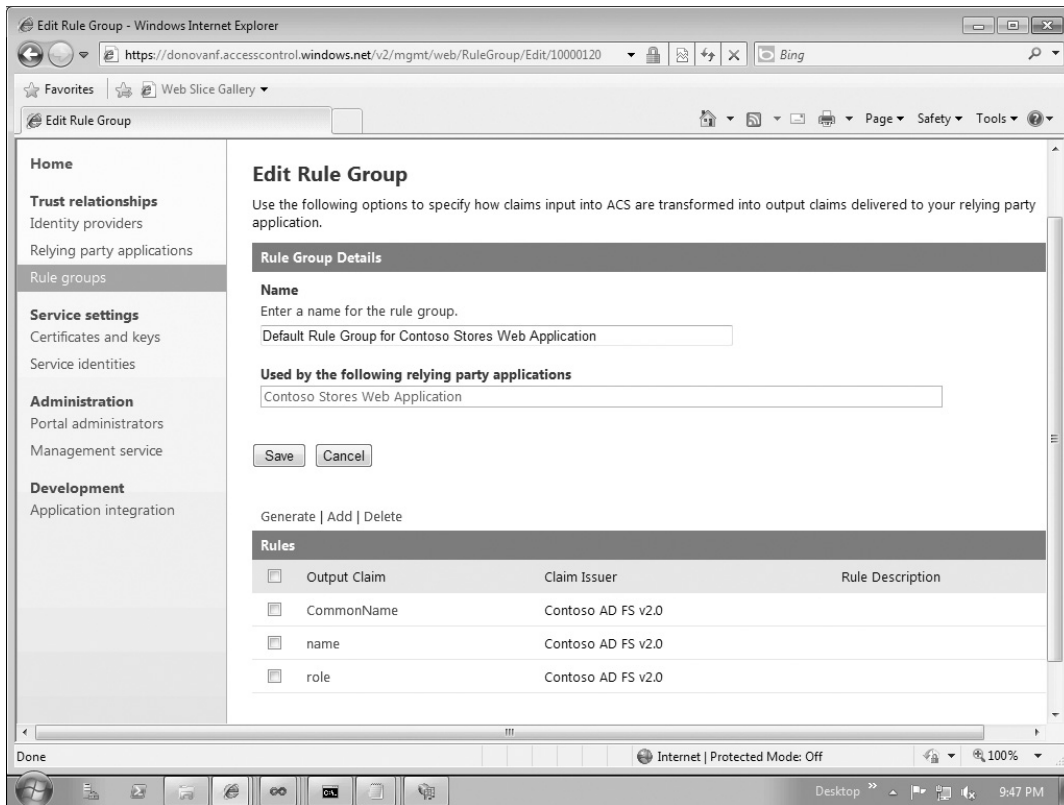


FIGURE 11-7

9. Click Save on the Edit Rule Group page. ACS will use this rule group to provide these claims to the RP application.

Now it's time to test this out. To take a very straightforward approach here, you will deploy directly to Azure production. Of course, this is not the prescribed practice for software development, but it simplifies the deployment by not requiring you to manage URL endpoint changes for your application across the Azure development fabric, staging, and production environments.

1. Open your Azure portal, navigate to your Hosted Services, and click New Hosted Service.
2. In the Pick a Subscription drop-down, select the desired subscription.
3. Enter the name for your service. This should be your project name.
4. Enter the URL prefix. This is your project name, which should be lowercase to match your certificate.
5. Choose a Region to host the service.
6. Select Do not Deploy and then click OK.

Azure will provision a hosted service, although you have not deployed your project to it yet. Once the hosted service is provisioned, you need to add your project's certificate.

1. Click on the certificate folder of your hosted service.
2. In the ribbon, click Add Certificate.
3. In the Certificate dialog, click Browse to locate the certificate you created for your project, enter the password, and then click Create.
4. Open your project in Visual Studio, right-click on your cloud application project, and choose Publish.
5. Select the Create Service Package Only radio button and click OK.
6. When the Publish is complete, a Windows Explorer window will open with .cspkg and .cscfg files present. Right-click on the address bar and choose Copy address as text.
7. In your Azure portal, click on your new hosted service and then click New Production Deployment in the ribbon.
8. Provide a Deployment name and then click the Browse button for the .cspkg and .cscfg files to upload, and click OK. You may receive a warning regarding your instance count. Since you are not concerned about redundancy for your RP at this point, you can click Yes to deploy anyway. Your result should look something like Figure 11-8.

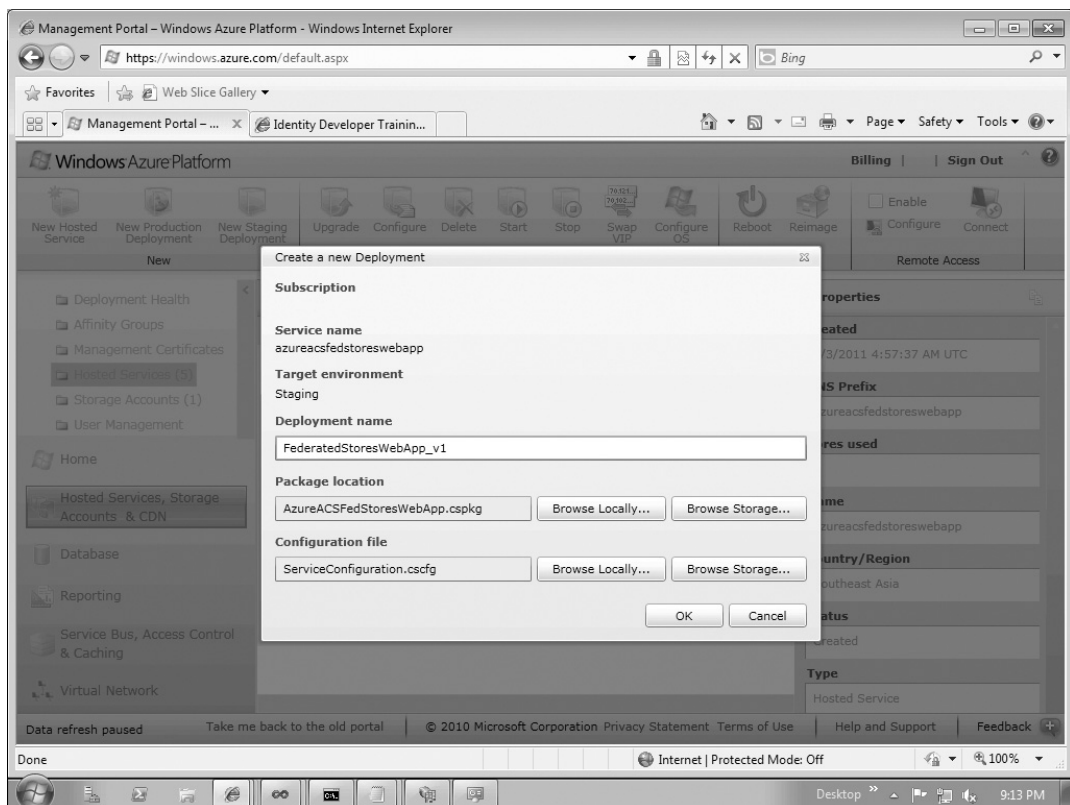


FIGURE 11-8

It may take up to ten minutes or so for Azure to deploy your web application. In addition, because the solution contains a lot of moving parts, with numerous technologies and configuration points, debugging can be tricky if errors occur.

