Microsoft

Microsoft SharePoint 2013 Developer Reference

Professional

Paolo Pialorsi

Microsoft SharePoint 2013 Developer Reference

Design and develop great solutions using SharePoint 2013

Develop your business collaboration solutions quickly and effectively with the rich set of tools, classes, libraries, and controls available in Microsoft SharePoint 2013. With this practical reference, enterprise-development expert Paolo Pialorsi shows you how to extend and customize the SharePoint environment—and helps you sharpen your development skills. Ideal for ASP.NET developers with Microsoft .NET and C# knowledge.

Discover how to:

- Create custom SharePoint apps and publish them in the Office Store
- Orchestrate your workflows with the new Workflow Manager 1.0
- Access and manage your SharePoint data with the REST APIs
- Federate SharePoint with Windows Azure Access Control Services
- Customize your SharePoint 2013 UI for a better user experience
- Gain a thorough understanding of authentication and authorization

Download code samples at:

http://aka.ms/SP2013DevRef/files

About the Author

Paolo Pialorsi, Microsoft Certified Master on SharePoint, is a consultant and trainer who specializes in developing distributed applications architectures and Microsoft SharePoint enterprise solutions. He is the author of *Microsoft SharePoint 2010 Developer Reference*.

microsoft.com/mspress

ISBN: 978-0-7356-7071-6

9 780735 670716 90000

U.S.A. $49.99
Canada $52.99
[*Recommended*]

Programming/Microsoft SharePoint

Microsoft

Microsoft

Microsoft SharePoint 2013: Developer Reference

Paolo Pialorsi

ISBN: 978-0-7356-7071-6

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

Printed and bound in the United States of America.

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

Acquisitions and Developmental Editor: Kenyon Brown

Production Editor: Christopher Hearse

Editorial Production: Zyg Group, LLC

Technical Reviewer: Jussi Roine

Copyeditor: Zyg Group, LLC

Indexer: Zyg Group, LLC

Cover Design: Twist Creative • Seattle

Cover Composition: Karen Montgomery

Illustrator: Rebecca Demarest

This book is dedicated to my unique and infinite love: Paola!

Contents at a Glance

Contents

Introduction

Microsoft SharePoint is one of the biggest productivity frameworks released by Microsoft during the last 10 years. SharePoint 2013 is just one more step of a fabulous journey (that began in 2001) in the world of business productivity, collaboration, knowledge sharing, search technologies, enterprise social networking, and web content management.

From a developer's perspective, SharePoint is a rich set of tools, classes, libraries, and controls that are useful for building custom solutions and apps focused on making business collaboration and enterprise social networking possible.

This book is an organized reference that provides the support that you need as you develop real and concrete SharePoint solutions and apps, taking advantage of the main libraries and tools offered by the product. This book covers the key topics in the field of developing on SharePoint, targeting both junior and intermediate programmers who want to improve their knowledge of SharePoint.

Beyond the explanatory content, each chapter includes clear examples and downloadable sample projects that you can explore for yourself.

Who should read this book

This book exists to help existing Microsoft .NET developers understand the architecture and core topics of SharePoint 2013 while building Internet, intranet, and extranet sites, as well as developing custom solutions and SharePoint apps.

Although most readers likely will have no prior experience with SharePoint 2013, the book is also useful for those familiar with earlier versions of SharePoint and are interested in getting up to date on the newest features.

Assumptions

This book expects that you have at least a minimal understanding of .NET development and object-oriented programming concepts. Moreover, to develop SharePoint solutions, you need to have a solid knowledge of ASP.NET and related technologies, such as Simple Object Access Protocol (SOAP), Microsoft Windows Communication Foundation (WCF), and web services. Although you can extend and customize SharePoint with most (if not all) .NET language platforms, this book includes examples in C# only. If you are

not familiar with this language, you might consider reading *Microsoft Visual C# 2012 Step by Step*, by John Sharp (Microsoft Press, 2013).

With a heavy focus on web development and server-side technologies, this book assumes that you have a basic understanding of web platforms, application servers, and scalable software architectures. Some of the topics covered in this book require a robust knowledge of .NET Framework 4.*x*, and WCF in particular.

Who should not read this book

This book does not target IT professionals who are seeking information on how to deploy, configure, and maintain a SharePoint farm. However, some discussion about deployment is given throughout the book for the sake of completeness. Similarly, this book does not cover topics concerning site branding or public-facing Internet sites.

Organization of this book

This book is divided into six parts, each of which focuses on a different aspect or technology within SharePoint 2013.

Part I, "Getting started," provides a quick overview of SharePoint 2013 and its data foundations, with a focus on using the technology as shipped, but not yet extending it with custom code.

Part II, "Developing SharePoint solutions," focuses on the core libraries for developing solutions on the server side using the SharePoint Server Object Model and the new LINQ to SharePoint provider. It also focuses on developing for the client side, using the various flavors of the SharePoint Client Object Model and SOAP services. This part of the book is full of examples and code excerpts, and you can use it as a concrete reference for everyday solutions.

Part III, "Developing SharePoint apps," covers how to develop SharePoint apps, which are some of the most interesting new features of SharePoint 2013 from a developer perspective. You will find a step-by-step guide about how to create various kinds of apps, as well as information about the new Representational State Transfer (REST) APIs introduced with SharePoint 2013 for consuming SharePoint from external apps. Moreover, you will learn how to develop remote event receivers to create apps capable of reacting to events happening in SharePoint.

Part IV, "Extending SharePoint," provides deep coverage of the various techniques and extensibility points available for customizing and extending the native SharePoint environment. Four chapters full of realistic examples will help you learn how to create Web Parts, custom pages, and web templates. You will also learn how to take advantage of Business Connectivity Services (BCS) to consume external data sources.

Part V, "Developing workflows," delves into workflow development. It starts with a brief introduction of Windows Workflow Foundation (WF) 4.0 and the new workflow architecture in SharePoint 2013, moving to workflows designed with SharePoint Designer 2013 or developed with Microsoft Visual Studio 2012. This part ends with more advanced topics, such as workflow forms, custom activities, and workflow management services.

Part VI, "Security infrastructure," examines the security infrastructure of SharePoint from an architectural viewpoint, covering topics like authentication, authorization, and the claims-based approach, and delves into identity federation and custom claims-based scenarios. You will learn how to federate SharePoint 2013 with Windows Azure Access Control Services (ACS) and with a custom self-developed identity provider.

Finding your best starting point in this book

The different sections of this book cover a wide range of technologies associated with SharePoint. Depending on your needs and your existing understanding of the SharePoint platform, you might wish to focus on specific areas of the book. Use Table 1 to determine how best to proceed.

TABLE 1 Where to start

If you are	Follow these steps
New to SharePoint development or an ASP.NET developer	Focus on Parts I, II, III, and IV, or read through the entire book in written order.
Familiar with earlier releases of SharePoint	Briefly skim Part I; Chapter 3, "Data provisioning," in Part II; and Part III if you need a refresher on the core concepts. Then read about the new app model in Chapter 8, "SharePoint apps," in Part III; and be sure to read Parts V and VI.
Interested primarily in developing workflows	Read Part II; Chapter 9, "The new SharePoint REST API," in Part III; and Part V.
Interested primarily in developing SharePoint apps	Read Part I; Chapter 3 and Chapter 4, "SharePoint features and solutions," in Part II; and Part III.

Most of the book's chapters include hands-on samples that let you try out the concepts you've learned. No matter which sections you choose to focus on, be sure to download and install the sample applications on your system.

Conventions and features in this book

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

- In most cases, the book includes exercises for Microsoft Visual C# programmers.
- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a task successfully.
- Language keywords (apart from code blocks) appear in italic font.
- A vertical bar between two or more menu items (for example, File | Close) means that you should select the first menu or menu item, then the next, and so on.

System requirements

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

- Windows 7 (x86 and x64), Windows 8 (x86 and x64), Windows Server 2008 R2 (x64), or Windows Server 2012 (x64)
- Microsoft Visual Studio 2012 (Ultimate, Premium, or Professional)
- Microsoft Office Developer Tools for Visual Studio 2012
- A valid Microsoft Office 365 developer subscription
- A computer that has a 1.6 GHz or faster processor (2 GHz recommended)
- 1 GB (32-bit) or 2 GB (64-bit) RAM (add more RAM if running SharePoint on-premises in virtual machines)
- 10 GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9–capable video card running at a resolution of 1024×768 or higher
- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection to download software and chapter examples

To run an on-premises SharePoint farm, you will need the following:

- Windows Server 2008 R2 Service Pack 1 (SP1) (x64) or Windows Server 2012 (x64)
- SQL Server 2008 R2 SP1 (x64) or SQL Server 2012 (x64)
- A computer that has at least a 64-bit four-core processor
- A minimum of 8 GB RAM (16GB RAM recommended)
- 80 GB of available hard disk space

Depending on your Windows configuration, you might require local administrator rights to install or configure Visual Studio 2012, SQL Server 2008/2012, and SharePoint 2013 products.

Code samples

You can download the code samples for this book from the following page:

http://aka.ms/SP2013DevRef/files

The code sample ZIP file includes a child ZIP file for each chapter, which provides sample projects. In particular, you can find the following:

- **Ch-03-Data-Provisioning.zip** Includes a single Microsoft Visual Studio 2012 project, which provisions some data structures (content types and list definitions).
- **Ch-05-Server-Object-Model.zip** Includes a single Visual Studio 2012 project illustrating how to use the SharePoint Server Object Model.
- **Ch-06-LINQ-for-SharePoint.zip** Includes a single Visual Studio 2012 project showing how to use LINQ to SharePoint.
- **Ch-07-Client-Side-Technologies.zip** Provides four Visual Studio 2012 projects, which illustrate, respectively, how to work with the .NET Client-Side Object Model (CSOM), the JavaScript Object Model (JSOM), the Microsoft Silverlight Object Model, and the REST service.
- **Ch-08-SharePoint-Apps.zip** Comprises a set of SharePoint app projects that show how to create apps providing the various hosting models (SharePoint hosted, autohosted, and provider-hosted).

- **Ch-09-New-REST-API.zip** Illustrates how to use the new REST APIs through a sample SharePoint app project.
- **Ch-10-Remote-Event-Receivers.zip** Explains how to create remote event receivers by providing a single Visual Studio 2012 project of a SharePoint app.
- **Ch-11-Developing-Web-Parts.zip** Includes a couple of Visual Studio 2012 projects, which provide samples of basic web parts, as well as of advanced web parts.
- **Ch-12-Customizing-the-UI.zip** Includes a single Visual Studio 2012 project that provides many samples about how to create custom pages, custom ribbons, custom actions, and so on.
- **Ch-13-Web-Templates.zip** Provides samples about how to create a site definition, a site template, and a web template.
- **Ch-14-Business-Connectivity-Services.zip** Includes a Visual Studio 2012 project of a SharePoint app consuming a third-party OData service, a sample project of a custom BCS model, and a WCF service available for consuming via BCS.
- **Ch-15-WF45-Intro.zip** Provides a simple Visual Studio 2012 project that illustrates the basic capabilities of WF 4.5, aside from SharePoint 2013.
- **Ch-16-SP-Workflow-Fundamentals.zip** Includes basic samples of workflows for SharePoint 2013 created by using Microsoft SharePoint Designer 2013.
- **Ch-17-Workflow-Development.zip** Provides some Visual Studio 2012 projects that illustrate how to create basic workflows, workflows in SharePoint app, custom workflow forms, and custom tasks.
- **Ch-18-Advanced-Workflows.zip** Provides three Visual Studio 2012 projects illustrating how to create advanced workflows and custom actions, and how to consume the new workflow management services.
- **Ch-20-Claims-Fed-OAuth.zip** Includes a set of Visual Studio 2012 projects that show how to create a custom identity provider, as well as a custom claims provider.

You can use these sample projects as a reference for everyday needs, and you may find it useful copy code excerpts from these samples into your real solutions.

Acknowledgments

This book has been a long and time-consuming process for me. I have worked toward the completion of this project for about one year. However, a book is the result of the work of many people. Unfortunately, only the author has his or her name on the cover. This section is only partial compensation for the other individuals who helped out.

First, I would like to thank Microsoft Press, O'Reilly, and all the publishing people who contributed to this book project. Mainly, I'd like to thank Ben Ryan and Kenyon Brown, who—once again—trusted in me and gave me the opportunity to realize an idea I have believed in for a long time. Ken supported me through this book project for more than a year; he helped me focus on the content outline, and provided suggestions and guidelines to accomplish this task. Another person deserving a really big acknowledgment is Linda Laflamme, who assisted me along the whole project timeline, keeping me on track, reviewing my chapters, and providing thorough suggestions, feedback, and tips. From the copyediting team, I would like to thank Christopher Hearse and Damon Larson for their accurate work.

I would also like to thank Jussi Roine, one of the most brilliant SharePoint Microsoft Certified Masters (MCMs) that I know, for his accurate, smart, proactive, and great technical review. Jussi, you did a really great job—thank you very much, buddy! You deserve gallons of beer!

I will never stop thanking my mentor, Giovanni Librando. As usual, Giovanni provided me a wealth of ideas, feedback, and tips to achieve this goal.

I'd like to thank my parents and my original family for their support and presence during the last year and for having trusted me during my entire professional career.

Lastly, but most importantly, I want to thank my family—my wife, Paola; my son, Andrea; and my daughter, Marta—for their support, patience, and understanding during the last year. It has been a difficult and very busy year. You have supported me greatly, and you renounced spending many hours with me because of this book. I know I've asked a huge sacrifice of you, and I want to thank you for your support, trust, and understanding!

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/SP2013DevRef/errata

If you find an error that is not already listed, you can report it to us through the same page.

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

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

We want to hear from you

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

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

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

Stay in touch

Let's keep the conversation going! We're on Twitter:

http://twitter.com/MicrosoftPress

PART I

Getting started

CHAPTER 1

Microsoft SharePoint 2013: A quick tour

This chapter explores Microsoft SharePoint 2013 and what it offers to developers who are creating real-world business solutions. To begin, you will focus on the main features and architecture of SharePoint, as well as the rich set of capabilities the platform provides. Next, you will compare the various SharePoint editions. Finally, you will explore the available developer tools. If you already know SharePoint 2013 or have worked with it, you can probably skip this chapter; however, if you haven't yet acquired SharePoint at all, or if you are working on previous versions of SharePoint, such as SharePoint 2007 or SharePoint 2010, you should continue on with the tour.

What is SharePoint?

Microsoft often defines SharePoint as a business collaboration platform that makes it easier for people to work together. As a software developer, I prefer to define it as a platform with a rich framework for developing business solutions. From a developer's perspective, SharePoint is simply a rich set of tools, classes, libraries, controls, and so on, that are useful for building business solutions focused on collaboration, content management, social networking, content searches, and more.

Many people think of SharePoint as a platform that's ready to use for building websites—usually for intranet or extranet scenarios. That's true, but it's less than half the story! Certainly, SharePoint *is* a platform for building websites, and of course, it can target intranet and extranet sites. But it is much more, as well; you can use it to build any kind of web solution, including Internet publishing sites, by taking advantage of its well-defined and ready-to-use set of tools, based on a secure, scalable, and maintainable architecture. You can think of SharePoint as a superset of Microsoft ASP.NET, with a broad set of services that can speed up the development of web-based collaborative solutions.

You should use SharePoint as a shared connection point between users, customers, and whoever else uses your websites and the applications they utilize. The basic idea of SharePoint is to share content, applications, and data to improve collaboration and provide a unique user experience.

SharePoint itself is primarily a container of content and apps. Content is organized in *lists*, and each list is made up of items. A list can consist of simple items with custom metadata properties called *fields*. Lists can also be libraries of documents, which are a particular kind of item that correspond to document files. Almost always when you develop a SharePoint solution, you manage lists and items.

In Chapter 2, "SharePoint data fundamentals," you will learn more about the architecture of data management in SharePoint 2013.

Main benefits

Microsoft grouped the features and services provided by SharePoint 2013 into five main categories of benefits: Share, Organize, Discover, Build, and Manage. Figure 1-1 shows these benefits, and the sections that follow provide a brief description of each.

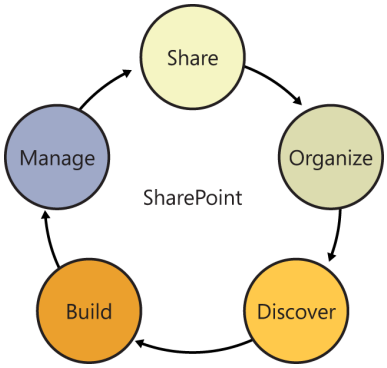


FIGURE 1-1 The native benefits of the SharePoint 2013 platform.

Share

SharePoint 2013 enables you to share ideas and content with others. For example, you can use SharePoint for storing and sharing documents, contacts, and tasks; organizing meetings; managing business processes; and more. When you share something with SharePoint, you can also put it in the social network of your colleagues, customers, partners, and contacts in general, regardless of whether they are on your corporate network, on Facebook, on Twitter, or elsewhere. Through SharePoint, people can discover what you shared, as well as share contents with you. Using the new social features of SharePoint 2013, you can keep track of what your colleagues are working on.

With SharePoint 2013 and the new Microsoft Office 2013, you can publish documents and content from any Office application, sharing them with people inside or outside your organization. You can take advantage of these capabilities from your desktop computer as well as from any Internet-capable mobile device, such as Microsoft Surface and other tablets running Microsoft Windows 8 or RT, as well as smartphones based on the Windows Phone operating system or devices based on iOS.

When you share content through SharePoint, you can update your activity feed in order to make people aware of what you are doing, keeping in touch with your colleagues wherever you are, with any kind of device.

Organize

Through SharePoint 2013, you can organize your projects and tasks, and even integrate SharePoint with Microsoft Outlook and Microsoft Project to keep your projects on track. The product will help you manage tasks, as well as their status and due dates. You will be able to keep your team connected, through specific team sites, which enable you and others to track meetings, share documents, store emails, and do whatever else is useful for your team collaboration.

The new SkyDrive Pro feature provided by SharePoint 2013, which supersedes SharePoint Workspace, allows you and your colleagues to sync all the shared files to your desktop, as well as to your tablet, with Windows 8. This way, the content will always be with you, even when you are offline, traveling, or working at home. Upon connection with the network, any files you worked on offline will be automatically synchronized with their online counterparts.

Discover

Since it was first introduced, one of stand-out features of SharePoint has been its search engine. Having a platform for storing, sharing, and organizing content would be useless without the capability to discover and retrieve it. With SharePoint 2013, you can search for content via a professional search engine, which can be customized for your needs.

With SharePoint 2010, Microsoft introduced an improved and more accurate relevance engine that was based on usage and history. Moreover, it included the FAST for SharePoint edition for supporting large-scale search scenarios, together with professional search-oriented features. Now, the FAST for SharePoint engine is no longer a separate product, and all of its main features are included in the standard SharePoint 2013 search engine. In addition, the SharePoint 2013 search engine has the ability to suggest more relevant results and provide recommendations on people and documents to follow. The search engine is now people-centric and social-centric, enabling you to find people and connect with them, based on their interests, projects they contributed to, and documents they worked on.

You can use all the content, search results, people, and insights to create reports, scorecards, dashboards, and whatever else is helpful for providing meaningful data. Microsoft Excel 2013, Excel Services, PowerPivot, and Power View for SharePoint can assist you in this task as well.

Given all these capabilities, you can consider SharePoint 2013 a solid platform for building data and content-based, search-driven applications, oriented toward social networking and collaboration.

Build

One of the most exciting new features of SharePoint 2013 is its apps-extensibility model. Thanks to this new feature, you can develop custom apps for Office 2013 and SharePoint 2013, using the power of the cloud. You can design everything from business apps for the marketplace at large to a corporate catalog targeting your employees.

