Microsoft

Inside Microsoft SharePoint 2013

Professional

Scot Hillier, Mirjam van Olst,
Ted Pattison, Andrew Connell,
Wictor Wilén, Kyle Davis

Inside Microsoft SharePoint 2013

Build custom SharePoint solutions with architectural insights from the experts

Take a deep dive into SharePoint 2013, and master the intricacies for designing and implementing robust apps and other business solutions for your organization. Led by an author team with in-depth knowledge of SharePoint architecture, you'll thoroughly explore the SharePoint 2013 development platform and new app model through hands-on tasks and extensive code samples.

Discover how to:

- Create SharePoint-hosted, provider-hosted, and autohosted apps
- Master the new app security model with OAuth and Certificates
- Develop workflows with the SharePoint 2013 workflow model
- Design a custom search experience and create search-based apps
- Leverage the client-side object model and REST APIs
- Produce catalog-driven web sites with Web Content Management capabilities
- Get cloud-based data sources with Business Connectivity Services
- Create and utilize remote event receivers for lists and libraries
- Generate new social networking apps and solutions

Download code samples at:

http://aka.ms/InsideSP2013/files

About the Authors

Scot Hillier is an MVP for SharePoint focused on creating business productivity solutions.

Mirjam van Olst is an MVP for SharePoint who helps companies implement SharePoint solutions.

Ted Pattison, MVP for SharePoint, cofounded a company for educating clients on how to become successful with SharePoint.

Andrew Connell, an MVP for SharePoint Server, is an independent consultant and instructor.

Wictor Wilén is a Microsoft Certified Master in SharePoint and works as a SharePoint Solution Architect at a consulting company.

Kyle Davis, MCITP, MCPD, is a SharePoint and cloud solutions architect and managing consultant.

microsoft.com/mspress

ISBN: 978-0-7356-7447-9

9 780735 674479 90000

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

Microsoft Office/Microsoft SharePoint

Microsoft Press

Celebrating 30 years!

Microsoft

Inside Microsoft SharePoint 2013

Scot Hillier
Mirjam van Olst
Ted Pattison
Andrew Connell
Wictor Wilén
Kyle Davis

ISBN: 978-0-7356-7447-9

2 3 4 5 6 7 8 9 10 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: Kara Ebrahim

Editorial Production: Online Training Solutions, Inc. (OTSI)

Technical Reviewers: Wayne Ewington and Neil Hodgkinson

Copyeditor: Online Training Solutions, Inc. (OTSI)

Indexer: Angela Howard

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

[2013-11-21]

Contents at a glance

Contents

Introduction

The purpose of this book is to help you design and develop custom business apps and solutions for SharePoint 2013, which includes the two products SharePoint Foundation and SharePoint Server 2013. Our goal is to teach you how to create, debug, and deploy the various components of apps and solutions such as Features, Pages, App Parts, Remote Event Handlers, and Workflows. Once you apply yourself and become comfortable developing with these building blocks, there's no limit to the types of apps and solutions you can create on the SharePoint 2013 platform.

Who this book is for

This book is written for experienced SharePoint developers who are proficient with Microsoft Visual Studio 2012, the Microsoft .NET Framework 4, and who understand the fundamentals of the SharePoint object model. The code samples in this book are written in JavaScript and C# and are intended to represent the spectrum of possible solutions. The primary audience for the book is SharePoint architects and developers looking to master SharePoint 2013 development.

Organization of this book

This book is organized into 17 chapters:

- Chapter 1, "SharePoint 2013 developer roadmap," provides a strategic view of SharePoint development options. The chapter presents the various development models and how they fit into the overall SharePoint development story.
- Chapter 2, "SharePoint development practices and techniques," provides guidance in setting up your development environment. Additionally, the chapter covers related technologies that are important for SharePoint development, such as Windows PowerShell.
- Chapter 3, "Server-side solution development," presents the fundamentals of sandbox and full-trust solution development. The chapter also presents the basics of the server-side object model.
- Chapter 4, "SharePoint apps," covers the new app model in detail. This chapter presents the tools and techniques necessary for developing apps.