In your browser, type `http://yourprojectname.cloudapp.net:8080/`. The browser will hit your web application and be redirected to ACS for a token. ACS, in turn, will redirect to AD FS for you to authenticate. Authentication at AD FS may happen via Integrated Windows Authentication or forms-based authentication, depending on how it is configured. In any case, upon authentication, AD FS mints a security token and posts it to ACS. ACS validates the security token and mints a new security token, passing through all the claims you configured it to pass, and posts it to your RP web application. The web application validates the token and loads the claims collection in the `IClaimsIdentity`, and your code iterates over this collection, filling the text box with claims for display and setting properties according to role permissions. This is very cool!

When your web page renders, notice that the text box is not enabled; therefore, you cannot make any changes in it. This is because there is no Manager role claim present in the claims collection. In your code above, you added logic that if the end user is in the Manager role, you enabled the `TextBox` control. Providing the Manager role claim can be accomplished by ACS alone.

Recall from earlier in the chapter that an STS can do claims transformations in order to augment the various claims sent to it by different IPs, outputting a single claim set for the RP. You'll create a claims transformation here. ACS will look for the Domain Admins role input claim received from AD FS, and, if it sees it, it will create a Manager role claim to output to the RP application.

1. Open your ACS portal and navigate to the Rule Groups page.
2. Click the link to the Default rule group for your web application.
3. You already created one role claim that simply passes every role claim directly through to the RP application. Click Add.
4. Select your AD FS identity provider as the Claim issuer.
5. For the Input claim type, choose Select type and select the `http://schemas.microsoft.com/ws/2008/06/identity/claims/role` claim from the drop-down list.
6. For the Input claim value, select Enter value and type **Domain Admins**, as this is the value you want to transform to a Manager role claim.
7. For the Output claim type, choose Select type and choose the `http://schemas.microsoft.com/ws/2008/06/identity/claims/role` claim from the drop-down list.
8. For the Output claim value, choose Enter value and type **Manager**.
9. For the Description, type something like **Transform Domain Admins role claim to Manager role claim**.
10. Click Save, and then click Save on the Edit Rule Group page.

What is important to note here is that you are not removing or replacing the Domain Admins role claim; that claim is still passed through to the RP. You are adding a new Manager role claim that ACS is generating due to the presence of the Domain Admins role claim. For another IP it might pass a Supervisor role claim that ACS could transform into

a Manager role claim. Or it can be the presence of any of the other claim types and their values that could be transformed into a Manager role claim. This provides a very powerful capability so the RP application is totally protected from needing to manage this complexity in code. It just expects a token that contains the claims it needs to do its business.

11. Close your browser from the previous access to your web application to clear the cookies. Re-open the browser and navigate to the web application again. Because you need to re-authenticate, a new security token from AD FS and ACS will be created and your new claim will now be present.

Your RP application has just experienced the power of claims-based authentication. The STS, ACS in this case, handled all the heavy lifting to get the end user authenticated with the IP, and then provided the appropriate security token for the RP application. And now the presence of the Manager role claim has enabled the text box in the application. Although this is a simple example, you can think of the far-reaching opportunities for claims to enable authorization permissions to be set in the application. Identity stores are decoupled and the application can trust ACS to provide it with the specific claims it needs. The RP application can stick strictly to meeting its business needs, irrespective of where or how the end user is authenticated.

Accessing the Azure Web Application from SharePoint via Claims-Based Authentication

It is beyond the scope of this chapter to configure a SharePoint web application to use claims-based authentication with AD FS v2.0 as its IP. So, Microsoft TechNet guidance for this is noted at the end of the chapter. However, provided that you have a SharePoint web application in place that is trusting security tokens from AD FS, you can provide your end users an SSO experience with your new Azure-hosted web application with very little effort.

Context-switching takes time, so landing information in front of an end user in a familiar context generally offers a significant productivity gain. Certainly your Azure web application can be accessed independently, but it's much better if it is surfaced in the context of SharePoint, where the end user is already working. Let's wire it up:

1. Open a browser and navigate to your SharePoint claims-based authenticated web application.
2. Click the Site Actions drop-down ⇨ View All Site Content ⇨ Create ⇨ Page ⇨ Web part Page, and then click Create.
3. On the New Web Part Page, provide a name, such as **AzureWebApp**.
4. For the Layout template, choose Full Page, Vertical.
5. For the Save Location, select Site Pages and click Create.
6. On the new web part page just created, click Add a Web Part in the middle of the page.
7. From the displayed Categories, select Media and Content.
8. Under Web Parts, select Page Viewer and click Add.
9. In the newly added Page Viewer web part, click the link to open the tool pane.

10. In the Page Viewer tool pane, replace the `http://` with the URL for your Azure web application, `https://yourprojectname.cloudapp.net:8080/`.
11. Expand the Appearance section and provide a Title for your web part.
12. Set the web part height to 600 pixels.
13. For the Chrome Type, choose Title Only and click OK.

There is one caveat here. Generally, because your SharePoint site is an intranet site, AD FS is configured to use Integrated Windows Authentication as its default. Therefore, the security settings for Local intranet site may be managed via Active Directory Group Policy. Enterprises sometimes do this so that the intranet site is trusted and end-users do not experience an authentication prompt each time they access the SharePoint site. To provide this same end-user SSO experience when your Azure web application is accessed, configure the browser Security settings for Trusted sites with the URL for your application, `https://yourprojectname.cloudapp.net`. You could also manage this through AD Group Policy so that each end user would not need to set this individually. You need to set this security policy for your Azure application to display in the SharePoint IFrame. Figure 11-9 shows an example of a SharePoint-integrated solution.