Developing a custom app is as simple as combining the apps-extensibility model with such well-known technologies and protocols as JavaScript, HTML, OAuth, and the versatility of the cloud. If you prefer, of course, you can also host your custom apps on-premises, but hosting an app in the cloud provides you with a more scalable infrastructure ready to grow with your business. For an in-depth discussion of creating custom apps, see Part III, "Developing SharePoint apps."

Manage

Nowadays, a key aspect of an IT solution is management, both from a tooling perspective and from the viewpoint of budget and costs reduction. SharePoint 2013 gives you a mature, maintainable, and manageable environment, which can be hosted on-premises as well as in the cloud, using Microsoft Office 365. You can also keep some of your services and content on-premises while deploying others on Office 365, within a hybrid infrastructure.

The new capabilities of Office 365 reduce the time to market for your solutions, allowing you to concentrate your resources and time on the project, the contents, and the custom features, rather than on the infrastructure under the cover.

Many of the solutions in this book are suitable both for on-premises and cloud scenarios, thanks to the common infrastructure behind the scenes.

SharePoint basic concepts

To give you a better understanding of what SharePoint is and how to best use its features, this section takes a brief tour through the product and provides introductions to a few of its most useful features and capabilities.

SharePoint Central Administration

The target audience for this book consists of SharePoint developers, not IT professionals. Therefore, the book does not cover administrative tasks, and it does not provide instructions on how to set up SharePoint from scratch. Nevertheless, as soon as you install a SharePoint server farm, you are presented with an administrative console called SharePoint Central Administration (SPCA) with which you manage the entire farm.

More Info To learn how to deploy and administer a SharePoint farm, read *Microsoft SharePoint 2013 Administrator's Companion*, by Brian Alderman (Microsoft Press, 2013).

SPCA is a website based on the SharePoint engine; it's designed to administer and monitor a SharePoint server farm. When you deploy a new farm, by default the first server takes the role of SPCA host. Nevertheless, in a well-defined SharePoint server farm, you should deploy at least two servers hosting SPCA, for better availability and business continuity of the farm. Using SPCA, you

can configure servers and servers' roles, define farm topology, and create new web applications and site collections.

Because SPCA is an actual SharePoint site, you can use everything you will learn in this book to customize this site, too. Thus, you can build solutions to extend the SharePoint administrative interface. However, keep in mind that because SPCA is an administrative site responsible for the whole farm, you should avoid using it as a development or test site.

The following list describes the main areas of SPCA:

- **Application Management** Here, you can manage existing web applications, as well as create new web applications, site collections, and content databases. You will learn more about these topics later in this chapter and in Chapter 2.
- **Monitoring** From this area, you have access to a set of tools for monitoring the farm, checking for issues, and solving problems.
- **Security** Here, you can manage administrative accounts and services' accounts of the farm, and configure all the security-related features.
- **General Application Settings** This is the area where you manage general settings, such as site directory and search engine settings, content deployment features, form services, and more.
- **System Settings** From this area, you can manage servers in the farm, the farm topology, services on servers, and farm customization features.
- **Backup and Restore** This area provides access to all the tools for managing and handling disaster recovery tasks.
- **Upgrade and Migration** Here, you can manage upgrade and patching tasks.
- **Apps** This area provides access to the app configuration and management tools. You can configure and monitor installed apps and apps licenses, as well as your corporate catalog of apps.
- **Configuration Wizards** This area provides a wizard to configure the farm from scratch.

Note You should consider using the configuration wizards very carefully, and in most cases you should avoid using them. In fact, a real SharePoint farm should never be installed using a wizard. On the contrary, you or the IT professionals you work with should carefully design the farm, assign roles to the servers, determine the services to run, and in general think about and model whatever else is needed to make your SharePoint farm work properly.

Figure 1-2 shows the SPCA home page. Note the status bar at the top of the screen, which in Figure 1-2 highlights some issues regarding the farm's current configuration that were detected by

the SharePoint Health Analyzer service. The SharePoint Health Analyzer is a very useful tool that monitors the status of the farm, helping to maintain it at the optimum service level.

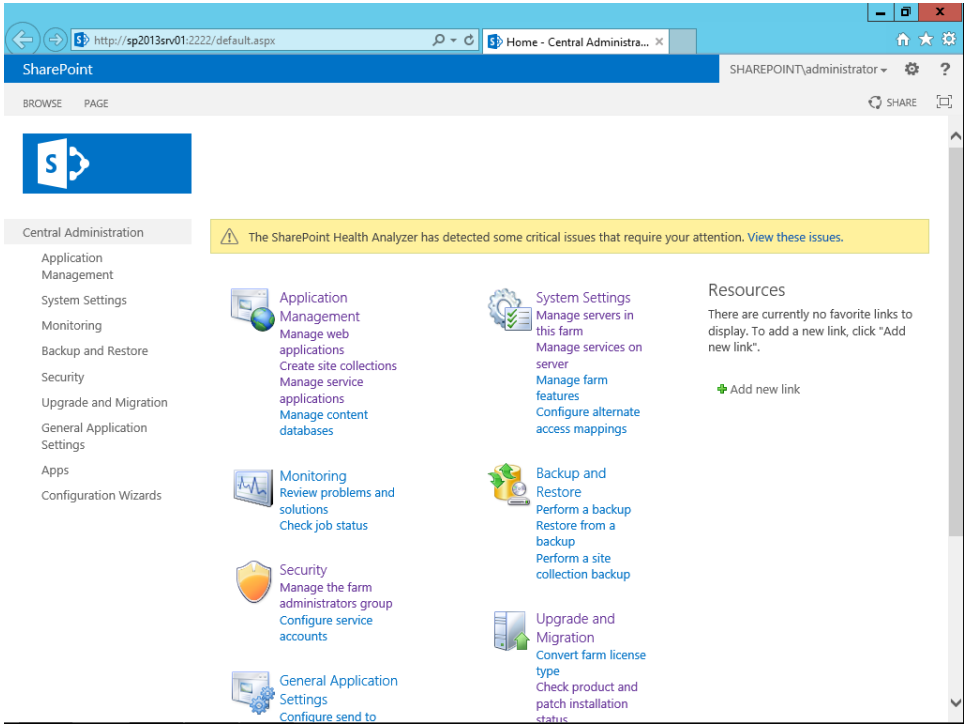


FIGURE 1-2 The SPCA home page of a SharePoint 2013 farm.

SharePoint Administration via PowerShell

As with many other server products from Microsoft, SharePoint can be managed using Windows PowerShell and scripting. SPCA is a good option for managing a SharePoint farm through a set of visual tools and a web browser. However, having a text-based scripting engine to query, manage, configure, and even install a SharePoint farm from scratch is a fundamental aid for IT professionals. In SharePoint 2013, everything you can do with SPCA can also be done using some PowerShell scripts. Moreover, PowerShell enables additional controls that are not available from SPCA.

The power of having a scripting engine for managing almost every aspect of a SharePoint farm is enormous and unpredictable. For example, you can define a PowerShell script to deploy a farm from scratch, or you can use a script to add a server to an already existing farm. You can create and configure web applications, sites, and services using a script. Moreover, you can create scripts to configure the topology of your farms. All these scripts become extremely useful and powerful whenever you need to reproduce the same tasks for multiple customers or sites.

Even if you are a developer, you can benefit from having a rich library of predefined and parameter-based PowerShell scripts. In fact, you can use those scripts to deploy development farms, as well

as test environments. Moreover, using a script, you can deploy your customizations onto an on-premises farm. This book will not cover PowerShell in depth, because there are many other topics to cover that deal more specifically with SharePoint development. Nevertheless, you should consider reading a book on PowerShell for SharePoint as a companion to this book.

More Info To learn more about Windows PowerShell, consult "Windows PowerShell" on MSDN (*http://msdn.microsoft.com/en-us/library/dd835506.aspx*) or *Windows PowerShell Pocket Reference*, by Lee Holmes (O'Reilly, 2012).

Site collections and websites

One fundamental concept embodied by SharePoint is that of a site collection. A *site collection* is a logical container that holds a set of SharePoint sites hosted in a web application. Whenever you work in SharePoint and you want to publish a site, regardless of whether it's an Internet, intranet, or extranet solution, you will have at least one web application with one site collection, made of one site. Grouping sites in site collections allows those sites to share content, administrative settings, security rules, and, optionally, users and groups.

To create a new site collection, you need a web application, which you can create by selecting the Manage Web Applications menu item from the SPCA home page, or by using the corresponding PowerShell command. Avoid using the web application that hosts SPCA. After you have a web application, you can create a new site collection by selecting the Create Site Collection menu item on the SPCA home page. A dialog box will appear, asking you for a title, a description, and a URL relative to the parent web application.

Every site collection is administered by a site collection administrator, who is a user authorized to administer an entire site collection, including the websites it contains. Every site collection must have at least one site collection administrator, but it can have more than one. Thus, when creating a new site collection, you need to designate a primary site collection administrator and, optionally, a secondary one. After having created a site collection, you will be able to add as many site collection administrators as you like. A site collection administrator has the rights to create, update, or delete any site contained in a site collection. The administrator also has full rights to administer content within those sites.

When you create a site collection, you should also choose a template from which to start. If you need, you can select it from a number of predefined templates that are shipped with SharePoint. By default, the template will create a new site collection with at least one site at the root of the site collection. Templates are divided into functional groups and into two families. In fact, SharePoint 2013 comes with a new family of templates, as well as the previous template family from SharePoint 2010, for backward compatibility. Following are the five main functional groups of SharePoint 2013 templates:

- **Collaboration** These are sites whose structure has been designed to facilitate collaboration. The Collaboration group includes the following templates: Team Site, Blank Site, Document

Workspace, Blog, Group Work Site, Developer Site, Project Site, Community Site, and Visio Process Repository.

- **Meetings** This group contains templates for sites related to meetings and meeting organization. The available templates are Basic Meeting Workspace, Blank Meeting Workspace, Decision Meeting Workspace, Social Meeting Workspace, and Multipage Meeting Workspace.
- **Enterprise** These templates target enterprise-level needs in the areas of document management, policies, and so on. They include Document Center, Discover Center, Records Center, Business Intelligence Center, Enterprise Search Center, My Site Host, Community Portal, and Basic Search Center.
- **Publishing** This group corresponds to sites intended for web-publishing purposes. The available templates are Publishing Portal, Enterprise Wiki, and Product Catalog.
- **Custom** This is where you can develop your own site templates. Also in this group is a list of all the available custom templates, if any exist.

Figure 1-3 shows the home page of a site collection created by using the Team Site template of SharePoint 2013.

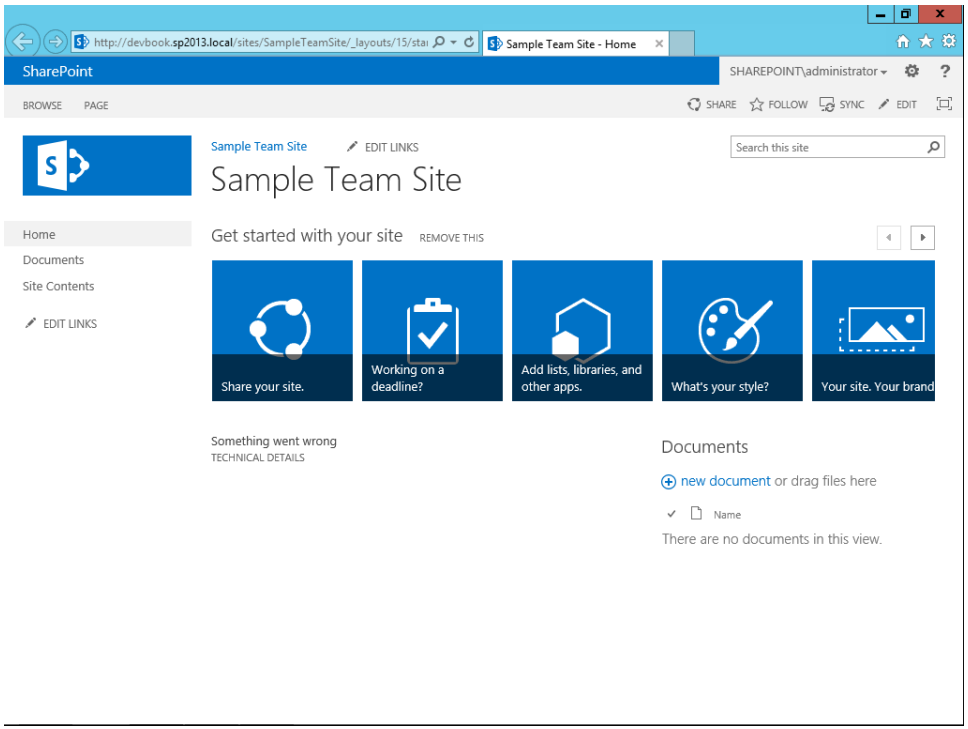


FIGURE 1-3 The home page of a Team Site template site collection.

Lists, libraries, items, documents, and other apps

Every SharePoint site is composed of lists of items. When the items are simple—that is, they don't correspond to documents or files, but are made of custom metadata properties only—they're termed *lists* and *list items*. When the items correspond to files, they're called *document libraries* or just *libraries*.

Every site template includes some predefined lists that are created when you construct a site using that template. For example, a team site provides a Documents library, a Site Assets library, a Site Pages library, and a few other predefined lists and libraries. Regardless of the site template you start from, you can always create new lists, libraries, and content, as well as activate features to customize your site.

You can browse the contents of these lists and libraries, and, if you have the proper permissions, you can create new apps, which can be lists of contents, libraries, or custom apps either taken from the public marketplace or installed from the corporate catalog. Consider that in SharePoint 2013, everything is called an app. However, a list or a library is still what it is—nothing more and nothing less. You can also add items to already existing lists or upload new files (for libraries) by simply dragging and dropping them from the file system to the webpage. Figure 1-4 shows the UI of SharePoint 2013 while browsing the contents of a document library.

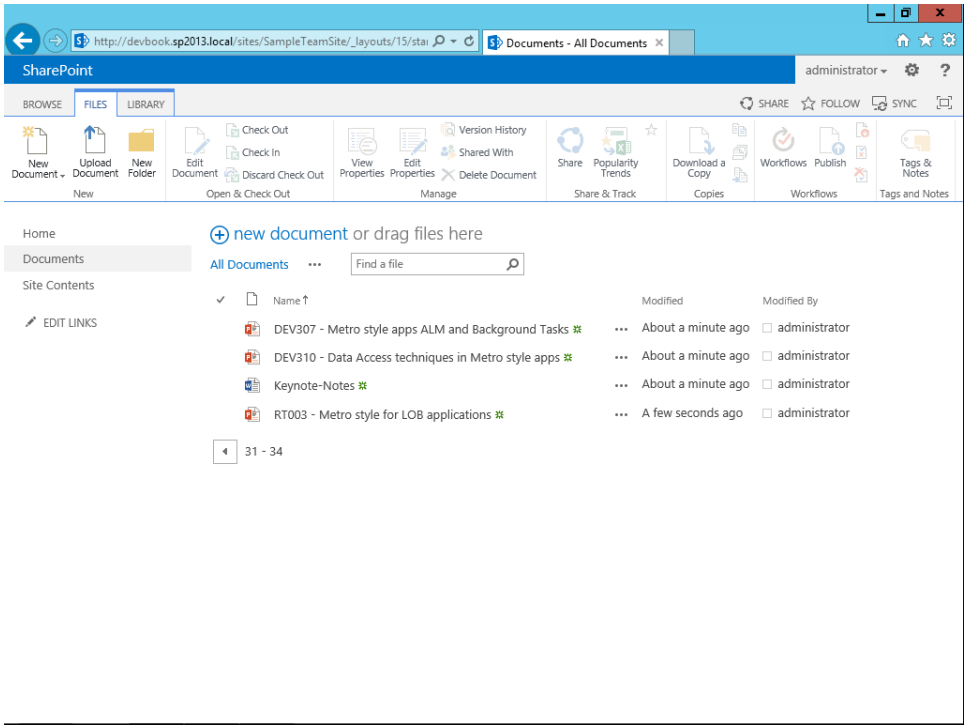


FIGURE 1-4 The default UI of SharePoint while browsing the contents of a document library.

Note also that Figure 1-4 shows the ribbon, which is a feature introduced with SharePoint 2010, to better support end users through a UI similar to the well-known Office interface.

When you want to create a new app, you simply click the gear icon, which is located in the upper-right corner of the webpage, and then select Add An App. As shown in Figure 1-5, you'll see the Apps You Can Add list, from which you can select the type of app that you would like to create.

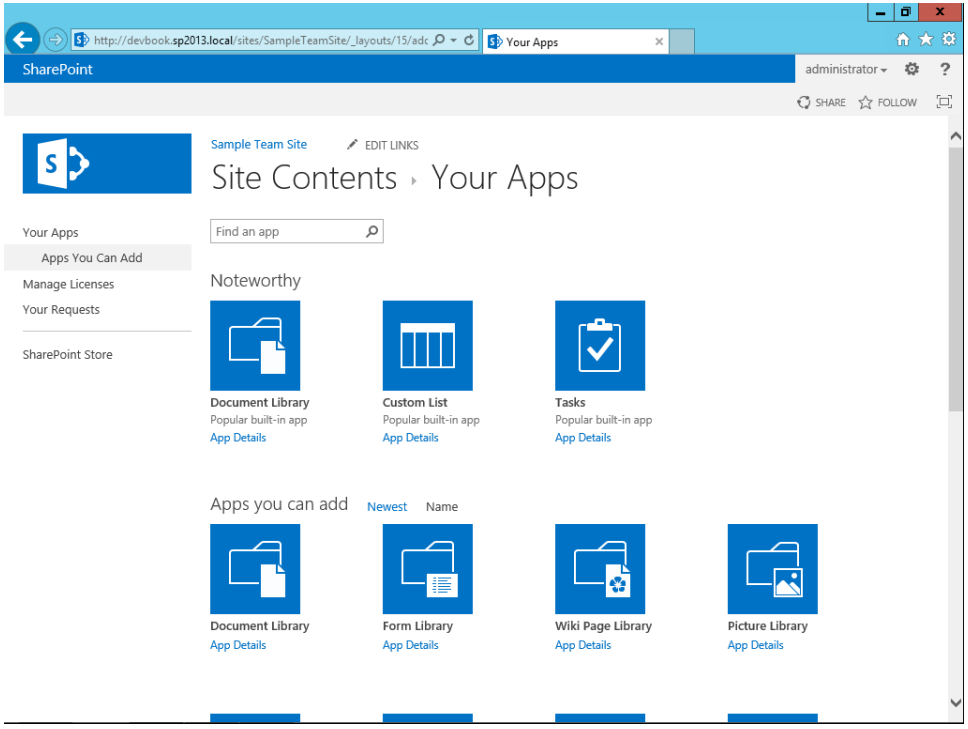


FIGURE 1-5 The UI for adding a new app to a SharePoint site.

If none of the supplied templates of lists and libraries quite fits your needs, you can try or buy an app from the marketplace, and you can install an app from a corporate catalog. Of course, in order to access these, your farm should be connected to the Internet and configured for supporting apps.

App Parts and Web Parts

App Parts are new features of SharePoint 2013, enabling you to enrich pages with external apps and content, which you can create on site or download from third-party sites or the cloud—for example, through the marketplace. An *App Part* is a block of HTML code, empowered with JavaScript and secured with OAuth, typically hosted outside the current site, and eventually integrating and/or consuming some contents within the current site. Later, in Part III of this book, you will learn how to create App Parts and how to consume them from a SharePoint site.