- Chapter 5, "Client-side programming," first provides a JavaScript and jQuery primer for SharePoint developers with an emphasis on professional patterns. The second half of the chapter presents the fundamentals of the client-side object model and REST APIs for SharePoint 2013.
- Chapter 6, "SharePoint security," presents the security concepts necessary for successfully developing solutions and apps. This chapter explains the concepts behind user authentication and authorization, in addition to the app principal. This chapter also presents the details behind the claims and OAuth security models.
- Chapter 7, "SharePoint pages," presents techniques and information for working with pages in SharePoint solutions and apps. The chapter covers core concepts such as master pages, content placeholders, and application pages.
- Chapter 8, "SharePoint Web Parts," presents the tools and techniques required to create Web Parts and app parts.
- Chapter 9, "SharePoint lists," presents the information necessary for creating lists and performing operations against them. This chapter contains many code samples for reading and writing, using both server and client technologies.
- Chapter 10, "SharePoint type definitions and templates," covers the techniques for creating field types and field controls. The second part of the chapter covers the new JSLink technology and how it can be used to customize views.
- Chapter 11, "SharePoint site provisioning," shows how to create site templates and site definitions. These templates can be reused in solutions and apps.
- Chapter 12, "SharePoint workflows," presents all the information necessary for developing custom workflows by using the new Workflow Manager engine. Techniques for both the SharePoint Designer and Visual Studio are presented.
- Chapter 13, "SharePoint search," presents architecture and development techniques for Enterprise Search. The chapter details the creation of no-code solutions as well as apps.
- Chapter 14, "SharePoint Enterprise Content Management," presents structure and development techniques for managed metadata, document services, and records management.

- Chapter 15, "Web content management," details the significant improvements made for supporting website development. The chapter presents improvements in data-driven sites, master page creation, and metadata navigation.
- Chapter 16, "Business Connectivity Services," provides the background and tools for creating solutions based on data in external systems. The chapter presents approaches for both solutions and apps.
- Chapter 17, "SharePoint social enterprise features," presents the details of the new social infrastructure. The chapter also shows how to create solutions that utilize social features.

Acknowledgments

The process of writing this book really began two years before the release of SharePoint 2013, when we were fortunate enough to be selected as the team to create the first SharePoint 2013 training materials for Microsoft. We worked through many "Dev Kitchens" with the SharePoint team and got great information from Mike Ammerlann, Rob Howard, Brad Stevenson, Mike Morton, Mauricio Ordonez, and many others. After learning the technologies, we worked with a great team headed by Keenan Newton to deliver training to Microsoft personnel around the country. Later, we worked with Uma Subramanian and the MSDN team to create samples and videos to be deployed online. Thanks to all these people and everyone at Microsoft for the wonderful support and opportunities.

Of course, the book could not possibly have come together without the patience and support of the team at Microsoft Press, starting with our editor, Ken Brown (O'Reilly Media). Although we frustrated him endlessly at times, he maintained focus and drove us all to success. We'd also like to thank Kara Ebrahim (Production Editor, O'Reilly Media), Kathy Krause (Copyeditor, Online Training Solutions, Inc. [OTSI]), Wayne Ewington (Technical Editor), and Neil Hodgkinson (Technical Editor).

Thanks, everyone. It feels great to be done!

Support & feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata

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 at:

http://aka.ms/InsideSP2013/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://aka.ms/tellpress

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

Stay in touch

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

CHAPTER 1

SharePoint 2013 developer roadmap

Microsoft SharePoint technologies have become increasing popular and have made it into the mainstream of IT infrastructures used by companies and organizations around the world. Today, millions of people work with SharePoint technologies every day, including business users, power users, executives, site administrators, farm administrators, and professional developers.

It is important for you, as a software developer, to view SharePoint technologies as a true platform for professional developers. The key point is that SharePoint technologies serve as a foundation on top of which you can design and implement business solutions. However, getting started can be daunting because there are several different versions of the SharePoint platform, and each version has several different variations.

Over the last decade, most of the companies that have used SharePoint technologies have deployed them as server-side software products on server computers that are under their control. This is a scenario that is often referred to as *SharePoint on-premises*. It is also important to note that the vast majority of SharePoint-related development projects have historically targeted the SharePoint on-premises scenario. However, this is beginning to change, and the change is occurring at a very fast pace.

Over the last few years, Microsoft has shifted the focus of their SharePoint adoption strategy from the original on-premises model to a newer subscription-based model where the SharePoint platform is made available to customers as a cloud-based service hosted in the Microsoft Office 365 environment. The hosted version of the SharePoint platform in the Office 365 environment is known as *SharePoint Online*.