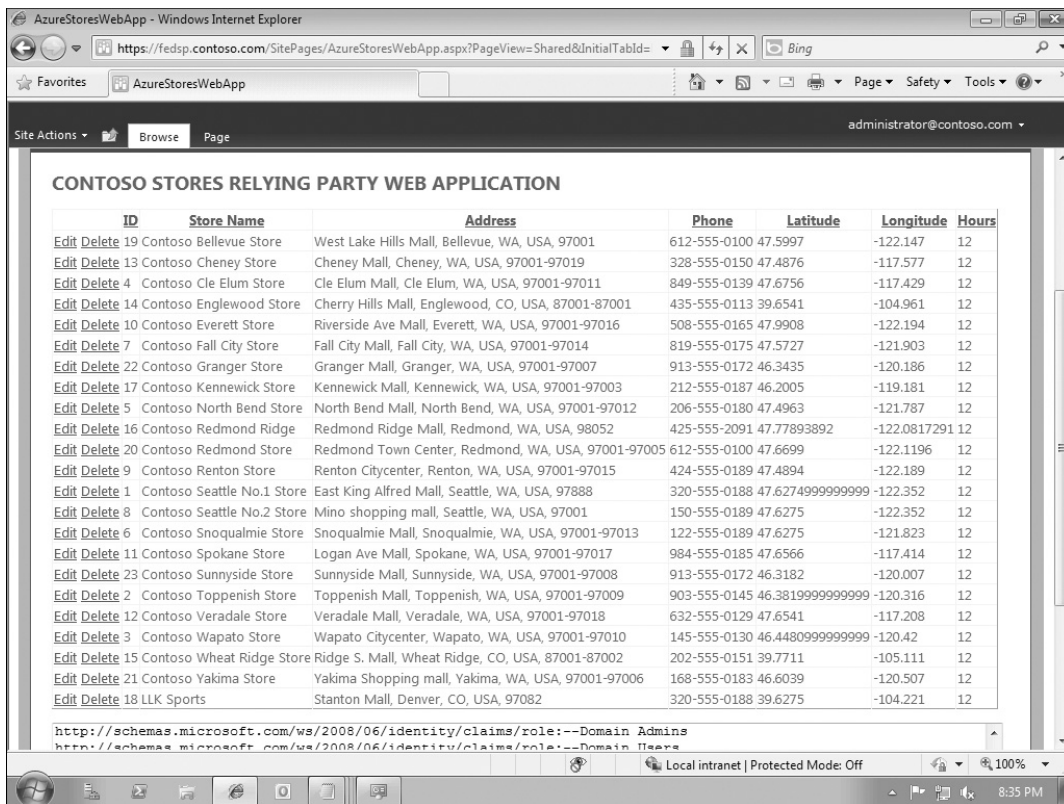


FIGURE 11-9

That's it — you now have a cross-domain, claims-based, Azure-hosted web application seamlessly integrated into your SharePoint site with an SSO end-user experience. There are a lot of moving parts here, but once the infrastructure of AD FS and ACS are in place and the pattern for making use of these is established, a broad array of new business solutions for blending on-premises SharePoint and cloud-based services emerges.

SUMMARY

The open and interoperable protocols associated with claims-based identity provides the way forward for getting identity information securely to and from cloud-based applications and services. On the Microsoft platform, WIF helps developers to bridge the chasm between on-premises and cloud by providing a single identity model to use for building claims-based applications. And AD FS and ACS provide the infrastructure components that enable a seamless SSO experience as an end user navigates across Azure-based RP applications, on-premises SharePoint, or a mash-up of both.

ADDITIONAL REFERENCES

Following are some additional references that you might find useful:

- **Web Services Federation Language (WS-Federation) Version 1.2** — http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html#_Toc223174923
- **Understanding WS-Federation** — <http://msdn.microsoft.com/en-us/library/bb498017.aspx>
- **Windows Azure AppFabric Team blog** — <http://blogs.msdn.com/b/windowsazureappfabric/>
- **Identity Developer Training Kit** — <http://go.microsoft.com/fwlink/?LinkId=148795>
- **Configure authentication using a SAML security token (SharePoint Server 2010)** — <http://technet.microsoft.com/en-us/library/ff607753.aspx>
- **Active Directory Federation Services (AD FS) v2.0** — [http://technet.microsoft.com/en-us/library/adfs2\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/adfs2(ws.10).aspx)

INDEX

A

- Access Control Service (ACS)
 - Windows Azure, 7
 - Windows Azure AppFabric, 4, 315, 316, 318–321, 325–333, 327
- access tokens, 142
- ACS. *See* Access Control Service
- Active Directory, 115
- Active Directory Federation Service (AD FS), 315, 332. *See also* AD FS v2.0
- Active Directory Group Policy, 334
- activity feeds, 118
- AD FS. *See* Active Directory Federation Service
- AD FS v2.0, 7, 8, 313–314, 316, 318–321
- AdaptiveStreamingAzure.exe, 266
- Add New Data Source, 36
- Add New Project, 284, 291
- Add Reference, 15, 191, 285–286
- Add Roles Wizard, 10
- Add Service Reference, 41, 191
- Add Web Site, 162
- .add_applicationReady(), 252
- AddChild, 17
- AddObject, 64–65, 217
- addresses, 199
- ADO.NET, 26, 56
- aggregated solutions, 83–111
 - mashups, 83–94
- aggregators, 84
- AJAX, 147, 273
 - Bing Maps, 176, 178–179, 202
 - SharePoint Online, 266
 - WCF service, 221, 230
- Amazon AWS, 2
- applications
 - Bing Maps, 183–203
 - architecture, 182
 - UI, 186–200
 - creating, 13–19
 - RP, 314–315
 - SharePoint Online, 257–274
 - architecture, 258–260, 279–281
 - dashboard, 281–282
 - HTTP, 270
 - integrated user experience, 266–274
 - My Site, 271
 - social experiences with, 271–274
 - Twitter, 141–152
 - OAuth, 141–149
 - Silverlight, 166–172
 - Windows Azure, claims-based authentication, 333–335
- Application Management, 31, 116
- Application_End, 324
- Application_Start, 324
- architecture
 - Bing Maps, 182
 - claims-based authentication, 316–317
 - Excel, 209–236
 - SharePoint Online, 258–260, 279–281
- AsmxBehavior, 129
- ASP.NET, 145
 - authentication, 323
 - SQL Azure, 26–27
 - Twitter, 143
 - Yahoo Pipes, 90
- ASP.NET Chart, 27, 40
- asynchronous calls, 129–130, 170, 197, 281

authentication. *See also* claims-based authentication
 AD FS, 332
 ASP.NET, 323
 CRM Online, 277
 LinkedIn, 123–124
 SharePoint Online CRM, 281
 SQL Server Authentication, 23
 WCF service, 220
 Windows Authentication, 334
 authorization, 312
 AzureLocalStorageTraceListener, 157