Web Parts have been some of the most notable features of SharePoint since its early versions. In fact, in SharePoint you can define pages made of configurable building blocks (Web Parts) that can be enabled, moved, or hidden by end users. The goal of this feature is to allow users to define their own pages, selecting content from a set of available Web Parts, with full personalization. Every page made of Web Parts is called a *Web Part page*.

With SharePoint 2013, the importance of Web Parts is declining, while the use of App Parts is becoming more prominent. You can think about App Parts as the heirs of Web Parts. A typical SharePoint 2013 solution contains some custom lists and document libraries, along with some apps presented as App Parts and configured in custom pages that show and manage the data stored in those lists and libraries, as well as outside the current site.

Architectural overview

In this section, you'll take a look at SharePoint architecture from a developer's perspective. Figure 1-6 shows some of the main components of SharePoint, from the foundation elements up to the main enterprise-level features.

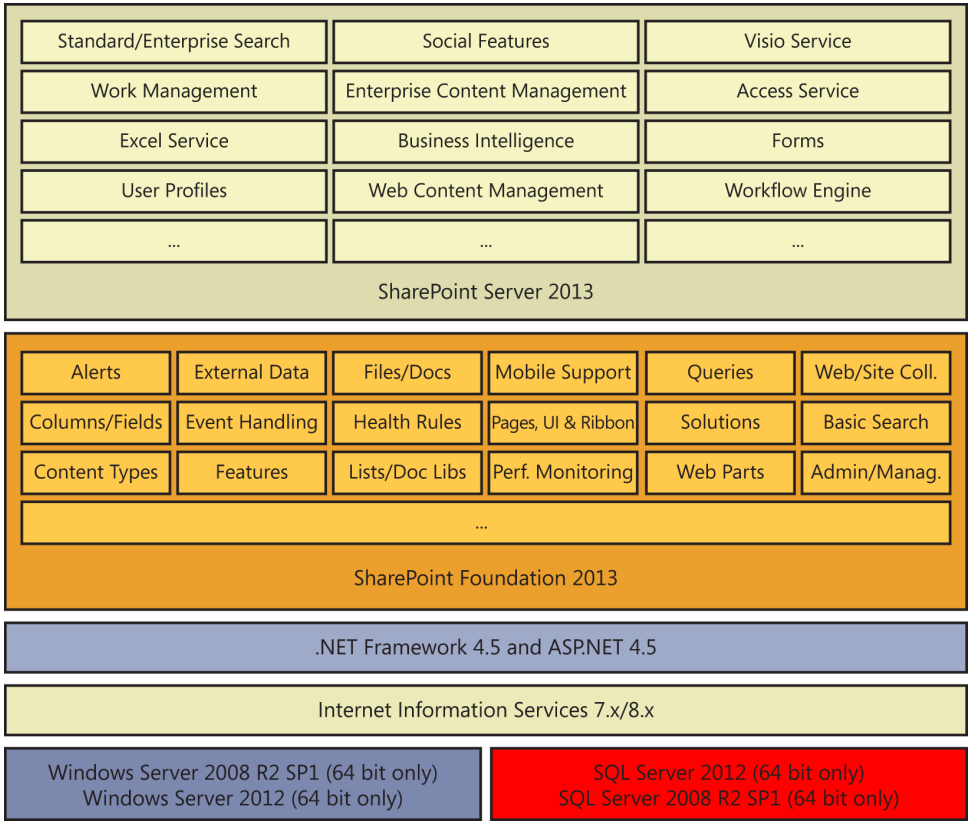


FIGURE 1-6 The architecture of SharePoint 2013.

At the very base of SharePoint 2013 sits the operating system. Starting with SharePoint 2013, the minimum requirement for a production environment is Microsoft Windows Server 2008 R2 Service Pack (SP) 1 (Standard, Enterprise, or Datacenter) or Microsoft Windows Server 2012 (Standard or Datacenter). Although in SharePoint 2010 it was possible to install the product on a workstation machine running Microsoft Windows 7 or Microsoft Windows Vista SP1/SP2, this is no longer allowed with SharePoint 2013. Because SharePoint 2013 is available only in 64-bit versions, the minimum requirement for a deployment environment is a server-based 64-bit operating system (Windows 8 does not qualify as a host operating system for SharePoint 2013).

More Info For further details about the software and hardware requirements of SharePoint 2013, read the document "Hardware and Software Requirements for SharePoint 2013" on TechNet Online, at *http://technet.microsoft.com/en-us/library/cc262485.aspx*.

In addition to the operating system, SharePoint 2013 also requires a database server based on Microsoft SQL Server 2008 R2 SP1 or Microsoft SQL Server 2012. Regardless of which edition of SQL Server you plan to use, you must be running a 64-bit version of the product. SharePoint uses the SQL Server database to store the configuration of SharePoint server farms, as well as the contents of deployed websites and the configuration and contents of all the services under the cover of the overall farm infrastructure.

On top of the operating system and database is an application server provided by Internet Information Services (IIS) 7.5. IIS 7.5 is mandatory, both because it hosts the web applications and because it publishes endpoints for SharePoint infrastructure services, making use of the Windows Process Activation Service (WAS) feature of IIS 7. Use of IIS 8 is suggested in new scenarios that you build from scratch, allowing you to take advantage of all the new features of Windows Server 2012 and IIS 8.

More Info You can find more details about WAS on the "Hosting in Windows Process Activation Service" page on MSDN, at *http://msdn.microsoft.com/library/ms734677.aspx*.

Because SharePoint 2013 is based on Microsoft .NET Framework 4.5 and extends ASP.NET 4.5, the infrastructure requires .NET Framework 4.5. Another element at the foundation of SharePoint 2013 is the Windows Identity Foundation 1.0 framework, which provides claims-based services, extended in order to support OAuth and the new security model of SharePoint 2013. Part VI of this book, "Security infrastructure," digs deeper into these topics.

On top of this foundation sits Microsoft SharePoint Foundation 2013, which is a free platform for building basic SharePoint solutions. Although free and the most basic edition of SharePoint, SharePoint Foundation 2013 contains a great deal of functionality that developers can use to meet the needs of basic portal scenarios.

At the top of the architecture is the SharePoint Server 2013 platform, together with its high-level and enterprise-level services, such as Excel Services, Managed Metadata Services, the User Profile services, the search engine, and so forth.

From a hardware perspective, the minimum memory requirement for a SharePoint 2013 server is 8 GB for a development environment, but this hardly gives you enough room to work. A more realistic minimum, however, is 16 GB for a successful development environment. For a production environment, the suggested memory is 12 GB for a web front-end or an application server, and 24 GB for an all-in-one server. Moreover, every SharePoint 2013 server should have a 64-bit CPU with a minimum of four cores.

Logical and physical architecture

Whenever you deploy a SharePoint environment, in reality, you're deploying a logical architecture called a SharePoint farm. A *SharePoint farm* is a set of servers that have different roles and offer various services that together make up a server farm suitable for hosting a full SharePoint deployment. Here are the common server roles in a SharePoint farm:

- **Front-end web servers** These servers publish websites, often called web applications.
- **Application servers** These servers host back-end services, such as Search services, the User Profile service, Excel Services, and so forth.
- **Database servers** These servers store configuration and content data for the entire SharePoint farm.

The smallest farm you can build is based on a single server; this type is often called the single server farm deployment. However, it is highly recommended that you avoid such a scenario, except for testing or development.

In fact, for the sake of scalability and business continuity, you should deploy a minimum of two front-end web servers, two application servers, and a back-end database server capable of supporting failover (clustering, mirroring, or AlwaysOn). This topology is commonly termed the *smallest fault-tolerant farm deployment*. If you need to scale out and support a wider range of users and sites, you can deploy a more complex farm by introducing some dedicated application servers. For example, real medium-scale and large-scale farms typically have dedicated servers for the search services, as well as dedicated servers for hosting the Office Web Apps services (which is a deployment requirement).

Due to the number and size of servers required for hosting a real production SharePoint farm, SharePoint 2013 farms are usually hosted in virtualized environments, either on-premises or in the cloud. For example, you could evaluate hosting SharePoint 2013 on an Infrastructure as a Service (IaaS) environment like Microsoft Windows Azure Virtual Machines. Moreover, you could also consider directly using Microsoft Office 365.

More Info You can find further information about topologies and architectural diagrams on the "Technical diagrams for SharePoint 2013" page, on TechNet at *http://technet.microsoft.com/en-us/library/cc263199(v=office.15).aspx*.

Regardless of the deployment topology you choose, SharePoint uses a SQL Server database for storing farm configurations and content. Specifically, it creates a main and fundamental farm configuration database as soon as you deploy a new farm. Usually, this database is called *SharePoint_Config* or *SharePoint_Config_{UniqueId}*. If you use the automated setup process, this database is created for you when you deploy the farm for the first time. If you use PowerShell to deploy a new farm, which is highly suggested, you can determine the name of this database by yourself. Furthermore, the SharePoint Deployment And Configuration Wizard creates a set of satellite database files for the main services deployed. For example, it creates a database that stores the contents of the SPCA administrative site. In case you use a PowerShell script to deploy the farm, you can determine the name and location of all SharePoint databases.

From a hierarchical perspective, each SharePoint farm is composed of services, which include all the infrastructure services that make up the SharePoint environment. The most important kind of services are web application services, which correspond to the entry point for web-published solutions. Each web application is made up of at least one site collection and one content database. However, you can deploy multiple site collections within a single web application, and you can deploy multiple content databases for a single web application. A content database is a database file that stores content for one or more site collections. As it relates to SharePoint, content can include items, documents, documents versions, pages, images, and so on. Thus, the database behind a site collection can grow very fast.

Starting with SharePoint 2010 and much more with SharePoint 2013, the server roles and the configurable services have been improved to better support scale-out scenarios. In fact, you can now distribute different roles to dedicated servers, eventually with hardware redundancy.

Figure 1-7 shows a graphical representation of a SharePoint farm with a couple of front-end web servers, both of which publish the same web applications with network load balancing. The first web application (Web Application #1) is made of two site collections (Site Collections #1 and #2), both of which share a common content database (Content #1). The second web application (Web Application #2) is made up of a third site collection (Site Collection #3) and stores its contents in a dedicated content database (Content #2). All the site collections contain one or more websites.

On the back end, there are four application servers, hosting SPCA, the search services, Excel Services, and some other services.

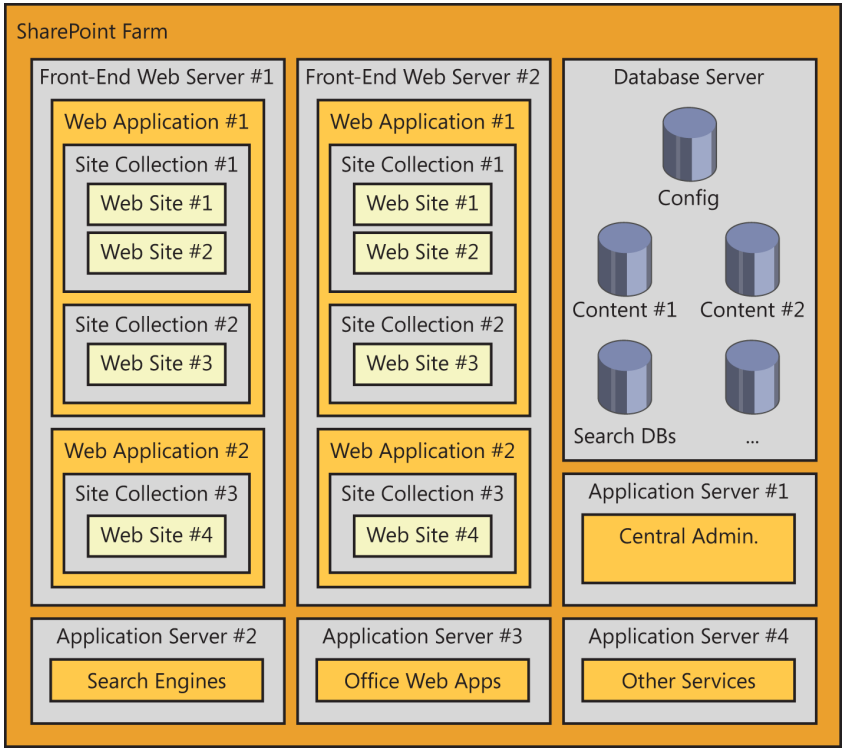


FIGURE 1-7 A simplified schema of a sample SharePoint farm with an N-tier topology.

All the data are persisted in a back-end database server that stores various database files for different purposes.

Service applications

Introduced in SharePoint Foundation 2010, service applications are software services that run in a SharePoint farm. Service applications are intended for sharing resources and capabilities across multiple sites and servers in the same farm, or even across farms. Most importantly, they are extensible and scalable, unlike the Shared Service Providers (SSPs) of Microsoft Office SharePoint 2007.

To clarify the idea of a service application, consider a couple of examples. The search engine in SharePoint 2013 is based on a service application. This means that you can share the same search engine across different servers in the same farm, which is not surprising, but you can also share the same search service across multiple farms. For example, in very large scenarios, you could deploy a search-dedicated farm, without any front-end web server, that exposes only a wide set of servers providing query, index, crawler, content -processing, and analytics components. You could then use this farm to serve many other SharePoint 2013 farms, taking advantage of that shared search service. Another example is Excel Services: if you have a farm that uses Excel Services extensively to make

calculations and create reports on external data, you could decide to deploy Excel Services on two or more dedicated servers in the farm, using them from all the other servers.

These configurations are possible because the architecture of service applications has been designed with scalability in mind. Thus, every service application that runs on a server in the farm can support scalability, and can be installed on two or more servers. At the same time, a farm uses a proxy to consume a service application, which can be published locally, or in some cases can be published by a third-party farm. While a front-end web server consumes a service application, however, it ignores the real location of the service and simply concentrates on consuming it. This is possible because each SharePoint Foundation 2013 farm has a native service application, called the *Application Discovery and Load Balancer Service,* that coordinates service discovery and load balancing for services deployed on more than one application server. By default, each service application proxy communicates behind the scenes with the back-end service application via a secure channel based on Windows Communication Foundation (WCF).

More Info You can find further information about service application architecture and developing a custom service application in the book *Microsoft SharePoint 2010 Developer Reference*, by Paolo Pialorsi (Microsoft Press, 2011), which is the previous edition of this book.

The role of databases

Every SharePoint farm includes one or more back-end database servers. In fact, the back-end SQL server stores the entire configuration of the farm, as well as contents of every site collection and the data for many service applications. For example, the search service stores crawled contents, properties for crawled data, and configuration properties in multiple separate and dedicated database files. For the sake of precision, in SharePoint 2013, the Search service application allocates four databases. The Managed Metadata service has another dedicated database file, but the list of native services using one or more databases on the back end could be longer.

Important Even though you can open a SharePoint database in SQL Server Management Studio and inspect the databases of a SharePoint farm, you should avoid doing that. In addition, you should not base your software solutions on the data structure of SharePoint databases. Thus, you should avoid querying and writing the content of these databases directly. If you do need to read or write their content, take advantage of the various libraries, APIs, and object models discussed later in this book.

Now let's concentrate on pages and content. Recall that each time you create a new site collection using SPCA, you have the opportunity to choose a starting site template. The site template is a set of configuration, layout, and content files that define a site model. You can build your own site templates (you will learn how to do that later in Part IV, "Extending SharePoint"), or you can select one of the existing site templates that are packaged with SharePoint. Whichever site template you choose, under

the covers, SharePoint starts from a set of files stored in the file system of all front-end web servers, and then creates some records in the content database that will host the site collection that you are creating. After the site collection has been created, when you browse to a page using a web browser, the SharePoint engine determines whether the page you have requested resides entirely on the file system, or whether it needs to retrieve some personalized content from the content database and merges that with the page model from the file system, or even whether the page content is completely stored in the content database.

Having a back-end content database available gives you the option to deploy multiple front-end web servers that can share the same content, improving horizontal scalability when necessary. At the same time, maintaining basic page models in the file system improves performance, because loading a page from the file system, unless it has been personalized, is generally faster than retrieving it from an external database server. In the section "SharePoint for developers," later in the chapter, you'll see how SharePoint differentiates between file system and database content sources.

SharePoint editions

SharePoint 2013 is offered in several editions. Even though this book is for developers (as opposed to sales or marketing personnel), it is useful to know the main differences between each edition of the product. The goal of this section is to give you the base knowledge required to choose the appropriate SharePoint edition for each of your projects.

More Info For a full comparison of the SharePoint editions, see the page "SharePoint Online" at *http://technet.microsoft.com/en-us/library/jj819267.aspx*.

SharePoint Foundation

SharePoint Foundation 2013 is the most basic edition of the product. It is free—providing that you run it on a licensed copy of Microsoft Windows Server—and it offers the fundamental features for building simple document storage and collaboration solutions. By default, this edition's main capabilities are accessibility, cross-browser support, basic search features, out-of-the-box pages and Web Parts, new UI features based on dialogs and ribbons, blogs, and wikis.

The Foundation edition also supports the basic infrastructure of Business Connectivity Services, although without any client-side or Office capability. Of course, you'll also find the SPCA controls, all the farm management tools, and services such as the SharePoint Health Analyzer. In fact, if you wanted to, you could deploy a multitier farm using just SharePoint Foundation. Finally, SharePoint Foundation offers all the features supporting custom development, including the Web Parts/App Parts programming model, the Server Object Model, the Client Object Model, event receivers (local or remote), claims-based security, and so on. All these topics will be covered in detail in Part II, "Developing SharePoint solutions," and Part III, "Developing SharePoint apps."

You should use this edition of SharePoint whenever you want to develop custom solutions that do not require any high-level features, such as the document management tools, user profiles, managed metadata, and so on. When you simply need to use SharePoint as a web-based "sharing point" to store content, such as documents, contacts, tasks, and so on, this is the edition that best meets those needs. Quite often, SharePoint Foundation is the right starting point for gaining experience with SharePoint. It also serves well as a bridge: you can start installing Foundation; plus, later on, you will be able to upgrade to SharePoint Server, if the need arises.

SharePoint Server Standard

The Microsoft SharePoint Server 2013 Standard edition is built on top of SharePoint Foundation 2013, adding useful features for building business-level solutions. In particular, you will find features supporting Enterprise Content Management (ECM) and Web Content Management solutions. This edition also provides legal compliance capabilities, including records management, legal holds, and document policies. It also offers support for document sets, which give you the ability to manage related documents as if they were a single entity. It supports document IDs, which assign a unique protocol number to SharePoint site documents. Using this edition, you can target content based on *audiences*, which are profile-based groups of targets. Moreover, you have the capability to use the Managed Metadata service for managing common metadata properties, navigation elements, publishing, and product catalogs across multiple site collections and web applications.

SharePoint Server is the right choice for implementing business-level solutions. For example, SharePoint Server can help you create a content management system (CMS) solution that provides content publishing, content approval, page layouts, web standards (XHTML, WCAG 2.0, and so on) support, and so forth. This edition also supports tags and metadata-driven search refinement, people search, and the whole set of social features. As a business-level tool, it provides features for managing not only content, but also people, profiles, and personal sites. Finally, this edition of the product provides support for developing and executing workflows, hosted either on-premises or in the cloud on Windows Azure.

SharePoint Server Enterprise

Microsoft SharePoint Server 2013 Enterprise edition targets large business solutions and enterprise-level organizations. It extends the capabilities of SharePoint Server Standard by offering support for dashboards, key performance indicators (KPIs), and business intelligence features. It improves search capabilities by offering contextual search, deep search query refinement, extreme scale-out search capabilities, rich web indexing, and so on. It also provides support for Excel Services, Visio Services, Forms Services, and Access Services.

When you need to develop business analysis solutions or complex search-based solutions, you should choose the Enterprise edition.

From a developer perspective, you can install the SharePoint Server Enterprise edition if you have licensing coverage for that, and you can develop solutions for all the editions using a unique environment.

SharePoint Online

Microsoft SharePoint Online is the cloud-based SharePoint offering, based on the Software as a Service (SaaS) paradigm included in Microsoft Office 365. With this edition, you can build SharePoint solutions without building a SharePoint farm on-premises. Instead, by having your farm in the cloud, you can enjoy an external solution free of management costs. As a developer, you are freed to focus only on data, processes, ideas, the content that you want to share, and the apps you want to build. The SharePoint Online offering is available in Standard mode, as well as in Dedicated mode. The Standard offering uses an environment shared with other customers, although it is isolated according to a clear set of multitenancy rules, and you can only extend that environment with code executed in a sandbox or custom apps. On the contrary, the Dedicated offering allows you to have a dedicated server farm on which you can deploy custom solutions with full-trust execution rights, as long as your solutions passes a verification process.

SharePoint for developers

SharePoint offers developers numerous features and capabilities for building custom web solutions. This section provides an overview of those features and services so you can better understand the topics that you will be exploring in the rest of this book.

ASP.NET integration

As a developer, you might be wondering how SharePoint 2013 integrates with ASP.NET to service requests and provide its high-level features on top of the ASP.NET native infrastructure.

Since IIS 7.0, in Windows Server 2008, application pools can run in one of two modes: integrated mode or classic mode. *Classic mode* works like older versions of IIS (IIS 6), taking advantage of the Internet Server Application Programming Interface (ISAPI) filter based on the Aspnet_isapi.dll file. *Integrated mode* provides a unified request-processing pipeline for requests that target both managed (.NET) and unmanaged (non-.NET) resources. Every request is served by a module registered in the application configuration.

SharePoint 2013 provides a *Microsoft.SharePoint.ApplicationRuntime* namespace in the Microsoft.SharePoint.dll assembly. This namespace contains a set of classes that integrate and/or override the default behavior of ASP.NET while in IIS integrated mode. The primary class that handles SharePoint requests is called *SPRequestModule*. It is configured in the web.config file of every SharePoint site, in the system.webServer/modules section. This class registers a number of application events that handle requests, authentication, errors, and so on. One fundamental task of this module is to register the virtual path provider (*SPVirtualPathProvider*), which resolves requests by determining whether the requested content should be retrieved from the content database or from the file system. A *virtual path provider* is a class that provides contents to the ASP.NET pipeline by retrieving them from a virtual file system.

Server-side technologies

SharePoint offers developers a rich set of server-side tools. First, you can use the SharePoint Server Object Model, which allows you to interact with SharePoint through a large set of libraries and classes. Using these classes, you can read, manage, and administer data stored in SharePoint. More generally, you can use the Server Object Model to do almost anything that SharePoint itself can do, because SharePoint itself uses that same object model. You can use the Server Object Model on a SharePoint server only, because it has some dependencies not satisfied by other servers. You will learn more about this tool in Chapter 5, "Server Object Model."

On the server side, you can also use the LINQ (Language Integrated Query) programming model, exploiting the LINQ to SharePoint provider, by which you can query and manage SharePoint data using a fully typed programming model, much as you would when managing data stored in SQL Server using LINQ to SQL. Chapter 6, "LINQ to SharePoint," discusses this LINQ query provider in more detail.

Client-side technologies

One of the biggest news of SharePoint 2013, from a developer perspective, is the improvement of the client-side technologies for consuming SharePoint data and interacting with remote SharePoint servers. In fact, you can exploit a rich set of client-side technologies offered specifically for this purpose. For example, the SharePoint Client Object Model lets you interact with SharePoint from a client using a set of classes that are similar to the Server Object Model, but work on any client that supports .NET, Microsoft Silverlight, or JavaScript. The Client Object Model is available in three different flavors: .NET managed, Silverlight, and JavaScript. The Client Object Model versions are almost functionally identical on all three platforms. You can also use SOAP (Simple Object Access Protocol) services published by SharePoint, even though they are deprecated and available for backward compatibility only. Furthermore, you can use the REST (Representational State Transfer) API to access and manage SharePoint data by using a protocol for querying and updating data via an HTTP/XML communication channel called OData (Open Data Protocol, documented at *http://www.odata.org)*. Moreover, starting with SharePoint 2013, you can take advantage of a new and rich set of APIs published via HTTP and accessible from any device; these APIs are useful for consuming data and interacting with site collections, sites, services, and whatever else you could need to create a SharePoint app or solution. From a security viewpoint, you can use the common OAuth (Open Authentication) standard to secure communication and authenticate/authorize both users and apps while consuming data and interacting with SharePoint services.

All of these client-side technologies are discussed throughout the book, and in particular in Parts II and III.

App Parts, Web Parts, and the UI

Another area of interest for developers is customizing the UI. Many SharePoint developers working on SharePoint 2010 or earlier spent their time developing Web Parts, Web Part pages, and UI customizations. SharePoint 2013 still provides a rich object model, and even backward compatibility,

for building custom Web Parts and Web Part pages, as well as a set of UI customization tools that simplify working with AJAX (Asynchronous JavaScript and XML), dialog boxes, the ribbon, and so on. Now, with SharePoint 2013, you can extend and customize the UI by creating apps and App Parts. You can think about App Parts as blocks of content, consumed from a remote app, that play the same role as Web Parts did in the past. You will see how to develop App Parts in Part III of this book.

Data provisioning

As soon as you begin working with SharePoint, you will face the need to define packages for automatically deploying data structures. Working with SharePoint generally involves designing new lists and new content types, which are reusable typed definitions of metadata models. However, if you define your models using the web browser, you won't have a high-level modeling approach; everything you do must be migrated and/or executed again in the quality assurance (QA) and production environment.

Fortunately, there are tools and techniques that allow you to model a data structure—optionally based on custom contents and fields—and deploy that model to customers' sites. These tools also provide support for deploying updated versions of the solution in the future. You'll see more on this subject later in this chapter, in the section "Features, solutions deployment, and sandboxing." You will learn how to define custom data models for automated provisioning in Chapter 3, "Data provisioning."

Event receivers and workflows

With SharePoint, since version 2007, you can use local event receivers to intercept users' actions and/or events and subsequently execute some lightweight server-side code. Now, with SharePoint 2013, you also have the capability to create remote event receivers for invoking external and remote services. These receivers are capable of handling events like item insertion, updating, deletion, and so on. This is a useful feature for implementing simple process-handling solutions or business-processes coordination, activating external processes upon user actions in SharePoint. Moreover, you can use remote event receivers to make apps communicate with parent websites. Chapter 10, "Remote event receivers," dives into this subject.

Similarly, when you need to define complex and long-running business processes that respond to events from the UI and interact with end users, you can define *workflows*. With SharePoint 2013, the workflow engine has been redesigned from scratch, using the new Workflow Manager 1.0 engine, based on Workflow Foundation 4.5, together with a new application server role that can be hosted on Windows Azure or on-premises. This functionality deserves a thorough exploration, so this book discusses it in four dedicated chapters, in Part V, "Developing workflows."

Features, solutions deployment, and sandboxing

As a complete development platform, SharePoint 2010 introduced deployment services and capabilities by which you can deploy and upgrade solutions during a project's lifetime. In SharePoint 2013, all these features are still available and suitable for developing complex customizations and solutions.

Specifically, SharePoint offers the opportunity to create deployment packages, called Windows SharePoint Services Solution Packages (WSPs). You can use these packages to automate setup and maintenance tasks across an entire server farm. In addition, you can deploy these solutions in a sandboxed environment. The packages consist of features, which are atomic sets of extensions that you can develop, install, activate, and manage with a specific set of administrative tools. In Chapter 4, "SharePoint features and solutions," you will learn how to create and deploy such packages. In Part III of the book, you will learn how to create and deploy custom apps as a suitable alternative to implementing SharePoint solutions.

Security infrastructure

The SharePoint security infrastructure is another topic that affects both software development and the architecture of solutions. In fact, to develop robust and solid solutions, a developer should have a high degree of confidence in, and knowledge about, SharePoint authentication and authorization policies. The key security aspects of SharePoint 2013 are its claims-based approach and support for the OAuth protocol. Part VI of the book is fully dedicated to security matters.

Business Connectivity Services

Business Connectivity Services is another feature that is generally useful when developing solutions. This feature supports consuming external data within SharePoint, and has a design almost identical to data directly stored in SharePoint. The sources of this external data can be an RDBMS, like SQL Server or any ODBC-compliant data source; a WCF/SOAP service; a custom .NET object model; or an OData service. Chapter 14, "Business Connectivity Services," will cover this topic.

Windows PowerShell for developers

Another interesting capability is that you can administer and automate SharePoint administrative tasks using the Windows PowerShell console. Windows PowerShell is a task-based command-line shell and scripting language designed especially for system administration. It can execute commands and scripts authored by developers or system administrators, as long as they have some minimal development expertise. What makes Windows PowerShell a powerful framework for developers is its extensibility model, together with its capability to execute custom code. For example, from the Windows PowerShell console, you can not only administer a farm, but also create scripts for populating data into target lists of SharePoint. You can manage, create, and configure testing environments, and you can create custom scripts to deploy your solutions.

Developer tools

SharePoint developers can take advantage of some Microsoft-supplied tools to support their work and reduce the effort involved in developing custom solutions. This section lists these tools and identifies when they might be useful.

SharePoint Designer 2013

SharePoint Designer 2013 is a rapid application development (RAD) tool for developing SharePoint no-code solutions. You can download it for free from Microsoft's website, at *http://www.microsoft.com/download/details.aspx?id=35491*. SharePoint Designer 2013 targets advanced users, who can use it to design and compose solutions without writing any code. For example, using SharePoint Designer 2013, you can

- Personalize pages, page layouts, Web Parts, Web Part pages, layouts, and themes.
- Create and manage lists and document libraries.
- Design simple workflows or import workflows designed using Microsoft Visio 2010 or 2013.
- Manage content types and site columns to model typed lists of contents.
- Model and register external data sources using the Business Data Connectivity engine.
- Create pages with lists data bound to external data sources.
- Manage users and groups.
- Manage files and assets of the target site.

Figure 1-8 shows the main page of SharePoint Designer 2013 when connected to a SharePoint site. As you can see, it provides a user-friendly interface, consistent with the Office 2013 user experience.

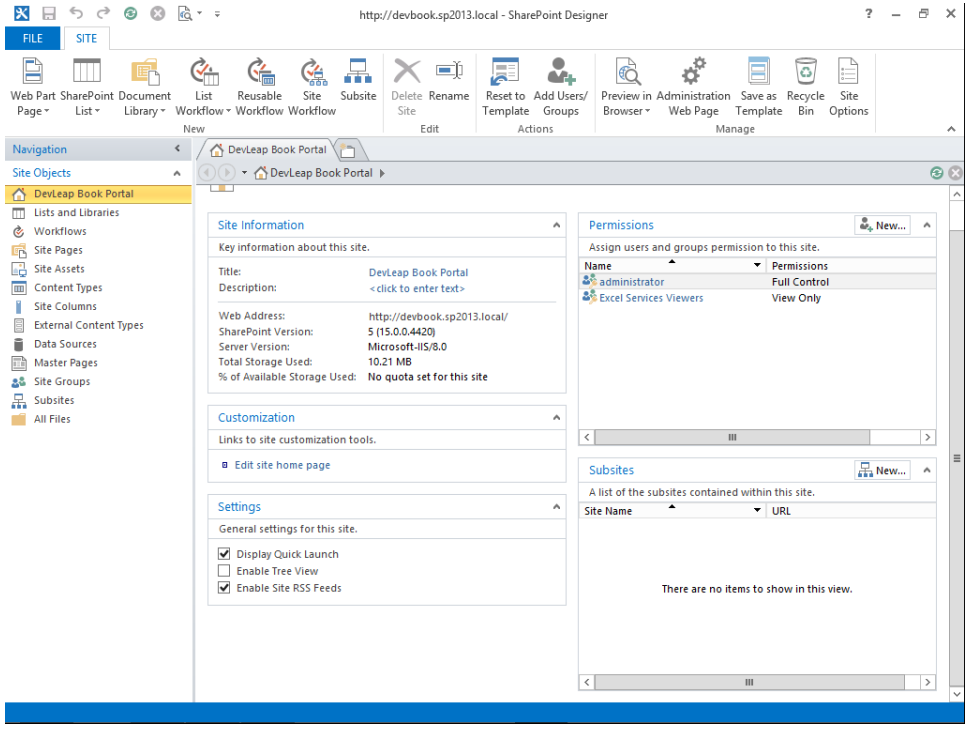


FIGURE 1-8 The SharePoint Designer 2013 main page.

As a developer, you will primarily use this tool to prototype solutions, to design Business Data Connectivity models, and to customize layouts—working with themes, master pages, XSLTs, and pages.

Note This book will not cover SharePoint Designer 2013 in depth, because it is aimed at developers who are willing to develop SharePoint solutions by writing custom code. For deep coverage of SharePoint Designer 2013, read *Microsoft SharePoint Designer 2013 Step by Step*, by Penelope Coventry (Microsoft Press, 2013).

Microsoft Visual Studio 2012

Visual Studio 2012 can be extended with a set of tools for developing SharePoint 2013 apps and solutions. These tools are named the *Microsoft Office Developer Tools for Visual Studio 2012* and can be installed through the Web Platform Installer kit or downloaded manually from MSDN. When you install Visual Studio 2012, you have also the opportunity to activate the SharePoint 2010 Developer Tools option, which installs a set of project and item templates that are ready to use in SharePoint solutions that target SharePoint 2010. Most of the code and projects you develop using the SharePoint 2010 developer tools are also supported by SharePoint 2013, for the sake of backward compatibility. Nevertheless, it is highly recommended to develop using the SharePoint 2013 tools and the new apps-oriented development model introduced in SharePoint 2013.

More Info The Microsoft Office Developer Tools for Visual Studio 2012 can be directly downloaded from the following URL: *http://msdn.microsoft.com/en-US/sharepoint/aa905690.aspx*.

The development tools for SharePoint also include some deployment tools, which are useful for packaging, releasing, and upgrading a SharePoint solution.

Note To use Visual Studio 2012 for developing SharePoint 2013 apps and solutions, you must run it under an administrative account, because you need some high-level permissions to manage the SharePoint servers while deploying solutions. In addition, you need to attach to the IIS worker process while debugging code. It is suggested to run your desktop as a standard user, but run Visual Studio 2012 with a Run As command to impersonate an administrative user. Moreover, to develop SharePoint solutions (WSPs), you need to have SharePoint installed on your development machine. On the contrary, to develop SharePoint apps, you do not need to have SharePoint on board, and you can remotely connect to an external SharePoint environment, including SharePoint Online on Office 365.

Figure 1-9 shows the Add New Project form of Visual Studio 2012, showing the project templates installed by the SharePoint extensions.

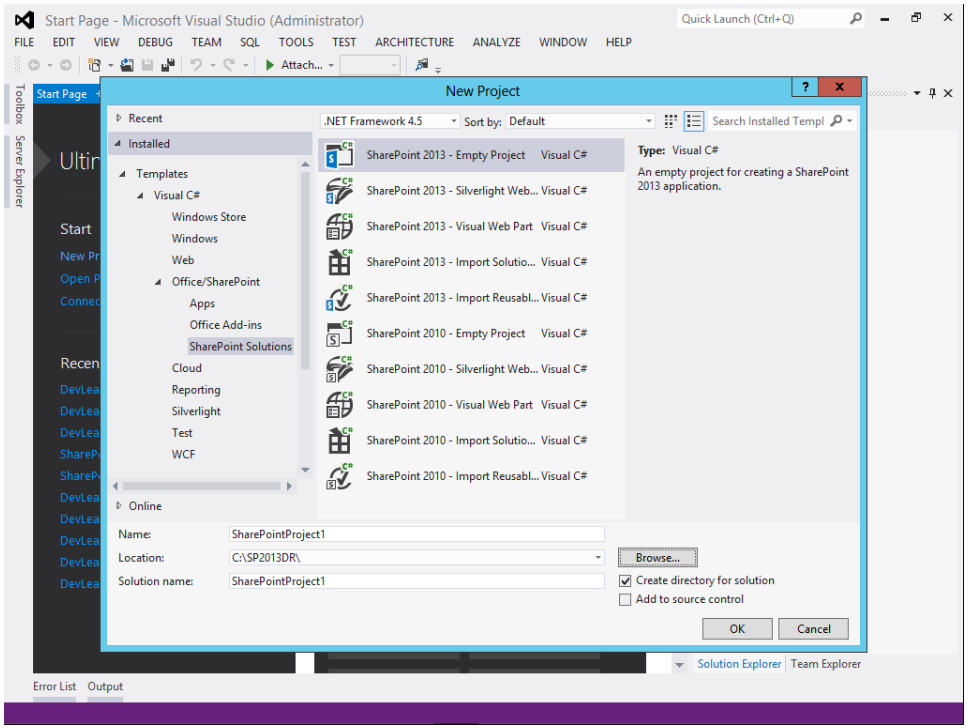


FIGURE 1-9 The Add New Project form in Visual Studio 2012.

You can create the following types of projects:

- **App for SharePoint 2013** This is the project template for creating a SharePoint 2013 app. It will be discussed in depth in Chapter 8, "SharePoint apps."
- **SharePoint 2013 Project** This is an empty project for starting a new SharePoint implementation. It provides a set of references to only the most useful libraries of SharePoint, and it provides support for automatic deployment.
- **SharePoint 2013 Silverlight Web Part** This is a project intended for developing a Web Part with a GUI based on Microsoft Silverlight.
- **SharePoint 2013 Visual Web Part** This is a project intended for developing a Web Part with a GUI based on an ASCX web control of ASP.NET.
- **Import SharePoint 2013 Solution Package** This imports an old or third-party solution package (WSP).
- **Import Reusable SharePoint 2013 Workflow** This project template is useful for importing workflows designed with SharePoint Designer 2013 that need to be extended or improved with Visual Studio 2012.

Regardless of which project template you start from, you can develop any of these extension types, because these models simply prepare a preconfigured environment. In fact, it's quite common to start with the App for SharePoint 2013 template or the SharePoint 2013 - Empty Project template, and then add items as you need them.

Microsoft Office Developer Tools for Visual Studio 2012 also provides a rich set of item templates for creating various types of content in SharePoint app projects. Here is a list of some of the main items:

- **List** This is for specifying a custom list of fields or creating a new list from an existing list template.
- **Remote Event Receiver** This allows you to handle SharePoint events using a remote service.
- **Content Type** This is for creating a reusable collection of fields and settings that you can apply to a SharePoint list.
- **Workflow** This allows you to create and deploy a workflow for SharePoint, based on the new workflow engine of SharePoint 2013.
- **Empty Element** This is an XML feature element for hosting files, pages, or any other customization, compliant with the features and elements schema available in SharePoint since version 2010.
- **Site Column** A site column item is useful for creating custom content types and list definitions.
- **Module** This is a module item for deploying files, pages, assets, and more on SharePoint.
- **Client Web Part (Host Web)** This is a client Web Part (App Part) for supporting a custom SharePoint app**.**
- **UI Custom Action (Host Web)** This is typically used in an app that adds a UI extension to its host site; for example, it can add an action to the ribbon or to a list menu.
- **Task Pane App** This is an app that appears in the task pane of an Office application.
- **Content App** This is an app that appears in the body of an Office document.

SharePoint Server Explorer

Another interesting feature offered by Visual Studio 2012 is SharePoint Server Explorer, an extension to Server Explorer in Visual Studio 2012 for targeting SharePoint servers. Through this extension, you can register as many SharePoint servers as you need and browse their topology and configuration using the classic tree-view approach, such as in Visual Studio Server Explorer windows.