B

bandwidth, 263–264
 BasicHttpBinding, 43
 BCS. *See* Business Connectivity Services
 <behavior>, 29
 behaviors, 129
 BI. *See* business intelligence
 Binary, 211
 ~/bin/debug, 201
 Bing, 5
 Bing Maps, 9, 173–204
 AJAX, 176, 178–179, 202
 applications, 183–203
 architecture, 182
 controls, 178–179
 geo-location web services, 179–180
 JavaScript, 178, 202
 Map Apps, 174–176
 Map Me!, 13–14
 REST, 176, 180, 202
 SDK, 176
 SharePoint Online, 186
 Silverlight, 176, 177–178, 179, 183
 deploying, 201–203
 SOAP, 176, 179, 182
 SPS, 186
 SQL Azure, 186
 SQL Server, 186
 Visual Studio, 183
 Windows Azure AppFabric, 186
 XAML, 176, 186–189

bingRestURIJSON, 180
 bingRestURIXML, 180
 Blob service, Windows Azure, 51, 262
 blobs, 264
 blogs, 118–119, 141
 body, 136, 137
 Bool, 211
 bool, 54
 btnAddStores_Click, 189, 196, 197
 btnClear_Click, 189
 btnFindme, 17
 btnGet, 180
 btnGetStores_Click, 189
 btnZoom_Click, 189
 Business Connectivity Services (BCS), 52
 CRUD, 70
 ECT, 77–81
 SharePoint Online, 186
 SQL Azure, 26, 27
 Windows Azure, 77–81
 YQL, 102
 business intelligence (BI), SQL Azure, 21–49
 dashboards, 27–48
 byte[], 54

C

CallBackUrl, 145
 CAML, 181
 CAs. *See* certificate authorities
 CDNs. *See* content distribution networks
 Central Administration, 116
 certificate authorities (CAs), 315
 Chart Toolkit, 48
 Chart web part, 25–26
 charts, Silverlight, 291–306
 chkboxLatLng, 189
 Choose a Data Model, 36
 Choose a New Data Type, 36
 Choose Model Contents, 36
 claims-based authentication
 architecture, 316–317
 CRM Online, 281

- security, 309–335
 - Windows Azure, 333–335
 - WS-Federation, 311–312
 - WS-Trust, 311–312
- Classic Mode Authentication, 313
- clientaccesspolicy.xml, 157
- ClientContext, 129
- client_GetUserProfileByNameCompleted, 129
- CloudTableClient, 217
- code-based solutions, 26–27
- CodePlex, 23
- colleagues, 113
- ComboBox, 244
- compute emulator, 53–54
- connect, 197
- content distribution networks (CDNs), 260
- Content Editor, 209, 210, 246, 253
- content-type, 137
- context, 63, 64–65
- Control Interactive SDK, 176–177
- controls, Bing Maps, 178–179
- cookies, 324
- CREATE, 24
- Create, 70, 180
- Create a Public View, 45
- Create RSS module, 109
- Create Service Package, 331
- Create View, 45
- CreateCert.cmd, 321
- CreateChildControls, 44
- CreateQuery, 71
- CreateTablesFromModel, 217
- CredentialsProvider, 177
- CRM. *See* customer relationship management
- CRM Online, 276–278
 - claims-based authentication, 281
 - dashboard, 281–282
 - document libraries, 282–283
 - LINQ queries, 289
 - SQL Server tables, 279
 - WCF endpoints, 288
 - Windows Azure proxy, 283–291
- CRMProxyService.svc.cs, 289

- crossdomain.xml, 157
- CRUD, 70, 71, 77
- .csv, 56, 68
- customer relationship management (CRM), 6.
 - See also* CRM Online
 - Microsoft Dynamics CRM, 6, 276–278
 - SharePoint Online, 275–308
 - architecture, 279–281
 - authentication, 281
 - Silverlight grid and chart, 291–306

D

- dashboards
 - BI, SQL Azure, 27–48
 - CRM Online, 281–282
- data, 234
- data binding, 90
- data model, 261–262
- databases, 22, 29–31
- DataContract, 226, 286
- DataGrid, 39
 - <DataGrid>, 299
- DataLayer, 62, 63, 70, 72
- DataMember, 226
- DataRow, 62, 63
- DataServiceContext, 215
- DataTable, 62, 63
- datatables.org, 102
- DateTime, 54, 211
- Default Display Form, 268, 269
- default.aspx.cs, 145
- Delete, 70
- deserialization, 161
- developer key, 178
- development environment, 9–13
- devicemanager.cs, 286
- Dispatcher.BeginInvoke, 197
- .Distinct, 244
- div, 90
- DNS, 165, 230
- _doAjax, 234
- document libraries, 282–283
- Doherty, Roger, 23

double, 54
DSInt, 54

E

Easy Setup Script, 11, 52
ECMAScript, 209
 object model, 245–254
 Windows Azure tables, 210
ECT. *See* external content types
Edit Rule Group, 330
Elements.xml, 302
elements.xml, 44
<enableWebScript />, 29
encryption, cookies, 324
<endpointBehavior>, 29
endpoints, 52, 69–77, 288
entity data model, 48, 56–58
Entity Framework, 56
event receivers, 6, 8
Excel
 architecture, 209–236
 client add-in, 237–245
 financial modeling, 205–254
 Windows Azure tables, 236–254
Excel Web Access web part, 245–254
ExecuteAsyncQuery, 198
ExecuteQuery, 181
ExecuteQueryAsync, 181, 197
Expression Blend, 179, 281
Expression Encoder, 262–263, 264, 266
external content types (ECT), 26, 77–81
external lists, 31–35

F

F12 debugger, 91
Facebook, 4–5
federation provider (FP), 311
feed readers, 84
Feedzilla, 84
Fetch Feed, 85, 87, 93–94
Filter module, 94–97
filters, WCF service, 48

financial modeling, 205–254
 Excel, 236–254
Firewall Rules, 29
flight status service, 5
foreach, 38
Form Web Parts, 268
FP. *See* federation provider
freesshd.com, 53

G

Generate and Publish Scripts Wizard, 23
GenerateReturnDataObject, 38
GeoCode, 161
Geocode Service, 179
geo-location web services, 179–180
GET, 136
GetAccounts(), 289
GetEwa, 252
getHost, 127
getJSON, 91
 .getRangeA1Async(), 253
GetTwitterTrends, 152
GetUserProfileByName, 129
Global.asax.cs, 324, 325
Go Daddy, 315
Google+, 4
Grid, 189
<Grid>, 299
Grid.RowDefinition, 189
grids, Silverlight, 291–306
GUID, 211
Guid, 54, 64

H

high definition (HD), 257, 263
home realm discovery (HRD), 317
Hosted Services, 29, 163
HRD. *See* home realm discovery
HTML, 137, 267
 Bing Maps, 176
 LinkedIn, 121–122
 Yahoo Pipes, 87, 97