As shown in Figure 1-10, the SharePoint Server Explorer interface lets you browse and manage the following:

- Sites and subsites
- Content types
- Features
- List templates
- Lists and document libraries
- Workflows

In addition, because SharePoint Server Explorer is based on an extensible object model, you can extend it to provide new functionalities, using Visual Studio 2012 to develop such solutions. You can already find many custom extensions that can be downloaded for free.

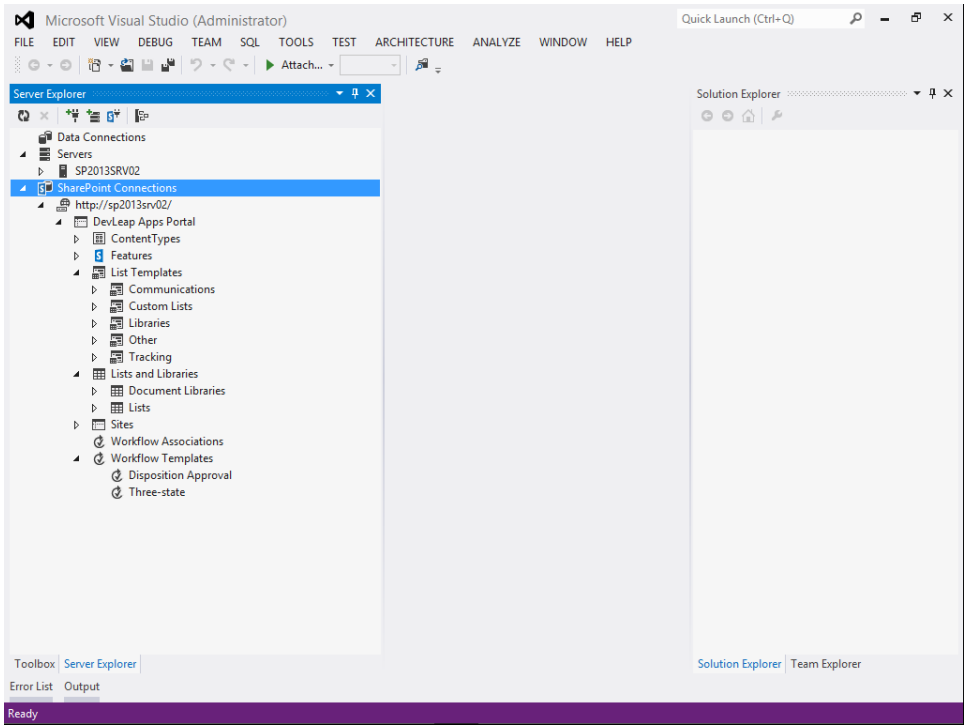


FIGURE 1-10 The SharePoint Server Explorer UI in Visual Studio 2012.

Solution Explorer and the Feature Designer

One last set of tools available in Visual Studio 2012 include Solution Explorer and the Feature Designer. These are tools for graphically designing and managing SharePoint packages (WSPs) and features. They are particularly useful for automating deployment of SharePoint solutions. You will learn more about these tools in Chapter 4.

Summary

This chapter explained what SharePoint is, what its main capabilities are, and how developers can take advantage of those capabilities. It described the product architecture and gave a quick comparison of the various SharePoint editions so that you can choose the one that best fits your needs. Finally, it covered the main tools available for developing SharePoint solutions.

CHAPTER 3

Data provisioning

The previous chapters showed you how many Microsoft SharePoint solutions rely on lists of items that contain data, such as contacts, files, and so on. When you develop a SharePoint solution, therefore, one of your main tasks is to provision data structures for these lists of items. In fact, whenever you need to develop a reusable and maintainable solution that will reside on many different site collections and has many different customers, you should formally define the data structures that you will use. Simply designing them through the SharePoint visual design interface from a web browser might seem easy (any end user can do it), but in the long run it will become a source of confusion. Formal definitions can be reused many times in multiple sites and can be versioned. Meanwhile, data structure definitions made manually through the visual design interface are difficult to reuse and can lead to duplication of definitions in multiple sites. Also, when you create SharePoint apps hosted on SharePoint, you can use the data model of lists and items provided by SharePoint for storing data and content related to your apps.

Note Within the context of this book, the term *data structure* refers to the formal definitions of custom list definitions, content types, and site columns. Such formal definitions help to ensure data consistency across lists and sites.

This chapter explores the rules for custom lists and the tools that SharePoint 2013 provides to create them. To learn how these tools behave in a real-world scenario, you will investigate how to define a custom list of contacts that can use custom forms and can be browsed through specific list views. The list in this case study will be based on two content types: *Customer* and *Supplier*.

Site columns

The first and main step in provisioning a custom data structure is to define site columns. A site column describes a reusable data type model that you can use in many different content types and list definitions, across multiple SharePoint sites. Unless you have never used SharePoint at all, you will have already defined many site columns using a web browser, within the appropriate section of the Site Settings page. To create a more flexible and reusable solution, you can also define a site column using some XML code, which in SharePoint is called a *feature element*.

More Info For further details about features and feature elements, read Chapter 11, "Developing Web Parts."

Listing 3-1 shows a very simple site column definition for a *Text* column that contains the company name of the sample contact.

LISTING 3-1 A simple site column defined in a feature element

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field
    ID="{A8F24550-55CD-4d34-A015-811954C6CE24}"
    Name="DevLeapCompanyName"
    StaticName="DevLeapCompanyName"
    DisplayName="Company Name"
    Type="Text"
    Group="DevLeap Columns" />
</Elements>
```

Aside from the *Elements* tag itself, which is simply a container element, the interesting part of the preceding column definition is the *Field* element. The most important feature of this element is the *ID* attribute, which is a globally unique identifier (GUID) that uniquely identifies the site column. You can use the *ID* attribute to reference this specific site column everywhere. Notice that you can create unique GUIDs by using the GUIDGEN tool provided with Microsoft Visual Studio 2012.

Listing 3-1 declares that the Company Name column will have an internal *Name* attribute of DevLeapCompanyName. *Name* is a required attribute, and like the *ID* attribute, it should also be unique, because it provides an alternative way to exclusively reference the column from code. In general, this example uses the developer's company name value as a prefix to better ensure the uniqueness of this name. The *Name* attribute value cannot contain spaces or any characters other than numbers (0 through 9) and letters (*a* through *z* and *A* through *Z*). Any other characters will be converted into the corresponding hexadecimal representation. For example, if you want to name a field Company Name, you must define it as *Company_x0020_Name*. If you want to name a field Revenue %, you must define it as *Revenue_x0020__x0025_*. The last thing to keep in mind is that the *Name* attribute cannot be longer than 32 characters.

The preceding site column definition also defines the optional *StaticName* attribute, which is another way of defining the internal name. The *StaticName* can be useful for referencing your field in custom code, regardless of the encoding used in the Name field. Finally, the site column definition defines the field's *DisplayName* attribute, whose value is the title that users

will see in their browsers. This last attribute can take advantage of the multilanguage support provided by Microsoft .NET in general, so declaring its value as a resource string reference ("$*Resources:<Assembly_Name>,<Resource_Name>;*") instead of an explicit value will result in a multilanguage value.

Why do you need three attributes to define field name types?

At first, using three attributes to define three kinds of names for a single field may seem redundant and overly complex, but each attribute serves a purpose. Consider this: the XML schema that we use as developers is also used internally by SharePoint to represent a site column. When you define a column using the web browser interface, SharePoint automatically determines the internal name (for instance, *Name* and *StaticName*) based on the name (which becomes the *display name*) that you give it, automatically converting any nonalphanumeric characters to their corresponding hexadecimal representations, and then trimming the resulting string to 32 characters for the *Name* attribute, leaving the *StaticName* attribute value as long as needed. If a site column with the same *Name* already exists, SharePoint appends a number to the name, using a zero-based index.

If you later change the *DisplayName* of the field, SharePoint will keep both the *StaticName* and the *Name* unchanged. That scheme gives your site column three different values for the three attributes: the *DisplayName*; the *StaticName*, which is simply the original *DisplayName* with hexadecimal conversion of nonalphanumeric characters; and the *Name*, with hexadecimal conversion of nonalphanumeric characters trimmed to 32 characters.

Lastly, using the SharePoint Server Object Model (for further details, see Chapter 5, "Server Object Model"), you can change the *StaticName*, but you cannot change the internal *Name* value. Therefore, when you have to define site columns using a feature element, the best practice is to assign the same value to the *Name* and to the *StaticName* (avoiding nonalphanumeric characters) and to provide a descriptive value for the *DisplayName* attribute.

The *Type* attribute is mandatory for site column definitions. It defines the data type assigned to the field. This *Type* attribute value can be one of a predefined set of SharePoint field types, or it can be a custom field type that you have defined and deployed. Table 3-1 presents some of the main field types provided by SharePoint.

More Info For a complete list of field types, refer to the online product reference at *http://msdn.microsoft.com/en-us/library/ms437580(v=office.15).aspx*.

TABLE 3-1 Common predefined field types

Field type name	Description
Boolean	Represents a *Boolean* value (*TRUE* or *FALSE*), stored as a *bit* in Microsoft SQL Server and accessible as an *SPFieldBoolean* object through the Server Object Model.
Choice	Allows the user to select a single value from a predefined set of values. The XML schema of the *Field* element must declare the values (for further details, see Listing 3-2). It is stored as an *nvarchar* in SQL Server, and is accessible as an *SPFieldChoice* object through the Server Object Model.
MultiChoice	Allows the user to select multiple values from a predefined set of values. The XML schema of the Field element has to declare the values. It is stored as an *ntext* in SQL Server, and is accessible as an *SPFieldMultiChoice* object through the Server Object Model.
Currency	Defines a currency value. *Currency* is bound to a specific locale, using an *LCID* attribute. It can have constraints using *Min, Max*, and *Decimals* attributes. It is stored as a *float* in SQL Server and is accessible as an *SPFieldCurrency* object through the Server Object Model.
DateTime	Saves a date and time value. *DateTime* is stored as a *datetime* in SQL Server, and is accessible as an *SPFieldDateTime* object through the Server Object Model.
Lookup and *LookupMulti*	Behave almost the same as *Choice* and *MultiChoice*; however, the set of values to choose from is taken from another list of items within the same site. These field types are stored as *int type*s in SQL Server, and are accessible as *SPFieldLookup* objects through the Server Object Model.
Note	Stores multiple lines of text. *Note* is stored as an *ntext* in SQL Server, and is accessible as an *SPFieldMultiLineText* object through the Server Object Model.
Number	Defines a floating-point number. *Number* can have constraints using *Decimals*, *Div, Max, Min, Mult*, and *Percentage*. It is stored as a *float* in SQL Server and is accessible as an *SPFieldNumber* object through the Server Object Model.
Text	Describes a single line of text of a configurable maximum length. *Text* is stored as an *nvarchar* in SQL Server, and is accessible as an *SPFieldText* object through the Server Object Model.
URL	Defines a URL with a specific *LinkType* (*Hyperlink* or *Image*). *URL* is stored as an *nvarchar* in SQL Server and is accessible as an *SPFieldUrl* object through the Server Object Model.
User and *UserMulti*	Describe a lookup for a single user or a set of users. These are stored as an *int type*s in SQL Server, and are accessible as *SPFieldUser* objects through the Server Object Model.

The last attribute defined in the site column example is the *Group* attribute, which simply defines a group membership to make it easier to find custom fields through the web browser administrative interface. *Group* is an optional attribute, but it is better that you define it whenever you create a custom site column, in order to organize your columns in personalized custom groups.

Although it's not an exhaustive keyword reference, Table 3-2 shows some of the many other interesting attributes that you can use when defining custom site columns. For a complete reference of the available attributes, you can read the following page on MSDN: *http://msdn.microsoft.com/en-us/library/aa979575.aspx*.

TABLE 3-2 Interesting optional *Boolean* attributes available for the *Field* element

Field attribute	Description
Hidden	Can assume a value of *TRUE* or *FALSE*. When *TRUE*, the field will be completely hidden from the UI and will be accessible only through code, using the Object Model.
ReadOnly	Can assume a value of *TRUE* or *FALSE*. When *TRUE*, the field will not be displayed in *ne*w and edit forms, but can be included in read-only data views. It will remain accessible using the object model.
Required	Can assume a value of *TRUE* or *FALSE*. Its name implies its role.
RichText	Can assume a value of *TRUE* or *FALSE*. It determines whether a text field will accept rich text formatting.
ShowInDisplayForm	Can assume a value of *TRUE* or *FALSE*. When *FALSE*, the field will not be displayed in the display form of the item containing the field.
ShowInEditForm	Can assume a value of *TRUE* or *FALSE*. When *FALSE*, the field will not be displayed in the editing form of the item containing the field.
ShowInNewForm	Can assume a value of *TRUE* or *FALSE*. If it is *FALSE*, the field will not be displayed in the form to add a new item containing the field.

While Listing 3-1 introduced a basic definition, Listing 3-2 adds another level of complexity by declaring a Choice field that will be used to select the contact's country affiliation.

LISTING 3-2 A Choice site column defined in a feature element

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field
    ID="{149BF9A1-5BBB-468d-AA35-91ACEB054E3B}"
    Name="DevLeapCountry"
    StaticName="DevLeapCountry"
    DisplayName="Country"
    Type="Choice"
    Group="DevLeap Columns"
    Sortable="TRUE">
      <Default>Italy</Default>
      <CHOICES>
        <CHOICE>Italy</CHOICE>
        <CHOICE>USA</CHOICE>
        <CHOICE>Germany</CHOICE>
        <CHOICE>France</CHOICE>
      </CHOICES>
  </Field>
</Elements>
```

This example shows how you can define a set of available values for a *Choice* field. Note that the list defines a *Default* element.

Another interesting task that you can accomplish when defining a site column is to declare a custom validation rule for its content. To do that, you simply define a *Validation* element as a child of the *Field* definition. The *Validation* element can have a *Message* attribute, which defines an error message to display to end users when validation fails, and a *Script* attribute, which defines a JavaScript rule that performs the validation. Alternatively, you can define a rule using the *Formulas* syntax of SharePoint, putting the rule inside the *Validation* element.

More Info For further details on calculated fields and formulas in SharePoint, refer to the "Calculated Field Formulas" MSDN page, at http://msdn.microsoft.com/en-us/library/bb862071.aspx.

Content types

A content type schema defines a model for a specific SharePoint complex data type, and is based on a set of site column references, together with some other optional information related to forms, rendering templates, a specific document template (only in the case of document items), and custom XML configuration.

Chapter 2, "SharePoint data fundamentals," showed how SharePoint uses a hierarchical structure for defining content types, which consists of a base content type named *System* with a single child named *Item*. SharePoint then applies an inheritance paradigm (similar to object-oriented class inheritance) to define each content type descendant of *Item*. Figure 3-1 shows an excerpt of the hierarchical inheritance tree for native content types. As a consequence of this behavior, you must define inheritance information for each new content type that you declare. For more details, read the "Content type IDs" section later in the chapter.

Listing 3-3 provides an example of the *Contact* content type, defined by referencing a set of site columns.

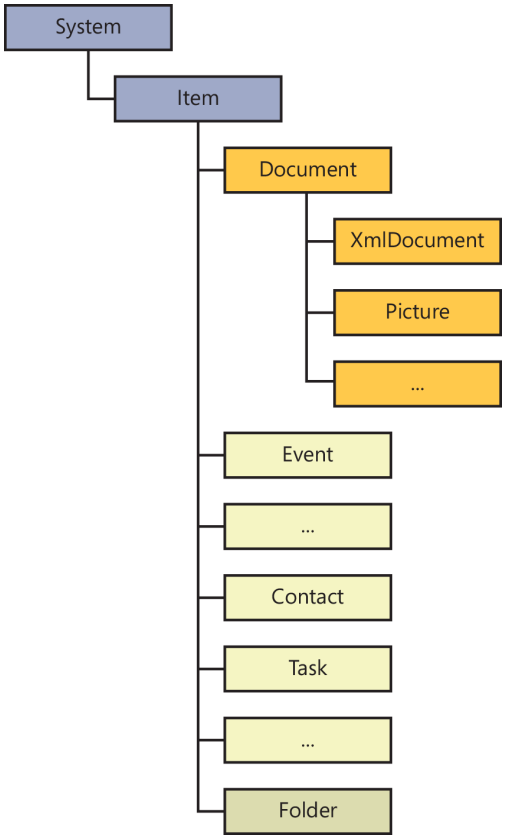


FIGURE 3-1 The content types inheritance hierarchy in SharePoint.

LISTING 3-3 A simple content type defined in a feature element, together with its site columns

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- Site Columns used by the Content Type -->
  <Field
    ID="{C7792AD6-F2F3-4f2d-A7E5-75D5A8206FD9}"
    Name="DevLeapContactID"
    StaticName="DevLeapContactID"
    DisplayName="Contact ID"
    Type="Text"
    Group="DevLeap Columns"
    Sortable="TRUE" />
```

```
  <Field
    ID="{A8F2455O-55CD-4d34-AO15-811954C6CE24}"
    Name="DevLeapCompanyName"
    StaticName="DevLeapCompanyName"
    DisplayName="Company Name"
    Type="Text"
    Group="DevLeap Columns"
    Sortable="TRUE" />
  <Field
    ID="{149BF9A1-5BBB-468d-AA35-91ACEBO54E3B}"
    Name="DevLeapCountry"
    StaticName="DevLeapCountry"
    DisplayName="Country"
    Type="Choice"
    Group="DevLeap Columns"
    Sortable="TRUE">
      <Default>Italy</Default>
      <CHOICES>
        <CHOICE>Italy</CHOICE>
        <CHOICE>USA</CHOICE>
        <CHOICE>Germany</CHOICE>
        <CHOICE>France</CHOICE>
      </CHOICES>
  </Field>
  <!-- Parent ContentType: Item (0x01) -->
  <ContentType ID="0x010OA60F69C4B1304FBDA6C4B4A25939979F"
               Name="DevLeapContact"
               Group="DevLeap Content Types"
               Description="Base Contact of DevLeap"
               Inherits="TRUE"
               Version="O">
    <FieldRefs>
      <FieldRef
        ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}"
        Name="Title"
        DisplayName="Full name" />
      <FieldRef
        ID="{C7792AD6-F2F3-4f2d-A7E5-75D5A820GFD9}"
        Name="DevLeapContactID"
        DisplayName="Contact ID"
        Required="TRUE" />
      <FieldRef
        ID="{A8F2455O-55CD-4d34-AO15-811954C6CE24}"
        Name="DevLeapCompanyName"
        DisplayName="Company Name" />
      <FieldRef
        ID="{149BF9A1-5BBB-468d-AA35-91ACEBO54E3B}"
        Name="DevLeapCountry"
        DisplayName="Country" />
    </FieldRefs>
  </ContentType>
</Elements>
```

This feature element example contains a *ContentType* element, which defines some descriptive information, such as the *Name*, *Group*, and *Description*. The *ContentType* element also defines a *Version* attribute, which indeed is used for managing versioning, as its name implies, but is still reserved by Microsoft for future use. Last, but most important, is the *ID* attribute, which defines the unique identifier for this content type in the site collection where it is defined. Inside the *ContentType* element is a *FieldRefs* element, which is the parent of a list of *FieldRef* or *RemoveFieldRef* elements. Each element in this list references a specific site column to be added or removed from this content type. You might notice that this example references all the site columns defined earlier in the feature element file. In fact, unless you are defining site columns for use in multiple content types, it's common to define the referenced site columns within the same feature element file—just before the content type that will use them.

Listing 3-3 also references a site column with the name Title and the ID *{fa564e0f-0c70-4ab9-b863-0177e6ddd247}*. This is the SharePoint native site column that defines the Title field for each SharePoint item. In the content type example, we changed the *DisplayName* value from *Title*, which still retains its internal name, to *Full name*, which will be the displayed name for this content type. By default, the *Title* field is also used by SharePoint to render the Edit Control Block menu, which allows you to display, edit, and manage a list item from the list UI.

Content type IDs

The *ID* attribute of a content type is not a simple GUID, as it was with the site columns definition; instead, it's a more complex value that describes the hierarchical inheritance of the type. In fact, every content type ID is composed of the *ID* of its hierarchical parent content type, followed by a hexadecimal value that's unique to the current content type. You could say that a content type ID defines its genealogy. This logic is recursive, starting with the *System* content type and extending all the way down to the current content type. Table 3-3 shows an excerpt of the base hierarchy of SharePoint content type IDs.

TABLE 3-3 An excerpt of the base hierarchy of SharePoint content type IDs

Content type	ID
System	*0x*
Item	*0x01*
Document	*0x0101*
XmlDocument	*0x010101*
Picture	*0x010102*
Event	*0x0102*
...	
Contact	*0x0106*
Task	*0x0108*
...	
Folder	*0x0120*

Table 3-3 demonstrates that the root content type is *System*, which is a special hidden content type with an *ID* value of *0x*. The *Item* content type is the only child of *System* and has an *ID* value of *0x01* (the *System ID + 01*). The *Document* content type, which is a child of *Item*, has an *ID* value of *0x0101* (the *Item ID + 01*), while its sibling *Event* has an *ID* of *0x0102* (the *Item ID + 02)*.

In general, the rule used to define content type IDs states that you can build an ID using either of two techniques:

- Parent content type ID + two hexadecimal values (cannot be *00*)
- Parent content type ID + *00* + hexadecimal GUID

Microsoft generally uses the first technique to define base content type IDs. Third parties, such as vendors or ISVs, typically use the latter technique to define custom content type IDs. If you want to define a hierarchy of custom content types of your own, follow these steps:

1. Identify the base content type from which you want to inherit.
2. Add 00 at the end of the base content type ID.
3. Add a hexadecimal GUID just after the 00.
4. Append two hexadecimal values to declare every specific child of your content type.

As a concrete example, suppose that you want to define a custom content type inherited from the *Document* base content type. You would start with *0x0101*, which is the *Document ID*, append *00* to it, and then append a hexadecimal GUID, making your *ID* something like *0x010100BDD3EC87EA65463AB9FAA5337907A3ED*.

If you wanted to use your custom content type as a base for some other inherited content types, you would append *01*, *02*, and so on for each child content type, as in the following:

- **Base ID** *0x010100BDD3EC87EA65463AB9FAA5337907A3ED*
- **Child 1** *0x010100BDD3EC87EA65463AB9FAA5337907A3ED01*
- **Child 2** *0x010100BDD3EC87EA65463AB9FAA5337907A3ED02*

More Info Content type IDs have a maximum length of 512 bytes. Because every two hexadecimal characters correspond to a single byte, a content type ID has a maximum length of 1,024 characters.

With that in mind, we can go back to the example custom *Contact* content type. First, you need to choose the base content type from which you want to inherit. For example purposes, assume that you decide to use the generic base *Item* as the parent content type. That means the custom content type ID will start with *0x01*, followed by *00* and then a hexadecimal GUID. The end result is the same as the ID highlighted in bold in Listing 3-3:

```
ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F"
```

The goal of the case study is to define a custom list that is based on a couple of content types (*Customer* and *Supplier*) inherited from this base *Contact* content type. Listing 3-4 shows the definitions of the *Customer* and *Supplier* content types.

LISTING 3-4 *Customer* and *Supplier* content type definitions

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field
      ID="{AC689935-8E8B-485e-A45E-FF5A338DD92F}"
      Name="DevLeapCustomerLevel"
      StaticName="DevLeapCustomerLevel"
      DisplayName="Customer Level"
      Type="Choice"
      Group="DevLeap Columns">
    <Default>Level C</Default>
    <CHOICES>
      <CHOICE>Level A</CHOICE>
      <CHOICE>Level B</CHOICE>
      <CHOICE>Level C</CHOICE>
    </CHOICES>
  </Field>
  <Field
      ID="{A73DE518-B9B9-4e8d-9D94-6099B4603997}"
      Name="DevLeapSupplierAccount"
      StaticName="DevLeapSupplierAccount"
      DisplayName="Supplier Account"
      Type="User"
      Group="DevLeap Columns"
      Sortable="TRUE" />
  <ContentType ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F01"
               Name="DevLeapCustomer"
               Group="DevLeap Content Types"
               Description="Customer of DevLeap"
               Version="0">
    <FieldRefs>
      <FieldRef
          ID="{AC689935-8E8B-485e-A45E-FF5A338DD92F}"
          Name="DevLeapCustomerLevel"
          Required="TRUE" />
    </FieldRefs>
  </ContentType>
  <ContentType ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F02"
               Name="DevLeapSupplier"
               Group="DevLeap Content Types"
               Description="Supplier of DevLeap"
               Version="0">
    <FieldRefs>
      <FieldRef
          ID="{A73DE518-B9B9-4e8d-9D94-6099B4603997}"
          Name="DevLeapSupplierAccount"
          Required="TRUE" />
    </FieldRefs>
  </ContentType>
</Elements>
```

Both of these content types extend the base *Contact* content type; each adds a specific site column. The *Customer* content type adds a required field to define the *customer level* (*A*, *B*, or *C*) for each *Customer* instance, while the *Supplier* content type adds a field to reference a local *account*, which you can browse as a SharePoint user. You can see the inheritance hierarchy of these custom types in Figure 3-2, which shows a portion of the Site Content Type page of a site collection.

Site Content Type	Parent	Source
DevLeap Content Types		
DevLeapContact	Item	DevLeap Book Portal
DevLeapCustomer	DevLeapContact	DevLeap Book Portal
DevLeapInvoice	Document	DevLeap Book Portal
DevLeapSupplier	DevLeapContact	DevLeap Book Portal

FIGURE 3-2 The Site Content Type page of a site collection where the custom content types are provisioned.

Finally, consider that Visual Studio 2012 automatically calculates the content type IDs when you add a new content type to a SharePoint project. In fact, if you try to add a content type to a SharePoint project within Visual Studio 2012, you will be prompted with a one-step wizard, regardless of whether you are creating a Windows SharePoint Services Solution Package (WSP) or a SharePoint app. In the wizard's first and only step, you must choose the basic content type from which you would like your custom content type to inherit (Figure 3-3).

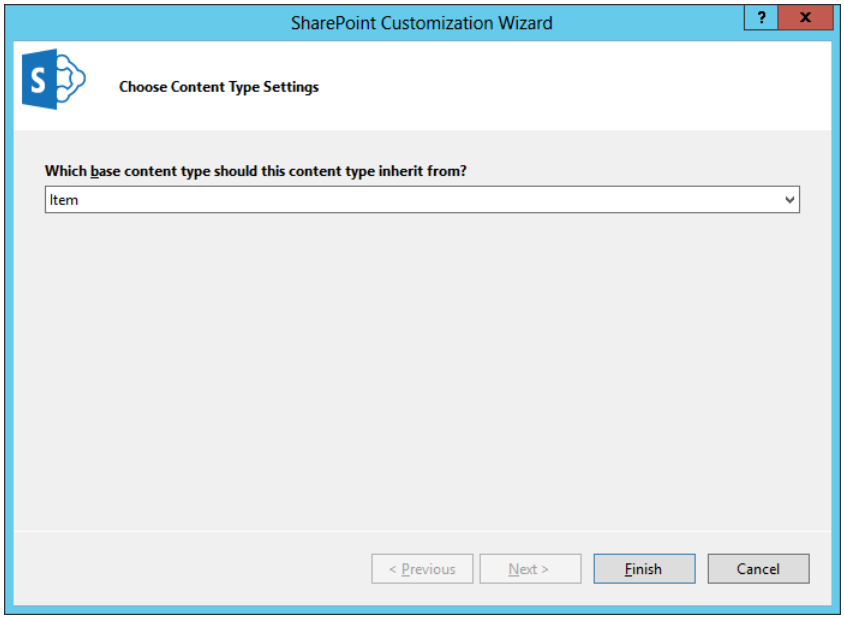


FIGURE 3-3 The wizard for creating a new content type.

After you make your choice and click finish to close the wizard, SharePoint displays a graphical designer useful to define the columns of the content type and its overall configuration. Figure 3-4 shows the two tabs available in the Content Type designer: Columns and Content Type.

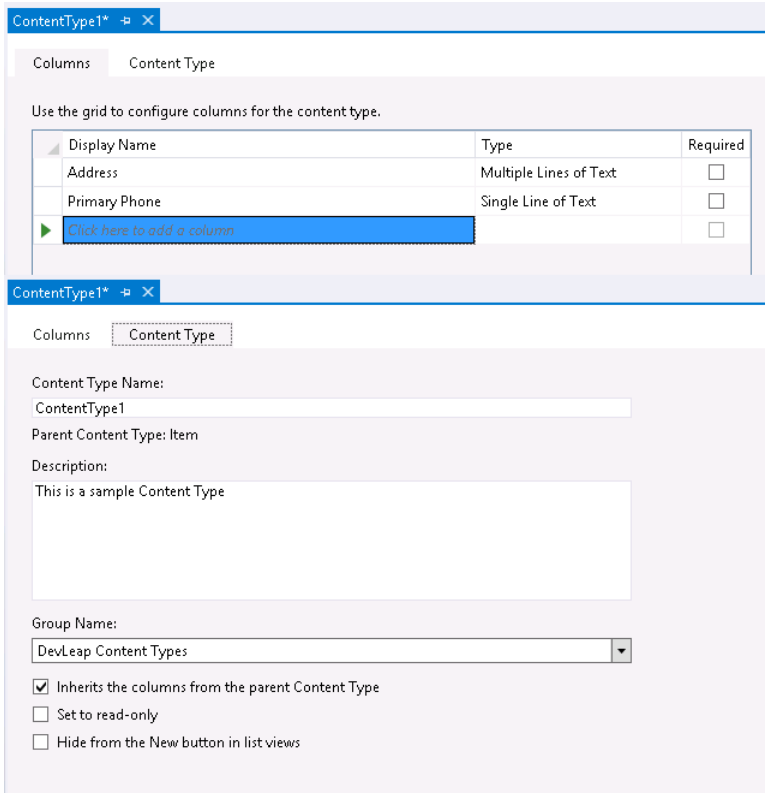


FIGURE 3-4 The two tabs available in the Content Type designer.

As you can see, the Columns tab is active. Here you can reference the site columns to use in the current content type. Note, however, that you can specify existing site columns only. The Content Type tab enables you to define the name, the description, and the group of the current content type. Lastly, through this second tab you can also determine whether the content type will inherit columns from its parent type or not, as well as if the current type will be read-only and/or hidden. Based on your settings, the designer creates an XML element manifest file that is similar to what you can code manually. Although this might seem like a worthwhile shortcut, it is somewhat limited. When you need a finer degree of flexibility in defining custom content types, manually creating or editing the XML file is a better solution.

More about content types

Sometimes you need a more restricted content type; in such cases, SharePoint offers several other interesting attributes to help you out. For example, the *ReadOnly* attribute makes the content type read-only when its value is set to *TRUE*. Likewise, when the *Sealed* attribute is set to *TRUE*, it seals a content type so that only a site collection administrator using the Server Object Model can unseal it for editing. Lastly, the *Hidden* attribute is useful for making a content type invisible so that contributors cannot create new items of this type in list views, but you will still have access to it through your

custom code. If you want to declare a content type as completely invisible—not only for end users but also for site collection administrators—you can make it belong to a special group named *_Hidden*.

In addition, you can configure a content type not only through *ContentType* element attributes, but also by declaring some child elements. One of these is the *FieldRefs* child element discussed earlier in this chapter. Another useful element is *XmlDocuments*, with which you can define any kind of custom XML configuration to apply to the content type. SharePoint itself uses this element to declare custom controls and pages for the content type. Listing 3-5 shows how to use this element.

LISTING 3-5 Using the *XmlDocuments* element inside a content type definition

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ContentType ID="0x0100a60f69c4b1304fbda6c4b4a25939979f01"
               Name="DevLeapCustomer"
               Group="DevLeap Content Types"
               Description="Customer of DevLeap"
               Inherits="TRUE"
               Version="0">
    <FieldRefs>
      <FieldRef
        ID="{AC689935-8E8B-485e-A45E-FF5A338DD92F}"
        Name="DevLeapCustomerLevel"
        Required="TRUE" />
    </FieldRefs>
    <XmlDocuments>
      <XmlDocument NamespaceURI=
       "http://schemas.microsoft.com/sharepoint/v3/contenttype/forms">
        <FormTemplates xmlns=
          "http://schemas.microsoft.com/sharepoint/v3/contenttype/forms">
          <Display>DevLeapCustomerDisplay</Display>
          <Edit>DevLeapCustomerEdit</Edit>
          <New>DevLeapCustomerNew</New>
        </FormTemplates>
      </XmlDocument>
    </XmlDocuments>
  </ContentType>
</Elements>
```

Listing 3-5 shows that the *XmlDocuments* element is just a container for one or more *XmlDocument* elements. Every *XmlDocument* element can have a *NamespaceURI* attribute that declares the scope of the custom configuration defined. Listing 3-5 declares a configuration that defines custom ASCX control files that are used for rendering display, edit, and add forms for instances of the current content type. The ASCX control files referenced should be deployed inside

the CONTROLTEMPLATES special folder of SharePoint, through a farm-level (full-trust) solution. The content of each *XmlDocument* element derives from the referenced *NamespaceURI*. The only requirement is that the XML content must be valid against its declared XML schema.

When you consider that in a farm-level (full-trust) solution you can access any custom *XmlDocument* that you define while provisioning content types later through the Server Object Model, you can see that the model provides you with an extremely customizable environment.

Document content types

Content types inherited from the *Document* base content type (ID: *0x0101*) are a special case that you must analyze a bit more carefully than usual. In fact, every document has numerous specific configurations that it must handle. For instance, in the "Content types" section earlier in the chapter, you learned that a document can have a document template, a document information panel, or both.

Listing 3-6 shows the definition for a custom document content type that declares an *Invoice* document model.

LISTING 3-6 Defining the *Invoice* content type, inherited from the *Document* content type

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- Parent ContentType: Document (0x0101) -->
  <ContentType ID="0x0101
00A5FD8267A91945DF9F3884D9EAA4F12F"
               Name="DevLeapInvoice"
               Group="DevLeap Content Types"
               Description="Invoice of DevLeap"
               Inherits="TRUE"
               Version="0">
    <FieldRefs>
      <!-- Field References here -->
    </FieldRefs>
    <DocumentTemplate TargetName="Forms/DevLeapInvoiceTemplate.dotx" />
  </ContentType>
</Elements>
```

The *Document* portion of the ID is highlighted in bold to remind you of the underlying behavior of SharePoint. The *DocumentTemplate* element (also highlighted) has a *TargetName* attribute that defines the URL (relative for the site collection) of the template item to use for every new *Invoice* instance. Listing 3-7 shows how to define a custom document information panel for a *Document* content type, assuming that you have already designed and deployed the panel.

LISTING 3-7 Defining a custom document information panel for an *Invoice* content type, inherited from the *Document* content type

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- Parent ContentType: Document (0x0101) -->
  <ContentType ID="0x010100a5fd8267a91945df9f3884d9eaa4f12f"
               Name="DevLeapInvoice"
               Group="DevLeap Content Types"
               Description="Invoice of DevLeap"
               Inherits="TRUE"
               Version="0">
    <FieldRefs>
      <!-- Field References here -->
    </FieldRefs>
    <XmlDocuments>
      <XmlDocument NamespaceURI=
        "http://schemas.microsoft.com/office/2006/metadata/customXsn">
        <xsnLocation>http://URL/customXsn.xsn</xsnLocation>
        <cached>False</cached>
        <openByDefault>True</openByDefault>
        <xsnScope>http://URL/documentLibrary</xsnScope>
      </XmlDocument>
    </XmlDocuments>
  </ContentType>
</Elements>
```

Listing 3-7 declares the absolute URL of the document information panel by using the *xsnLocation* element. It also disables caching in the Microsoft Office client by setting the *cached* element to *FALSE*. Lastly, it defines how the document should behave relative to this new panel, through the *openByDefault* element, which is set to *TRUE*, meaning that the panel should open by default. The *xsnScope* element is required, but for now it is reserved by Microsoft for internal use only.

List definitions

Now that you have defined your content types, you are ready to use them in a real list of contacts, comprising customers and suppliers. In fact, generally, whenever you define a set of custom content types, you also define one or more list definitions that use these content types. A *list definition* is simply a formal representation, using an XML schema, of a list data model from which you are able to create one or more instances of items corresponding to that model.

In SharePoint, a list definition is a combination of two files: a Schema.xml file, which defines the data structure and configuration of the list definition model, and a feature element file that describes the *ListTemplate*, which defines the information required for provisioning and deploying the list definition model.

List schema file

The list schema file is an XML document that describes all the metadata for the list data structure. The following are the main areas of the Schema.xml file for a list definition:

- **Content Types** This section defines the content types that will be available within the list definition.
- **Fields** This section declares the list-level site columns, which correspond to the entire set of site columns referenced by all the content types associated with the list definition.
- **Views** This section defines the views that will be available to the end user for navigating among the items of list template instances.
- **Forms** This section declares the ASPX pages that will be provided to the end user to add, display, and update items of a list instance based on the current list definition.
- **Validation** This section defines the validation rules for list items.
- **Toolbar** This section declares the type of toolbar that must be provided in the browser interface.

In addition to the preceding list, the complete XML schema contains some other elements as well. Listing 3-8 shows an excerpt from a Schema.xml file that describes a list definition, together with these main sections.

LISTING 3-8 Excerpt of a list definition schema file

```
<?xml version="1.0" encoding="utf-8"?>
<List xmlns:ows="Microsoft SharePoint"
    Title="DevLeapContacts"
    FolderCreation="FALSE"
    Direction="$Resources:Direction;"
    Url="Lists/DevLeapContacts"
    BaseType="0"
    EnableContentTypes="TRUE"
    xmlns="http://schemas.microsoft.com/sharepoint/">
    <MetaData>
        <ContentTypes>
        <!-- Here are referenced the content types -->
        </ContentTypes>
        <Fields>
        <!-- Here are declared the list-level site columns -->
        </Fields>
        <Views>
        <!-- Here are defined the views -->
        </Views>
        <Forms>
        <!-- Here are declared the forms used to add, display, update items -->
        </Forms>
        <Validation>
        <!-- Here are declared the validation rules for list items -->
        </ Validation >
        <Toolbar />
        <!-- To define what kind of toolbar to use in the Web browser UI  -->
    </MetaData>
</List>
```

The *List* element

The *List* element is the root of the schema file and declares some basic attributes for the list definition. The *Title* attribute defines the name of the list definition. The *BaseType* attribute defines the base list type to use for the current list definition. The global onet.xml file of SharePoint (for further details, please read Chapter 13, "Web templates") declares the list of all the available integer values for the *BaseType* values within a *BaseTypes* element.

Note The global onet.xml file is located in the SharePoint15_Root\TEMPLATE\GLOBAL\XML folder.

The available *BaseTypes* values are

- **0** Generic/Custom List
- **1** Document Library
- **2** Not used, may be reserved for future use
- **3** Discussion Forum (deprecated, use 0 instead)
- **4** Vote or Survey
- **5** Issues List

For example, Listing 3-8 used a *BaseType* with a value of *0* because we are defining a generic/custom list definition. The *Url* attribute is optional and defines the path to the root directory containing any ASPX file specific for the list definition. The *FolderCreation* attribute is also optional, and informs SharePoint whether to show (*TRUE*) or not show (*FALSE*) the New Folder command on the list toolbar. Finally, the *Direction* attribute is optional and declares the reading direction: *RTL* (right to left) or *LTR* (left to right). In Listing 3-8, the *Direction* value is read from a resource string so that the list will be compliant with the current locale settings of the site collection. Lastly, to make the users aware of the existence of the different available content types (*Contact*, *Customer*, and *Supplier*) when they are creating new items, we need to explicitly enable content types on the list definition, setting the *EnableContentTypes* attribute to a value of *TRUE*. There are many other attributes available for the *List* definition element; Table 3-4 shows some of them.

More Info For a complete reference of all the available attributes for the List element, refer to the official product documentation on MSDN, at *http://msdn.microsoft.com/en-us/library/ms415091(v=office.15).aspx*.

TABLE 3-4 Some of the main attributes for the *List* element of a Schema.xml list definition file

Attribute	Description
DisableAttachments	Optional *Boolean* value to disable attachments on the list.
EnableMinorVersions	Optional *Boolean* value that controls versioning with major and minor version of items.
ModeratedList	Optional *Boolean* value to enable content approval on inserted items.
PrivateList	Optional *Boolean* value to specify that the list is private.
VersioningEnabled	Optional *Boolean* value to enable versioning on the list. This value can be changed when creating a list instance.

The *MetaData* element

The main child element of *List* is the *MetaData* element, which wraps all the other elements in the Schema.xml file.

One of the main child nodes of *MetaData* is the *ContentTypes* element. This element declares the entire list of content types referenced by the current list definition. Listing 3-9 declares the *ContentTypes* element for the custom Contacts list.

LISTING 3-9 The *ContentTypes* section of metadata for the sample list definition