HTML Form, 268, 272

HTTP, 142, 270

`HttpContext.Current.User`, 323

`http://localhost`, 145, 146

HTTPS, 270, 281

`HttpRequest`, 153

Huna, Emmanuel, 53

Hyper-V, 9–11, 52

I

IaaS. *See* Infrastructure as a Service

`IClaimsIdentity`, 323, 332

`IClaimsPrincipal`, 323

`ICRMProxyService.cs`, 286

identity providers (IPs), 310, 313,
315, 332

identity store, 312

IDV Solutions, 176

`IIIdentity`, 323

IIS. *See* Internet Information
Services

Imagery Service, 179

images, 124

Import and Export Data Wizard, 23

`Import.aspx.cs`, 59, 60

`ImportData`, 60, 62

`IN.API`, 127

Information Worker Virtual Machine
(IW VM), 9, 52

Infrastructure as a Service (IaaS), 2, 3

`in.js`, 123

`Instances` count, 156

`Int`, 211

`Int32`, 54

`Int64`, 54, 211

intelligence solutions, 8

Internet Explorer, 91

Internet Information Services (IIS), 38–40,
152, 155, 163

`IPrincipal`, 323

IPs. *See* identity providers

IW VM. *See* Information Worker
Virtual Machine

J

JavaScript, 97, 135, 147

Bing Maps, 178, 202

jQuery, 123, 231

Microsoft Dynamics CRM, 6

SharePoint Online, 266–267

Silverlight, 125–126, 127, 134

JavaScript Object Notation (JSON), 102, 136,
137, 148, 180

deserialization, 161

jQuery, 91

Twitter, 147, 161

WCF service, 219–220, 224

Yahoo Pipes, 87–89

JavaScript-based API (JSAPI), 121, 123, 136

Johnson, Chris, 11

jQuery, 91, 147

AJAX, 273

JavaScript, 123, 231

Microsoft Dynamics CRM, 6

SharePoint Online, 266, 271

web parts, 89–90, 100

`jquery.js`, 231

`jquery-templates.js`, 231

JSAPI. *See* JavaScript-based API

JSON. *See* JavaScript Object Notation

`JSON.stringify`, 136

`jTemplates`, 231

K

Key Performance Indicators (KPIs), 26

L

Latitude, 17

line-of-business (LOB), 4, 275

Link, 109

LinkedIn, 8, 120–138

authentication, 123–124

HTML, 121–122

JSAPI, 136

profiles, 121, 124–128

- Silverlight web parts, 121–122
- status messages, 133–137
- Visual Studio, 122, 124
- Web 2.0, 4–5
- LinkedIn Developer Network, 121
- LINQ, 26, 71, 217, 244, 289
- LINQ to XML, 147
- lists, 31–35, 200
- ListItemCreationInformation, 198
- listOfQueries, 170
- Loaded, 300
- LoadProfile, 128
- LOB. *See* line-of-business
- Location, 17, 189, 198
- locationFilter, 198
- long, 54
- Longitude, 17

M

- MainMap, 189, 198
- MainPage, 300
- MainPage.xaml, 166, 186, 189, 298
 - View Code, 168, 192
- MainPage.xaml.cs, 17, 125, 192, 300, 304
- Manage Service Applications, 31, 116
- Managed Metadata, 52
- Map Apps, Bing Maps, 174–176
- Map Me!, 13–19
- MapLayer, 189
- Mapping LDAP, 320
- mashups
 - aggregated solutions, 83–94
 - Yahoo Pipes, 84–94
- MessageBox.Show, 180
- metadata, 52, 262, 326
- method, 234
- Microsoft Dynamics CRM, 6, 276–278
- Microsoft Office. *See* Office
- Microsoft Query. *See* Query
- Microsoft Sync Framework 2.1, 23
- Microsoft.Client.Silverlight.dll, 293
- Microsoft.Maps.MapControl, 17, 177
- Microsoft.SharePoint.Client.
 - Silverlight.Runtime.dll, 293

- Microsoft.WindowsAzure.StorageClient, 239
- middleware, 4
- mirroring, 22
- Model-View-ViewModel (M-V-VM), 300
- My AppID, 31
- My Site, 114–115, 129, 133, 271
- myprxy, 168
- myWCFProxy, 43

N

- native environment, 11–13
- .NET Framework, 208, 214, 242, 277
- New Project, 238, 283
- New Query, 30
- New Storage Account, 211
- New Windows Azure Project wizard, 155
- Nintex Live, 8
- no-code solutions, SQL Azure, 25–26
- notes, 117–118

O

- OAuth. *See* Open Authorization
- object, 123
- ObservableCollection, 129
- ODBC. *See* Open Database Connectivity
- Office
 - add-ins, 208
 - Silverlight UI, 208–209
- Office 365, 23, 48, 100, 260
 - Bing Maps, 179
 - SharePoint Online, 258
 - Windows Azure, 207–209
- Office Services, 209
- OLE DB, 206
- onLinkedInAuth, 126
- onLinkedInLoad, 123
- OnLoad, 43
- OnServiceConfigurationCreated, 324
- OnStart, 65
- Open Authorization (OAuth), 7, 121, 141–149
- open data tables, 102–104, 111
- Open Database Connectivity (ODBC), 22, 206