```
<ContentTypes>
  <ContentType
    ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F"
    Name="DevLeapContact"
    Group="DevLeap Content Types"
    Description="Base Contact of DevLeap"
    Inherits="TRUE" Version="0" Hidden="TRUE">
    <FieldRefs>
      <FieldRef ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}"
                Name="Title" DisplayName="Full name" Required="TRUE" />
      <FieldRef ID="{C7792AD6-F2F3-4f2d-A7E5-75D5A8206FD9}"
                Name="DevLeapContactID" DisplayName="Contact ID"
                Required="TRUE" />
      <FieldRef ID="{A8F24550-55CD-4d34-A015-811954C6CE24}"
                Name="DevLeapCompanyName" DisplayName="Company Name" />
      <FieldRef ID="{149BF9A1-5BBB-468d-AA35-91ACEB054E3B}"
                Name="DevLeapCountry" DisplayName="Country" />
    </FieldRefs>
  </ContentType>
  <ContentType
    ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F01"
    Name="DevLeapCustomer"
    Group="DevLeap Content Types"
    Description="Customer of DevLeap"
    Inherits="TRUE" Version="0">
    <FieldRefs>
      <FieldRef ID="{AC689935-8E8B-485e-A45E-FF5A338DD92F}"
                Name="DevLeapCustomerLevel" Required="TRUE" />
    </FieldRefs>
    <XmlDocuments>
      <XmlDocument NamespaceURI=
        "http://schemas.microsoft.com/sharepoint/v3/contenttype/forms">
        <FormTemplates xmlns=
          "http://schemas.microsoft.com/sharepoint/v3/contenttype/forms">
          <Display>DevLeapCustomerDisplay</Display>
          <Edit>DevLeapCustomerEdit</Edit>
          <New>DevLeapCustomerNew</New>
        </FormTemplates>
      </XmlDocument>
    </XmlDocuments>
  </ContentType>
```

```
  <ContentType
    ID="0x0100A60F69C4B1304FBDA6C4B4A25939979F02"
    Name="DevLeapSupplier"
    Group="DevLeap Content Types"
    Description="Supplier of DevLeap"
    Inherits="TRUE" Version="0">
    <FieldRefs>
      <FieldRef ID="{A73DE518-B9B9-4e8d-9D94-6099B4603997}"
                Name="DevLeapSupplierAccount" Required="TRUE" />
    </FieldRefs>
  </ContentType>
</ContentTypes>
```

Listing 3-9 defines all the content types already defined in the previous section, repeating their IDs to link these copies to the original definitions. Why repeat these declarations instead of simply referencing them in some way—such as by just linking their IDs, for example? During a content type's lifetime, its structure might change. To prevent and avoid any data loss, SharePoint copies content type definitions inside the list definitions that use them. Doing so preserves data models and data instances even if someone later changes them. Imagine what would happen if you had a simple content type reference rather than a copy; if you were to provision a *Customer* content type and use it in a custom list, then a few months later, when you have thousands of customer instances in your list, you delete a column from the *Customer* content type—or worse, you delete the entire content type! Having a complete copy of the content type definition allows SharePoint to maintain your data, even when the original content type changes or is removed.

On the other hand, whenever you want to make a change to one of your provisioned content types and you want that change applied to every instance in a site collection, you need to explicitly force the update through the browser-based content type administration page, through code using the Server Object Model, or by manually updating any references in the provisioned XML files, including the Schema.xml files for list definitions.

Listing 3-9 defines all three content types (*Contact*, *Customer*, and *Supplier*) and declares the base *Contact* as hidden, which forces users to explicitly create *Customer* or *Supplier* instances.

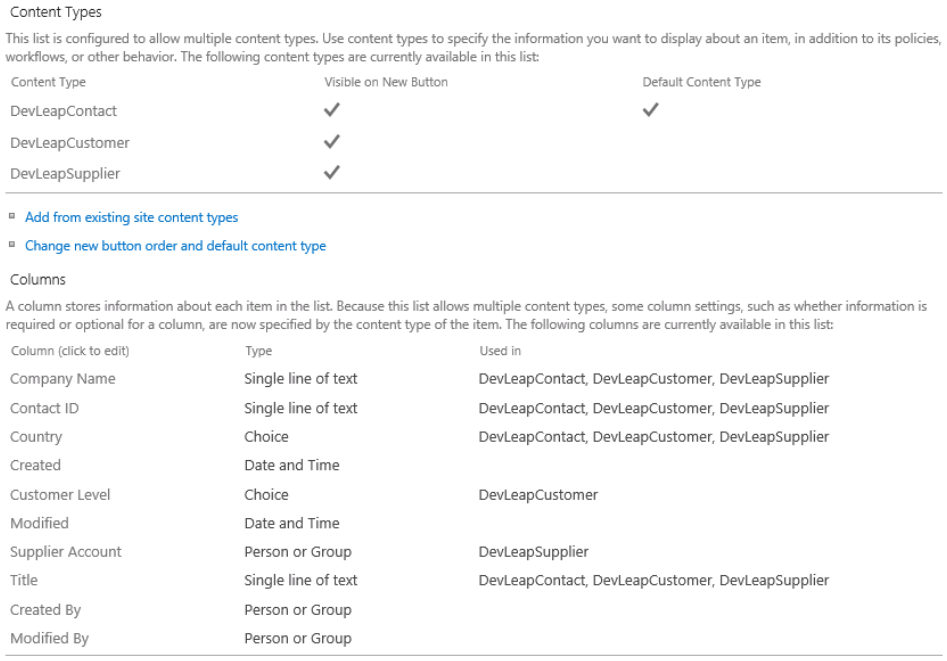
Another child of *MetaData* is the *Fields* element. It defines the list-level columns used to store metadata of item instances. These list-level columns are almost the same as the site columns defined in the first section of this chapter. Once again, their definitions are duplicated rather than referenced, and for the same reason: to support changes of the models without data loss during the site columns' lifetimes. The *Fields* section of the list definition contains all the columns used by any of the content types declared in the same Schema.xml file. Listing 3-10 shows the *Fields* element declared for the custom Contacts list.

LISTING 3-10 The *Fields* section of the *MetaData* element for the sample list definition

```
<Fields>
  <Field ID="{c7792ad6-f2f3-4f2d-a7e5-75d5a8206fd9}"
         Name="DevLeapContactID"
         StaticName="DevLeapContactID"
         DisplayName="Contact ID"
         Type="Text"
         Group="DevLeap Columns"
         Sortable="TRUE" />
  <Field ID="{a8f2455O-55cd-4d34-aO15-811954c6ce24}"
         Name="DevLeapCompanyName"
         StaticName="DevLeapCompanyName"
         DisplayName="Company Name"
         Type="Text"
         Group="DevLeap Columns"
         Sortable="TRUE" />
  <Field ID="{149bf9a1-5bbb-468d-aa35-91aceb054e3b}"
         Name="DevLeapCountry"
         StaticName="DevLeapCountry"
         DisplayName="Country"
         Type="Choice"
         Group="DevLeap Columns"
         Sortable="TRUE">
  <Default>Italy</Default>
  <CHOICES>
    <CHOICE>Italy</CHOICE>
    <CHOICE>USA</CHOICE>
    <CHOICE>Germany</CHOICE>
    <CHOICE>France</CHOICE>
  </CHOICES>
  </Field>
  <Field ID="{ac689935-8e8b-485e-a45e-ff5a338dd92f}"
         Name="DevLeapCustomerLevel"
         StaticName="DevLeapCustomerLevel"
         DisplayName="Customer Level"
         Type="Choice"
         Group="DevLeap Columns">
    <Default>Level C</Default>
    <CHOICES>
      <CHOICE>Level A</CHOICE>
      <CHOICE>Level B</CHOICE>
      <CHOICE>Level C</CHOICE>
    </CHOICES>
  </Field>
  <Field ID="{a73de518-b9b9-4e8d-9d94-6099b4603997}"
         Name="DevLeapSupplierAccount"
         StaticName="DevLeapSupplierAccount"
         DisplayName="Supplier Account"
         Type="User"
         Group="DevLeap Columns"
         Sortable="TRUE" />
</Fields>
```

Just as with the *ContentTypes* section, the *Fields* section is simply a wrapper for the copies of all the previously defined site columns. Notice that the *ID* values for the site columns are the same as those of the global site columns, serving to keep the global site columns linked to the local list-level columns.

Figure 3-5 shows how the List Settings page of a list based on the custom Contacts list definition looks in a web browser. Note that all three content types and all the list-level columns are present.



Content Types

This list is configured to allow multiple content types. Use content types to specify the information you want to display about an item, in addition to its policies, workflows, or other behavior. The following content types are currently available in this list:

Content Type	Visible on New Button	Default Content Type
DevLeapContact	✓	✓
DevLeapCustomer	✓	
DevLeapSupplier	✓	

- Add from existing site content types
- Change new button order and default content type

Columns

A column stores information about each item in the list. Because this list allows multiple content types, some column settings, such as whether information is required or optional for a column, are now specified by the content type of the item. The following columns are currently available in this list:

Column (click to edit)	Type	Used in
Company Name	Single line of text	DevLeapContact, DevLeapCustomer, DevLeapSupplier
Contact ID	Single line of text	DevLeapContact, DevLeapCustomer, DevLeapSupplier
Country	Choice	DevLeapContact, DevLeapCustomer, DevLeapSupplier
Created	Date and Time	
Customer Level	Choice	DevLeapCustomer
Modified	Date and Time	
Supplier Account	Person or Group	DevLeapSupplier
Title	Single line of text	DevLeapContact, DevLeapCustomer, DevLeapSupplier
Created By	Person or Group	
Modified By	Person or Group	

FIGURE 3-5 The List Settings page of a list instance based on the custom Contacts list definition.

Just after the *Fields* section comes the *Views* element, which is a child of *MetaData*. This section is really interesting because it is where you define the views on data that will be available to the end users in the web browser. Each *View* element, which is a child of *Views*, defines a data view declaring some configuration attributes (illustrated in Table 3-5).

More Info For a complete list of all the available *View* attributes, refer to the official documentation on MSDN, at *http://msdn.microsoft.com/en-us/library/ms438338(v=office.15).aspx*.

TABLE 3-5 Some of the main attributes for the *View* element of a Schema.xml list definition file

Attribute	Description
Type	The type of view. *Type* can be *HTML*, *Chart*, or *Pivot*.
BaseViewID	An *Integer* value that declares the ID of the view. *BaseViewID* must be unique within a Schema.xml file.
Url	The public URL to access the view from the browser.
DisplayName	The name of the view in the web browser.
DefaultView	A *Boolean* value that declares if the view is the default view for the current list.
MobileView	A *Boolean* value that specifies if the current view has to be made available to mobile devices.
MobileDefaultView	A *Boolean* value that declares if the view, enabled for mobile access, is the default view for mobile devices.
SetupPath	Defines the site-relative path to the ASPX file corresponding to the current view model. It allows provisioning a custom page for the current view.
WebPartZoneID	A string that declares the ID of the WebPartZone control where the current view will be loaded, within the ASPX Web Part page.

The *View* element also allows you to declare some other configuration details using child elements. Listing 3-11 shows the default view definition for the list of contacts.

LISTING 3-11 The default *View* definition for the sample list

```
<View BaseViewID="1" Type="HTML"
      WebPartZoneID="Main"
      DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24;"
      DefaultView="TRUE" MobileView="TRUE"
      MobileDefaultView="TRUE"
      SetupPath="pages\viewpage.aspx"
      ImageUrl="/_layouts/images/generic.png"
      Url="AllItems.aspx">
  <Toolbar Type="Standard" />
  <RowLimit Paged="TRUE">50</RowLimit>
  <ViewFields>
    <FieldRef Name="Attachments">
    </FieldRef>
    <FieldRef Name="LinkTitle">
    </FieldRef>
  </ViewFields>
  <Query>
    <OrderBy>
      <FieldRef Name="ID">
      </FieldRef>
    </OrderBy>
  </Query>
  <XslLink>main.xsl</XslLink>
  <JSLink>clienttemplates.js</JSLink>
</View>
```

Listing 3-11 declares a *BaseViewID* with a value of *1*, and specifies that this view will be the default (*DefaultView*), not only for classic web browsers, but also for mobile devices (*MobileDefaultView*). The URL to access the view will be *AllItems.aspx*, and this page will be based on the *SetupPath* file pages\viewpage.aspx filling out the *WebPartZone* control whose *ID* is *Main*.

The child elements of the *View* tag in Listing 3-11 inform SharePoint to use the *Standard* value for the toolbar. The maximum number of rows (*RowLimit*) is set to return a value of *50*, enabling paging.

Note If not specified, the default *RowLimit* is *30*.

After these configuration elements, Listing 3-11 defines some other elements that determine the data to show, declaring a *Query* element to filter and sort data, and a set of *ViewFields* elements to show, as well as some optional grouping rules. The *Query* element is simply a Collaborative Application Markup Language (CAML) query that defines the values to extract from the source list, the ordering rule, and which values will be shown in the current view. For example, Listing 3-11 queries all the items in the list, sorting them by the value of their *ID* fields.

Note CAML is an XML-based querying language that can be used to define filtering, sorting, and grouping on SharePoint data. The CAML language reference is available on MSDN, at *http://msdn.microsoft.com/en-us/library/ms467521(v=office.15).aspx*. In case you are a SharePoint 2010 developer, consider that CAML hasn't changed that much between SharePoint 2010 and SharePoint 2013.

Another important child section of the *View* element is the *ViewFields* element, which declares the fields to show in the resulting view. These fields are referenced by their internal names, using a specific *FieldRef* element.

The last child elements in the *View* are the *XslLink* and *JsLink* elements. Since SharePoint 2010, SharePoint can render views using XSLT transformations. The *XslLink* element specifies the path to the XSLT file used to render the view. This XSLT file path is relative to the folder *SharePoint15_Root\TEMPLATE\LAYOUTS\XSL*. Moreover, starting from SharePoint 2013, the *JsLink* element allows declaring a JavaScript file to include and use for rendering the view.

Note SharePoint15_Root refers to the SharePoint root folder, which is typically located at C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15.

As an alternative to providing an explicit XSLT file path, you can use an *Xsl* element to simply declare the XSLT transformation inside the Schema.xml file. Because you may want to reuse the XSLT transformation, however, a better choice is to reference an external file. This is especially useful when you are developing a full-trust solution. The capability to define the XSLT transformation inside the

Schema.xml file is provided for those situations, such as for sandboxed solutions and SharePoint apps, when you want to avoid copying files to the file system of the target SharePoint farm.

The *Forms* element is another important configuration section for the list definition, as shown in Listing 3-12.

LISTING 3-12 The *Forms* configuration section of the custom Contacts list definition

```
<Forms>
  <Form Type="DisplayForm"
  Url="DispForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
  <Form Type="EditForm"
  Url="EditForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
  <Form Type="NewForm"
  Url="NewForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
</Forms>
```

The *Forms* element contains a set of *Form* elements that declare the forms available to the end user. Each *Form* element requires a *Type* attribute that takes one of the following values:

- ***DisplayForm*** The form to display a list item
- ***EditForm*** The form to edit an existing list item
- ***NewForm*** The form to add a new list item

Every form also requires a URL where it can be accessed. Forms might include an optional *SetupPath* attribute from which to load the ASPX page model, as well as a *WebPartZoneID* attribute, which specifies the ID of the Web Part zone used to load the rendering control of the form. As an alternative to the *SetupPath* attribute, you could have a *Path* attribute, which defines a physical file system path relative to the _layouts folder for a template file, and a *Template* attribute, which specifies the name of the template to use. You can also use CAML syntax to define the template for the body, buttons, opening section, and closing section of each of these forms, using these specific child nodes of the *Form* element: *ListFormBody*, *ListFormButtons*, *ListFormClosing*, and *ListFormOpening*.

The last configuration section shown is the *Validation* element. This element, introduced with SharePoint 2010, supports defining validation rules that can apply to each item of the list. Listing 3-13 shows how to declare a custom validation rule together with a validation error message that end users will see if validation fails.

LISTING 3-13 Declaring a sample validation rule for the custom Contacts list definition items

```
<Validation Message="Please check your data, there is something wrong!">
      =Title<>"Blank"
</Validation>
```

The validation rule forces the items to have a Title field with a value not equal to *Blank*. Notice that list-level validation rules work properly only with fields shared by all the content types of the list. If you enforce a rule against a field that is not defined in all the content types of the list, then your rule will always throw an error when applied to the wrong content types. For example, if you define a rule at the list level for the DevLeapCustomerLevel field of the *Customer* content type, you will not be able to add or update any *Supplier* instances, because the *DevLeapCustomer* field is not present in the *Supplier* content type. In such cases, you should instead define the validation rule at the site column level.

Defining a custom view

When defining custom list definitions, you'll frequently want to declare some custom views that correspond to the business rules of your data model. For example, the sample model could feature one view that shows only customers and another that shows only suppliers. This section demonstrates how to define the former view; the latter's definition will be almost identical.

First, define a new *View* element under the *Views* element of the Schema.xml file. The new view will have a unique *BaseViewID*; in this example it will be *2*. The *DisplayName* will be All Customers, the *Type* will be HTML, and the *Url* will be *AllCustomers.aspx*. All the other attributes values of the *View* element are trivial. You can see the complete definition of this view in Listing 3-14.

LISTING 3-14 Defining a custom view for a custom Contacts list definition

```
<View BaseViewID="2" Type="HTML"
      WebPartZoneID="Main"
      DisplayName="All Customers"
      DefaultView="FALSE" MobileView="TRUE"
      MobileDefaultView="FALSE"
      SetupPath="pages\viewpage.aspx"
      ImageUrl="/_layouts/images/generic.png"
      Url="AllCustomers.aspx">
  <Toolbar Type="FreeForm" />
  <XslLink>Contacts_Main.xsl</XslLink>
  <RowLimit Paged="TRUE">20</RowLimit>
  <ViewFields>
    <FieldRef Name="Attachments">
    </FieldRef>
    <FieldRef Name="LinkTitle">
    </FieldRef>
    <FieldRef Name="DevLeapContactID">
    </FieldRef>
    <FieldRef Name="DevLeapCompanyName">
    </FieldRef>
    <FieldRef Name="DevLeapCountry">
    </FieldRef>
    <FieldRef Name="DevLeapCustomerLevel">
    </FieldRef>
  </ViewFields>
```

```
  <Query>
    <Where>
      <Eq>
        <FieldRef Name="ContentType" />
        <Value Type="Text">DevLeapCustomer</Value>
      </Eq>
    </Where>
    <OrderBy>
      <FieldRef Name="ID">
      </FieldRef>
    </OrderBy>
  </Query>
</View>
```

There are some areas of interest in this view definition. First, the code defines a *Query* to filter only items with a *ContentType* value of *DevLeapCustomer* and orders the result by the item *ID*. Then it references all the fields of the *Customer* content type, defining a set of *FieldRef* elements within the *ViewFields* element. Lastly, a custom XSLT transformation is defined for rendering the custom view. SharePoint will search for this XSLT file, Contacts_Main.xsl, in the SharePoint15_Root\TEMPLATE\LAYOUTS\XSL *folder*. The file has to be placed in that folder using the solution-provisioning tools provided by Visual Studio 2012 to create a full-trust solution. (For further details, see Chapter 4, "SharePoint features and solutions.") Otherwise, as you have already seen, you can define the XSLT code directly in the *View* schema definition, inside an *Xsl* element.

The XSLT file you reference or define in the *View* definition is a common XSLT transformation that will receive a wide range of parameters at run time from SharePoint. In the XSLT code, for example, you can access the *XmlDefinition* variable, which provides the XML definition of the current *View*. To define an XSLT for a custom view, you must provide an XSLT template that matches the *BaseViewID* of the targeted view. For the Contacts example, the following template was defined:

```
<xsl:template match="View[@BaseViewID="2"]" mode="full">
  <!-- Here is our custom XSLT transformation -->
</xsl:template>
```

The XSLT also receives a parameter named *Rows* that contains all the items to be rendered. Listing 3-15 shows an excerpt of the XML content of the *Rows* parameter. You can read it simply by using an XSLT template that copies the source content with an *<xsl:copy-of />* element.

LISTING 3-15 The content of the *Rows* parameter provided to a custom XSLT for rendering a list view

```
<Rows>
  <Row ID="1" PermMask="0x7fffffffffffffff" Attachments="0"
  Title="Customer 01" FileLeafRef="1_.000" FileLeafRef.Name="1_"
  FileLeafRef.Suffix="000" FSObjType="0"
  Created_x0020_Date="1;#2010-02-13 16:24:12" Created_x0020_Date.ifnew="1"
  FileRef="/sites/SP2010DevRef/Lists/Test/1_.000"
  FileRef.urlencode="%2Fsites%2FSP2010DevRef%2FLists%2FTest%2F1%5F%2E000"
  FileRef.urlencodeasurl="/sites/SP2010DevRef/Lists/Test/1_.000"
  File_x0020_Type=""
  HTML_x0020_File_x0020_Type.File_x0020_Type.mapall="icgen.gif||"
  HTML_x0020_File_x0020_Type.File_x0020_Type.mapcon=""
  HTML_x0020_File_x0020_Type.File_x0020_Type.mapico="icgen.gif" ContentTypeId
="0x0100A60F69C4B1304FBDA6C4B4A25939979F010044C1B948A829E64CBD49ED3F42A868C7"
DevLeapContactID="C01"DevLeapCompanyName="Company 01"
  DevLeapCountry="Italy" DevLeapCustomerLevel="Level C"
  ContentType="DevLeapCustomer"></Row>
  <!—And many other rows here, one for each list item to show -->
</Rows>
```

Listing 3-15 illustrates that the *Rows* parameter provides each row along with its data columns, specified as attributes of a *Row* element. To output the content of the rows, you simply need to retrieve the values of these attributes, placing them inside the proper HTML elements to adhere to the graphical layout that you need to render.

However, many SharePoint developers do not like writing XSLT files, because XSLT is inflexible (although very powerful) from a syntax viewpoint. Luckily, starting with SharePoint 2013, you have the option to provide a custom JavaScript file through the *JsLink* child element of the *View* element, in order to move rendering templates into client-side code. Generally speaking, this technique is known as client-side rendering (CSR). Listing 3-16 uses this new technique to define a custom view.

LISTING 3-16 A custom view definition for the custom Contacts list definition using JavaScript rendering

```
<View BaseViewID="3" Type="HTML"
      WebPartZoneID="Main"
      DisplayName="All Customers via JS"
      DefaultView="FALSE" MobileView="TRUE"
      MobileDefaultView="FALSE"
      SetupPath="pages\viewpage.aspx"
      ImageUrl="/_layouts/images/generic.png"
      Url="AllCustomersViaJS.aspx">
  <Toolbar Type="FreeForm" />
  <XslLink>main.xsl</XslLink>
  <JsLink Default="TRUE">~site/Scripts/CustomCustomersView.js</JsLink>
  <RowLimit Paged="TRUE">20</RowLimit>
```

```
  <ViewFields>
    <FieldRef Name="Attachments">
    </FieldRef>
    <FieldRef Name="LinkTitle">
    </FieldRef>
    <FieldRef Name="DevLeapContactID">
    </FieldRef>
    <FieldRef Name="DevLeapCompanyName">
    </FieldRef>
    <FieldRef Name="DevLeapCountry">
    </FieldRef>
    <FieldRef Name="DevLeapCustomerLevel">
    </FieldRef>
  </ViewFields>
  <Query>
    <Where>
      <Eq>
        <FieldRef Name="ContentType" />
        <Value Type="Text">DevLeapCustomer</Value>
      </Eq>
    </Where>
    <OrderBy>
      <FieldRef Name="ID">
      </FieldRef>
    </OrderBy>
  </Query>
</View>
```

In Listing 3-16, shows the *JsLink* element (highlighted in bold) configured as the default (*Default*="*TRUE*") rendering template. SharePoint will look for the JavaScript file at a URL relative to the current site collection, because of the *~site* token at the very beginning of the URL. You can deploy the JavaScript code of the CustomCustomerView.js file to the target site simply working at the website level, using a sandboxed solution or an app deployment process. In the JavaScript code, you can reference the Client Object Model of SharePoint in order to query the current list configuration, as well as the items to render. This technique is extremely powerful. While provisioning lists for Office 365, for example, you can use this technique to move all the rendering logic to the client side, using jQuery or CSS rendering templates. With its XSLT and JavaScript support, SharePoint opens up some great business opportunities; because it gives you the capability to display fully customized rendering of list views, your solutions can support fully customized template layouts, even in extreme web content management solutions.

More Info For more information about CSR, you can read the document "How to: Customize a list view in apps for SharePoint using client-side rendering," available at *http://msdn.microsoft.com/en-us/library/jj220045.aspx*.

The *ListTemplate* definition file

ListTemplate is the feature element file that declares all the deployment properties needed to provision the list definition. It must be provisioned into a custom feature together with the Schema.xml file. Listing 3-17 shows the *ListTemplate* for the sample Contacts list definition.

LISTING 3-17 The *ListTemplate* feature element for the sample Contacts list definition

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <ListTemplate
        Name="DevLeapContacts"
        Type="10001"
        BaseType="0"
        OnQuickLaunch="TRUE"
        SecurityBits="11"
        Sequence="410"
        DisplayName="DevLeap Contacts"
        Description="A list of Contact for DevLeap"
        Image="/_layouts/images/dlcon.png"/>
</Elements>
```

The *Type* attribute is the most important attribute in the *ListTemplate* element. *Type* takes an integer value that should be unique at the site collection level. The code sample uses a value of *10001* to avoid overlapping with values of out-of-the-box list templates. In general, you should use a large integer value to avoid overlapping with SharePoint. Consider that values in the range between 100 and 1200 are already taken, and developers should allocate numbers greater than 10000. The uniqueness of this attribute allows you to define custom UI extensions that will target the entire set of lists with that *Type* value.

The other attributes are straightforward. The *BaseType* attribute states the base type for the current list definition. The *Name* attribute represents the internal name of the list, and the *DisplayName* is the text shown to end users, together with the *Description* and the *Image*. You can load the values of these descriptive attributes from external resource strings to provision list definitions in a multilanguage environment. The *OnQuickLaunch Boolean* attribute value controls whether SharePoint shows any instance of the list in the Quick Launch menu. You can also provision a list instance through a custom feature of type *ListInstance*, which will be explained in Chapter 4.

Finally, the *SecurityBits* attribute defines the security behavior of the list. This is a two-digit string, where the first digit controls whether users can read all items (*1*) or only their own items (*2*). The second digit defines edit access permissions. The possible values are

- **1** Users can edit any item.
- **2** Users can edit only their own items.
- **4** Users cannot edit items.

For example, a value of *22* for the *SecurityBits* attribute means that users can see and edit only their own items, while the default value of *11* means that users can see and edit all the items in the list.

More Info For a complete list of attributes for the *ListTemplate* element, refer to the official product documentation on MSDN, at *http://msdn.microsoft.com/en-us/library/ms462947(v=office.15).aspx*.

Working with lists in Visual Studio 2012

Just as you can define content types with Visual Studio 2012 and its designers, you can also define basic lists. In fact, whenever you add an item of type List to a SharePoint project, regardless of whether it is a solution or an app, you are provided with a graphical designer that allows you to design fields, content types, and views, and provide descriptive information for the list. First, you are prompted with the wizard shown in Figure 3-6. Here you can specify the name of the target list and create a customizable list definition based on a basic content type or a list instance based on an existing list definition.

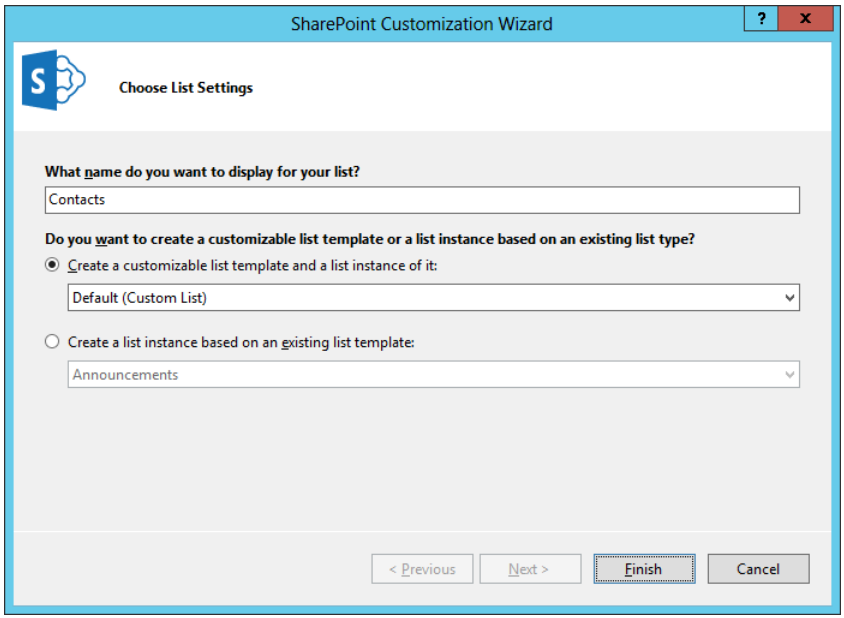


FIGURE 3-6 The wizard for creating a new list in a SharePoint solution or app.

After you complete the page and click Finish, you can configure the resulting item through a specific designer. If you created a new list definition, you will have access to a designer with three

tabs, for configuring fields, content types, and views of the custom list definition. Figure 3-7 shows the designer for this chapter's example Contacts list, displaying the columns defined in the schema of the list definition.

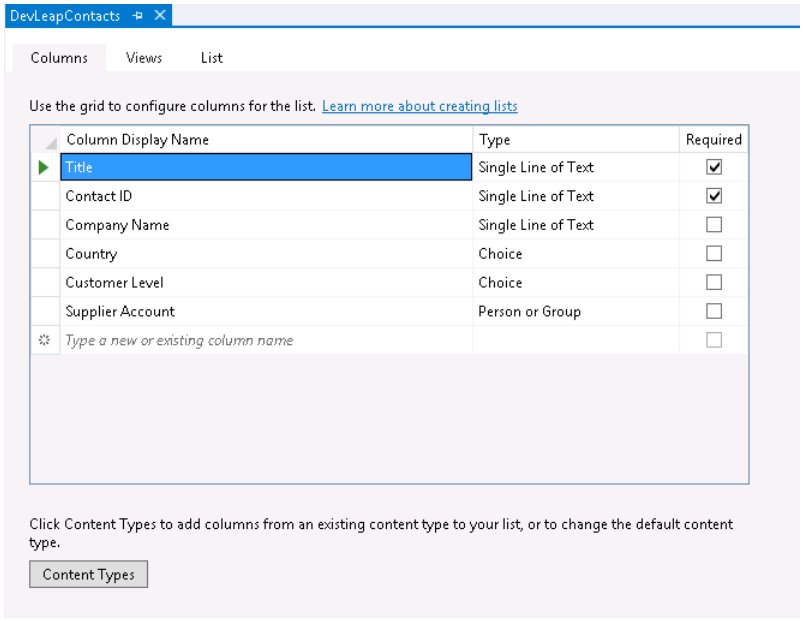


FIGURE 3-7 Configuring the fields of a custom list definition within Visual Studio 2012.

The designer also provides also a Content Types button; click it to open the dialog box shown in Figure 3-8. Here you can determine the content types associated with the current list template.

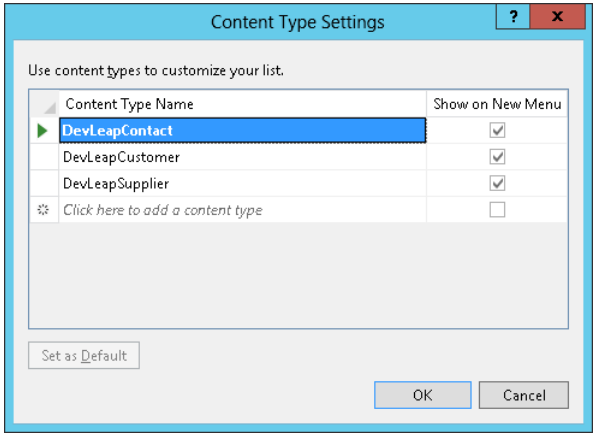


FIGURE 3-8 The dialog box for configuring the content types associated with a list definition.

Once you have defined the content types and the columns, you can determine the views for the custom list definition. Click the Views tab to access the controls shown in Figure 3-9.

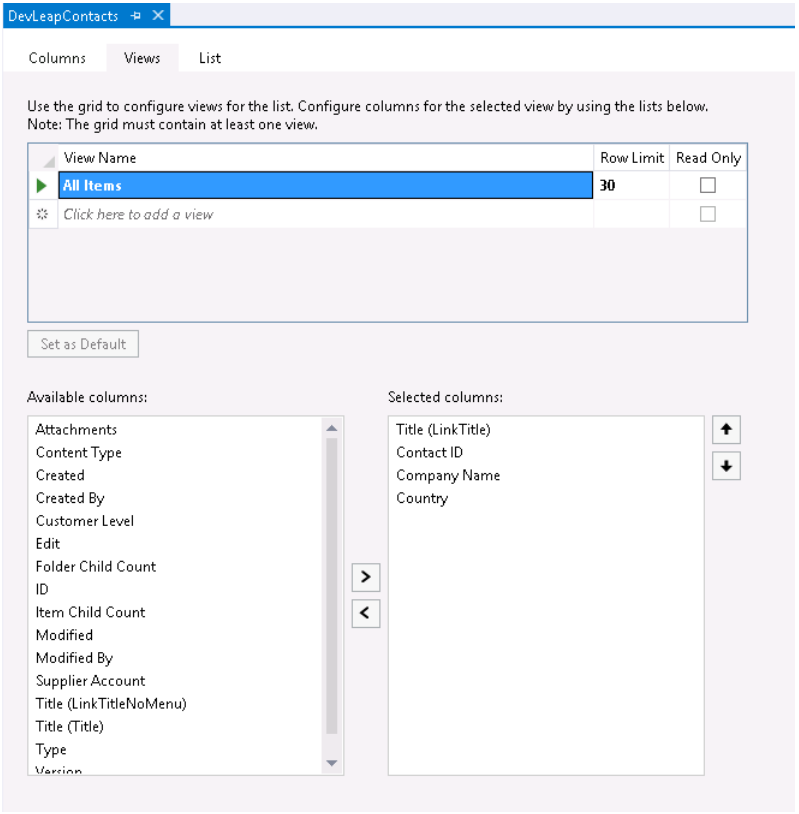


FIGURE 3-9 Determining the views for the custom list definition.

Whether you are defining an instance of your custom list definition or simply declaring an instance of an already existing list definition, you can configure some descriptive aspects of the target list using the List tab, shown in Figure 3-10.

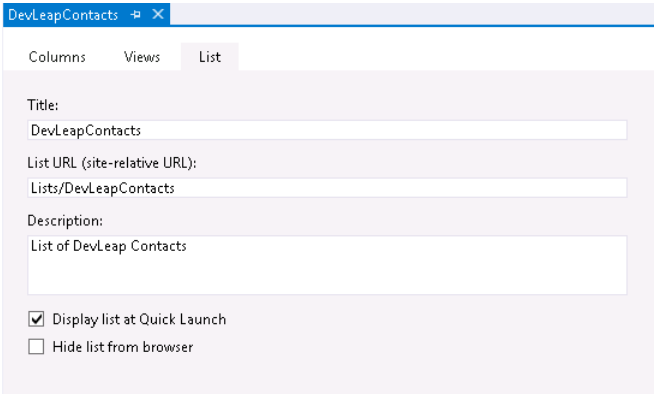


FIGURE 3-10 The List tab for configuring the list instance descriptive parameters.

By default, Visual Studio 2012 always defines a list instance together with the list definition. If you do not want to provision a list instance, you can comment the code of the *ListInstance* element created within the Elements.xml file available inside the list item in the Visual Studio project outline.

Summary

This chapter described how to define XML files to provision SharePoint data models and structures. In particular, it showed how to use feature element files to deploy site columns, content types, and list definitions. It also discussed how to do similar things using the designer provided by Microsoft Visual Studio 2012, instead of using low-level XML files. These features promise a great return on investment and a common maintenance plan.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

About the Author

PAOLO PIALORSI is a consultant, trainer, and author who specializes in developing distributed application architectures and Microsoft SharePoint–based enterprise solutions. During his professional career, he has passed more than 40 Microsoft certification exams. Paolo has a great deal of experience working with SharePoint, and he is a Microsoft Certified Master (MCM) for SharePoint 2010. He is one of the content owners of the Italian version of the SharePoint & Office Conference, and he is a popular speaker at worldwide industry conferences.

He is the author of many Microsoft Press books on Microsoft .NET, Microsoft Windows 8, and SharePoint. Recent books include *Programming Microsoft LINQ in Microsoft .NET Framework 4*, *Build Windows 8 Apps with Microsoft Visual C# and Visual Basic Step by Step*, *Build Windows 8 Apps with Microsoft Visual C++ Step by Step*, and *Microsoft SharePoint 2010 Developer Reference*. He has also written three Italian-language books, on the topics of .NET, XML, and web services.

You can reach Paolo via the following:

- **The SharePoint Developer Reference blog** *http://www.sharepoint-reference.com*
- **Twitter** *@PaoloPia*; *http://www.twitter.com/PaoloPia*
- **LinkedIn** *http://it.linkedin.com/in/paolopialorsi/*

What do you think of this book?

We want to hear from you!

To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

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

Thank you in advance for your input!

Microsoft® Press