OpenID, 7

OperationContract, 286

Operations Designer, 77

Operators, 93, 94

P

PaaS. *See* Platform as a Service

PartitionKey, 54, 64, 214, 225

 TableServiceEntity, 58

 Windows Azure, 63, 212

passwords, 142

paths, 264

person-activities, 136

Pipe Output, 87, 93–94

Pipes editor, 85–87

PKI. *See* public key infrastructure

Platform as a Service (PaaS),

 2, 3

pluginLoaded, 127

policies, 7

POST, 136

postLinkedInStatus, 135

primary keys, 22

private containers, 264

Private Documents library, 114

private keys, 315

private-public key pairs, 315

profiles

 LinkedIn, 121, 124–128

 properties, 116–117, 124–133

 social computing, 115–117

programmableweb.com, 101

Project Wizard, 238

PropertyConstants, 135

PropertyData, 129

proxy, WCF, 279–280,

 283–291, 292

public access, blobs, 264

public key infrastructure (PKI), 315

PushPin, 17

Pushpin, 198

PushPinLayer, 17

PUT, 136

PuTTY, 53

Q

queries. *See also* jQuery; Yahoo Query Language

 LINQ, 217, 244, 289

 SQL Azure, 22, 24

 WCF service, 35

 YQL, 102

Query, 30, 206

R

range.setValuesAsync(), 253

Raw, 136

ReadItem, 70

ReadList, 70

Really Simple Syndication (RSS), 84, 85, 87, 109, 110

regulations, 7

relational database, 22

relying party (RP), 311, 314–315, 321–325, 327

Relying Party Application, 327

request tokens, 142

resource STS (R-STs), 311

responseString, 180

REST, 5, 7, 121, 136, 148, 180

 Bing Maps, 176, 202

 CRUD, 71

 GeoCode, 161

 OAuth, 121, 141

 Twitter, 141, 147

 WCF service, 152

 Windows Azure Table service, 71

 Yahoo Pipes, 100

 YQL queries, 102

resultData, 170

Route Service, 179

RowKey, 54, 58, 214, 225

 Guid, 64

 Windows Azure, 212

RP. *See* relying party

RSS. *See* Really Simple Syndication

R-STs. *See* resource STS

S

SaaS. *See* Software as a Service

sales tracker, 8

- SalesForce.com, 23
- SAML. *See* Security Assertion Markup Language
- sandboxes, 300
- SaveChanges, 63, 217
- scalability
 - SQL Azure, 21–22
 - Windows Azure, 3, 63
 - SharePoint Online, 279
 - storage, 260
- script, 123
- SDK
 - Bing, 5
 - Bing Maps, 176
 - Control Interactive SDK, 176–177
 - Microsoft Sync Framework 2.1, 23
 - Windows Azure, 52
- Search Service, 179
- Secure Shell (SSH), 52–53
- security, 7
 - claims-based authentication, 309–335
 - SharePoint Online CRM, 281
- Security Assertion Markup Language (SAML),
 - 141, 142, 313, 314, 316
- security tokens, 310, 314, 315, 327
- security token service (STS), 311, 313, 315, 326, 332
- SELECT, 24
- Select, 217
- SELECT *, 181
- Select Rule Template, 320
- SelectionChanged, 299
- semantic Web, 4
- Send LDAP Attributes as Claims, 320
- Server Explorer, 212, 219
- Server Roles, 10
- </service>, 326
- ServiceConfiguration.cscfg, 65–66
- ServiceDefinition.csdef, 65, 67
- Set Object Permissions, 33
- SetGeoCode, 161
- SetView, 198
- Shared Documents, 114
- SharePoint client object model (SP COM), 181
- SharePoint Designer (SPD), 77
- SharePoint Online, 152
- AJAX, 266
- application, 257–274
 - architecture, 258–260, 279–281
 - dashboard, 281–282
 - HTTP, 270
 - integrated user experience, 266–274
 - My Site, 271
 - social experiences with, 271–274
- BCS, 186
- Bing Maps, 179, 186
- CRM, 275–308
 - architecture, 279–281
 - authentication, 281
 - security, 281
 - Silverlight grid and chart, 291–306
- data model, 261–262
- HTML, 267
- JavaScript, 266–267
- jQuery, 266, 271
- Office 365, 258
- platform, 258–259
- Silverlight, 279, 280–281
- Site Collection, 261
- SQL Azure, 186
- UI, 305
- WCF proxy, 279–280
- web parts, 279
- Windows Azure, 165, 258
 - scalability, 279
- SharePoint Server (SPS), 152, 165, 186
- SharePoint Solution Package, 181
- Show ME, 176–177
- ShowItems, 197
- Silverlight, 123, 136
 - Add Reference, 15, 191
 - asynchronous calls, 129–130, 197, 281
 - Bing Maps, 176, 177–178, 179, 183, 201–203
 - chart, 291–306
 - Chart Toolkit, 48
 - Control Interactive SDK, 176–177
 - Expression Blend, 281
 - Expression Encoder, 262–263
 - grid, 291–306
 - JavaScript, 125–126, 127, 134

- JSON, 148
- lists, 200
- Microsoft Dynamics CRM, 6
- SharePoint Online, 279, 280–281
- SQL Azure, 26–27
- Twitter, 166–172
- UI, Office, 208–209
- WCF service, 291
- web parts, 121–122, 152, 280, 302
- XAML, 15, 124, 128–129
- XML, 127
- Silverlight Toolkit, 293
- Single Sign-On (SSO), 314
- site collections, 115, 129, 261
- Site.css, 59
- Site.Master, 59
- siteUrl, 198
- smooth streaming, 263
- SOAP, 5, 176, 179, 182, 277
- social computing, 113–138
 - activity feeds, 118
 - blogs, 118–119
 - My Site, 114–115
 - notes, 117–118
 - profiles, 115–117
 - tagging, 117–118
 - wikis, 119–120
- Social Data Service, 271, 273
- Software as a Service (SaaS), 3, 23, 260, 275
- Solution Explorer, 239, 243, 284, 292, 302
- Sort module, 94–97
- Source Code, 176–177
- Source Editor, 268, 272
- SP COM. *See* SharePoint client object model
- SPD. *See* SharePoint Designer
- spjs-utility.js, 269
- SPLinkedIn.ascx, 126, 127
- SPS. *See* SharePoint Server
- SPServices, 271
- SPS-StatusNotes, 135
- SQL Azure, 4
 - BI, 21–49
 - dashboards, 27–48
 - Bing Maps, 186
 - code-based solutions, 26–27
 - ASP.NET, 27
 - databases, 29–31
 - external lists, 31–35
 - intelligence solutions, 8
 - no-code solutions, 25–26
 - BCS, 27
 - scalability, 21–22
 - SharePoint Online, 186
 - SQL Server, 22–23
 - SQL Server 2008 R2 Management Studio, 22, 23–25, 30
 - storage, 22
 - WCF service, 34
- SQL Azure Migration Wizard, 23
- SQL Server, 22–23, 186
 - Excel, 206
 - tables, CRM Online, 279
- SQL Server 2008 R2 Management Studio, 22, 23–25, 30
- SQL Server Authentication, 23
- SQL Server Integration Services (SSIS), 23
- SQL Server Reporting Services (SSRS), 26
- SqlDataAdapter, 26
- SSH. *See* Secure Shell
- SSIS. *See* SQL Server Integration Services
- SSO. *See* Single Sign-On
- SSRS. *See* SQL Server Reporting Services
- status messages, 133–137
- statusMessage, 136
- StatusNotes, 135
- storage
 - emulator, 53–54
 - SQL Azure, 22
 - Windows Azure, 4, 65
 - scalability, 260
 - tables, 54–68, 71–77
- Storage Accounts, 29, 163, 211
- StorageConnectionString, 67
- String, 54, 211
- STS. *See* security token service
- STS Federation Utility, 326
- sub-selects, 102
- System.Collections.dll, 41

System.Data.DataTable, 60
 System.Data.Services.Client,
 56, 239
 System.Drawing.dll, 41
 <system.serviceModel>, 29
 System.Web.DataVisualization, 26
 System.Web.DataVisualization.dll,
 40, 44
 System.Web.UI.DataVisualization.
 Charting.dll, 41
 System.Windows.Controls.Data.dll, 293
 System.Windows.Controls.Data
 .Input, 166, 186
 System.Windows.Controls.
 DataVisualization.Toolkit.dll, 293

T

tables

- open data tables, 102-104, 111
- scalability, 63
- SQL Azure, 22
- SQL Server, 279
- sub-selects, 102
- Windows Azure, 206-207
 - ADO.NET Entity Framework, 56
 - DNS, 230
 - ECMAScript, 210
 - Excel financial modeling,
 236-254
 - populating, 58-68
 - storage, 54-68, 71-77
 - WCF Data Services, 209-236
- Table service, Windows Azure, 51, 71
- TableName, 67
- TableServiceContext, 63,
 64, 215
- TableServiceEntity, 58, 214
- tag cloud, 117
- tagging, 117-118
- Target Application Settings, 31
- Target Framework,
 214, 242
- Test Connection, 36
- Text Editor, 268, 272

- textboxes, 199
- third-party libraries, 141
- Timestamp, 54, 58
- tokens, 142, 146
 - security tokens, 310, 314, 315, 327
- transactional database, 22
- Transact-SQL (T-SQL), 22, 23, 24
- trend reports, Twitter, 8, 149-152
- T-SQL. *See* Transact-SQL
- Twitter, 139-172
 - applications, 141-152
 - OAuth, 141-149
 - ASP.NET, 143
 - blogs, 141
 - integration of, 152-172
 - JSON, 161
 - passwords, 142
 - REST, 141, 147
 - Silverlight, 166-172
 - third-party libraries, 141
 - tokens, 146
 - trend reports, 8, 149-152
 - username, 142
 - Visual Studio, 145, 153-154
 - WCF service, 153
 - Web 2.0, 4-5
 - WOEID, 150-152

U

- UIs (user interfaces)
 - Bing Maps, 186-200
 - Office, 208-209
 - SharePoint Online, 305
 - Silverlight, 208-209
- Union module, 93, 110
- Update, 70
- uploads, 262-266
- URIs, 161, 180
- UseDevelopmentStorage=true, 67
- User Profile Service, 52, 116
- <UserControl>, 298
- username, 142
- userprofileservice.asmx, 120
- using, 324

V

variable bitrate resolution (VBR), 262
 VeriSign, 315
 VHD. *See* Virtual Hard Drive
 View Access Keys, 265
 View All Site Content, 45
 View Code, 168, 192
 View in Designer, 166
 Virtual Hard Drive (VHD), 52
 virtual machines (VMs), 2, 9–11
 Visio, 209
 Visual Studio, 15, 55, 76
 Add New Project, 291
 Bing Maps, 179, 183
 ECT, 77
 LinkedIn, 122, 124
 New Project, 238, 283
 Office, 208
 Server Explorer, 212, 219
 Solution Explorer, 285, 292
 Twitter, 145, 153–154
 WPF Designer, 298
 Visual Studio Tools for Office (VSTO),
 238, 246
 VMs. *See* virtual machines
 VSTO. *See* Visual Studio for Office

W

WCF. *See* Windows Communication
 Foundation
 WCF Service Web Role, 155, 284
 Web 2.0, 4–5
 Web 3.0, 4
 web parts, 8
 Content Editor, 253
 HTML Form, 268, 272
 jQuery, 89–90, 100
 Microsoft Dynamics CRM, 6
 Office 365, 100
 SharePoint Online, 279
 Silverlight, 152, 280, 302
 LinkedIn, 121–122
 SQL Azure, 26

WCF service, 38, 40–46
 Yahoo Pipes, 84, 89
 YQL, 109–110
 Web Part menu, 268, 272
 WebClient, 161
 web.config, 29, 45, 288
 WebRole, 65, 157
 Where On Earth ID (WOEID), 150–152, 161
 Whitley, Shannon, 144–145
 WIF. *See* Windows Identity Federation; Windows
 Identity Framework
 wikis, 119–120
 Windows Authentication, 334
 Windows Azure, 2, 3–4
 ACS, 7
 ADFS 2.0, 8
 application
 claims-based authentication, 333–335
 RP, 321–325
 Windows Azure AppFabric ACS, 325–333
 BCS, 77–81
 Bing Maps, 179
 Blob service, 51, 262
 building services, 51–81
 compute emulator, 53–54
 endpoints, 52
 WCF service, 69–77
 entity data model, 56–58
 financial modeling, 205–254
 GetTwitterTrends, 152
 Hyper-V, 52
 IIS, 155, 163
 ImportData, 60, 62
 .NET Framework, 208
 Office 365, 207–209
 Office Services, 209
 PartitionKey, 63, 212
 RowKey, 212
 scalability, 63
 SharePoint Online, 279
 storage, 260
 SDK, 52
 SharePoint Online, 165, 258
 scalability, 279
 SPS, 165

- SSH, 52–53
 - storage, 4, 65
 - scalability, 260
 - Table service, 51
 - REST, 71
 - tables, 206–207
 - ADO.NET Entity Framework, 56
 - DNS, 230
 - ECMAScript, 210
 - Excel financial modeling, 236–254
 - populating, 58–68
 - storage, 54–68, 71–77
 - WCF Data Services, 209–236
 - uploads, 262–266
 - WCF Data Services, 236
 - WCF proxy, CRM Online, 283–291
 - WCF service, 28, 51, 163, 248
 - endpoints, 69–77
 - web role, WCF service, 55–56
 - Windows Azure AppFabric, 4
 - ACS, 315, 316, 318–321
 - RP, 327
 - Windows Azure application, 325–333
 - Bing Maps, 186
 - Windows Azure Marketplace DataMarket, 4, 5
 - Windows Azure Platform Management Portal, 265
 - Windows Communication Foundation (WCF), 129
 - Data Services, 26
 - DataServiceContext, 215
 - Office, 209
 - Windows Azure, 209–236
 - endpoints, CRM Online, 288
 - proxy, 292
 - SharePoint Online, 279–280
 - Windows Azure, 283–291
 - REST, 180
 - service, 7
 - Add Web Site, 162
 - AJAX, 221, 230
 - ASP.NET Chart, 27, 40
 - asynchronous calls, 170
 - authentication, 220
 - Content Editor, 210
 - creating, 35–40
 - CRUD, 77
 - DNS, 165
 - ECMAScript object model, 248
 - filters, 48
 - IIS, 38–40
 - JSON, 219–220, 224
 - Office 365, 48
 - queries, 35
 - REST, 152
 - Silverlight, 291
 - SQL Azure, 26–27, 34
 - Twitter, 153
 - web parts, 38, 40–46
 - Windows Azure, 28, 51, 55–56, 69–77, 163, 248
 - Windows Forms, 39, 180, 208
 - Windows Identity Federation (WIF), 7
 - Windows Identity Foundation, 141
 - Windows Identity Framework (WIF), 281
 - Windows Presentation Foundation (WPF), 208
 - Windows Server 2008 R2, 9–11
 - WOEID. *See* Where On Earth ID
 - workflows, 8
 - WPF. *See* Windows Presentation Foundation
 - WPF Designer, 298
 - WS-Federation, 311–312, 316
 - .wsp, 305
 - WS-Trust, 311–312, 316
- X**
- X.509 certificate, 315
 - XAML
 - Bing Maps, 176, 186–189
 - data binding, 90
 - <Grid>, 299
 - jQuery, 90
 - MainPage.xaml, 189, 298
 - Silverlight, 15, 124, 128–129
 - .xap, 201
 - XAPModule, 302
 - XML, 26

- deserialization, 161
- LINQ to XML, 147
- REST, 136, 148
- security tokens, 327
- Silverlight, 127
- Twitter, 147
- YQL, 102

XRM, 6

Y

Yahoo Pipes

- consuming pipes, 89–92
- creating pipes, 87–89
- Filter module, 94–97
- mashups, 84–94
- multiple feed sources, 92–94
- Pipes editor, 85–87
- Sort module, 94–97

- using existing pipes, 97–101
- web parts, 84, 89
- YQL, 101–110

Yahoo Query Language (YQL), 84, 85

- BCS, 102
- console, 102–109
- JSON, 102
- queries, REST, 102
- RSS, 109
- sub-selects, 102
- web parts, 109–110
- XML, 102
- Yahoo Pipes, 101–110

YouTube, 270

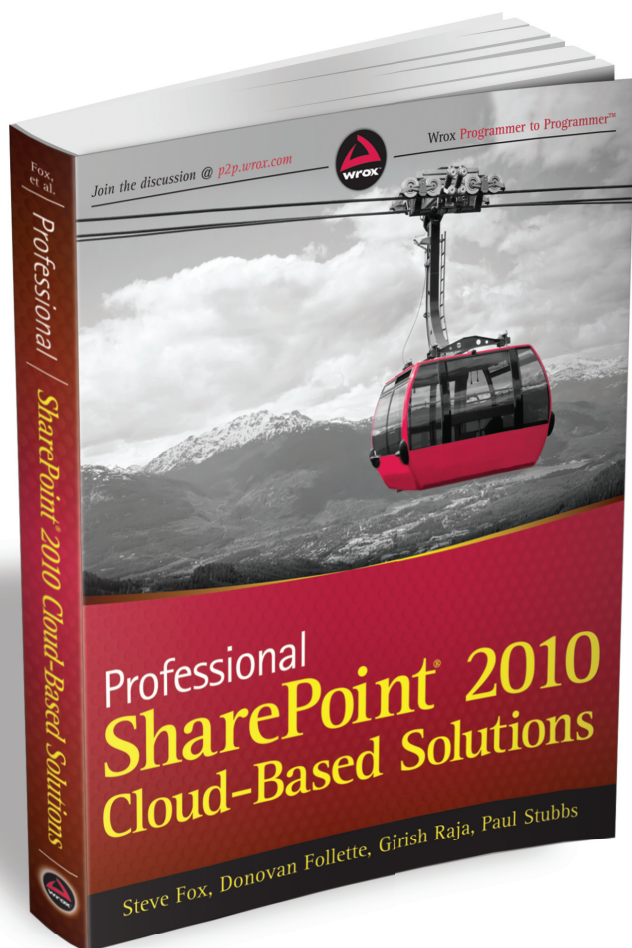
YQL. *See* Yahoo Query Language

Z

Zoom In, 202

Try Safari Books Online FREE for 15 days + 15% off for up to 12 Months*

Read this book for free online—along with thousands of others—
with this 15-day trial offer.



With Safari Books Online, you can experience searchable, unlimited access to thousands of technology, digital media and professional development books and videos from dozens of leading publishers. With one low monthly or yearly subscription price, you get:

- Access to hundreds of expert-led instructional videos on today's hottest topics.
- Sample code to help accelerate a wide variety of software projects
- Robust organizing features including favorites, highlights, tags, notes, mash-ups and more
- Mobile access using any device with a browser
- Rough Cuts pre-published manuscripts

START YOUR FREE TRIAL TODAY!

Visit www.safaribooksonline.com/wrox17 to get started.

*Available to new subscribers only. Discount applies to the Safari Library and is valid for first 12 consecutive monthly billing cycles. Safari Library is not available in all countries.



An Imprint of **WILEY**
Now you know.

Safari 
Books Online

www.it-ebooks.info



Programmer to Programmer™

Connect with Wrox.

Participate

Take an active role online by participating in our P2P forums @ p2p.wrox.com

Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble

Join the Community

Sign up for our free monthly newsletter at newsletter.wrox.com

Wrox.com

Browse the vast selection of Wrox titles, e-books, and blogs and find exactly what you need

User Group Program

Become a member and take advantage of all the benefits

Wrox on **twitter**

Follow @wrox on Twitter and be in the know on the latest news in the world of Wrox

Wrox on **facebook**

Join the Wrox Facebook page at facebook.com/wroxpress and get updates on new books and publications as well as upcoming programmer conferences and user group events

Contact Us.

We love feedback! Have a book idea? Need community support?
Let us know by e-mailing wrox-partnerwithus@wrox.com

Related Wrox Books

Professional Workflow in SharePoint 2010: Real World Business Workflow Solutions

ISBN: 978-0-470-61788-5

With the new Workflow Foundation 4 (WF4) toolkit in SharePoint 2010, companies have new ways to build custom solutions for common or frequent business processes. This unique book is packed with instructions and tips that show you how. You'll use WF4 to create and implement office-practical apps such as expense report approvals, RFPs, sale pipeline management, and more. The book also covers how to design custom activities with SharePoint Designer 2010.

Professional Business Connectivity Services in SharePoint 2010

ISBN: 978-0-470-61790-8

With this in-depth guide, a team of SharePoint experts walks you through the features of the new BCS, including the ability for users to view and modify the data from SharePoint 2010 with BCS. You'll explore how to use BCS, deploy solutions, create external content types and lists, create .NET host connectors, and more.

Professional SharePoint 2010 Development

ISBN: 978-0-470-52942-3

Professional SharePoint 2010 Development offers an extensive selection of field-tested best practices that shows you how to leverage the vast power of this multi-faceted tool to build custom workflow and content management applications.

Professional SharePoint 2010 Branding and User Interface Design

ISBN: 978-0-470-58464-4

Creating a branded SharePoint site involves understanding both traditional web design techniques as well as topics that are typically reserved for developers. This book bridges that gap by not only providing expert guidance for creating beautiful public facing and internal intranet sites but it also addresses the needs of those readers that only want to understand the basics enough to apply some style to their sites.

Professional SharePoint 2010 Administration

ISBN: 978-0-470-53333-8

SharePoint 2010 boasts a variety of incredible features that will challenge even the most experienced administrator who is upgrading from SharePoint 2007. Written by a team of SharePoint experts, this book takes aim at showing you how to make these new features work right for you.

Real World SharePoint 2010: Indispensable Experiences from 22 MVPs

ISBN: 978-0-470-59713-2

This book is a must-have anthology of current best practices for SharePoint 2010 from over 20 of the top SharePoint MVPs. They offer insider advice on everything from installation, workflow, and web parts to business connectivity services, web content management, and claims-based security.

SharePoint Server 2010 Enterprise Content Management

ISBN: 978-0-470-58465-1

If you're interested in building web sites using the new capabilities of enterprise content management (ECM) in SharePoint 2010, then this book is for you. You'll discover how SharePoint 2010 spans rich document management, records management, business process management, and web content management in a seamless way to manage and share content.