It's clear that Microsoft sees SharePoint Online as the future direction of SharePoint technologies. However, it's also true that a significantly large portion of the existing SharePoint customer base is still using the older SharePoint on-premises model. Microsoft's ongoing effort to move its SharePoint customer base from the original on-premises model to SharePoint Online raises a few important questions:

- Are the SharePoint on-premises model and SharePoint Online just two different variations of the same development platform, or do they represent two entirely different platforms?
- When developing a business solution for SharePoint 2013, is it important to choose between targeting SharePoint on-premises and targeting SharePoint Online?

- Can you write a generic business solution that runs equally well in a SharePoint on-premises environment and in the SharePoint Online environment?

Unfortunately, the answer to each of these questions is "it depends," because they are all dependent upon the scenario at hand. In one scenario, you might be able to write a generic business solution that works in on-premises environments and in SharePoint Online. In another scenario, you might find it necessary to use a development technique that works in on-premises environments but doesn't work at all in SharePoint Online. In a third scenario, you might decide to take advantage of features in the Office 365 environment that are not available in the on-premises environment.

The bottom line is that there are an incredible number of details and techniques that you have to learn if you want to build a level of expertise across the entire SharePoint platform. The goal of this book is to cover the SharePoint 2013 development story from end to end to prepare you to make the correct choices in any SharePoint development scenario you might encounter.

A brief history of SharePoint

Microsoft has released five versions of SharePoint technologies, which are listed in Table 1-1. Each SharePoint release has included an underlying core infrastructure product and a second product that adds business value to the infrastructure. The core infrastructure product has always been free to customers who already have licenses for the underlying server-side operating system, Windows Server. Microsoft makes money on SharePoint technologies in the on-premises model by selling customers server-side licenses as well as client access licenses (CALs).

TABLE 1-1 A brief history of SharePoint

Year	Core infrastructure product	Business value product
2001	SharePoint Team Services	SharePoint Portal Server 2001
2003	Windows SharePoint Services 2.0	Microsoft SharePoint Portal Server 2003
2007	Windows SharePoint Services 3.0	Microsoft Office SharePoint Server 2007
2010	Microsoft SharePoint Foundation 2010	Microsoft SharePoint Server 2010
2013	Microsoft SharePoint Foundation 2013	Microsoft SharePoint Server 2013

SharePoint 2001 introduced an environment that allowed users to create sites, lists, and document libraries on demand based on a data-driven design. The implementation was based on a Microsoft SQL Server database that tracked the creation of sites and lists by adding records to a static set of database tables. This initial version of SharePoint had a couple of noteworthy shortcomings. First, it was cumbersome to customize sites. Second, the files uploaded to a document library were stored on the local file system of a single, dedicated web server, which made it impossible to scale out SharePoint Team Services sites by using a farm of web servers.

SharePoint 2003 was the first version to be implemented on top of the Microsoft .NET Framework and ASP.NET. This version began to open up new opportunities for professional developers looking to

extend the SharePoint environment with Web Parts and event handlers. Also in this version, Microsoft altered the implementation for document libraries to store files inside a back-end SQL Server database, which made it possible to scale out SharePoint sites by using a farm of web servers.

SharePoint 2007 introduced many new concepts to the underlying SharePoint architecture, including site columns, content types, and features and solution packages. Microsoft also improved the integration of SharePoint with ASP.NET, which made it possible for .NET developers to extend SharePoint sites by creating familiar ASP.NET components such as master pages, user controls, navigation providers, authentication providers, and custom *HttpModule* components.

SharePoint 2010 was the fourth release of SharePoint technologies. It included Microsoft SharePoint Foundation 2010 and Microsoft SharePoint Server 2010. SharePoint 2010 introduced the new service application architecture and a significant modernization to the user interface experience with the server-side ribbon, model dialogs, and new Asynchronous JavaScript and XML (AJAX) behavior that reduced the need for page post backs. It was also with the SharePoint 2010 release that the Microsoft Visual Studio team released the original version of the SharePoint Developer Tools, which moved SharePoint developers out of the dark ages and into a far more productive era.

SharePoint 2013 is the fifth and most recent release of SharePoint technologies. It includes SharePoint Foundation 2013 and Microsoft SharePoint Server 2013. As you will see, the most significant changes that Microsoft has made to SharePoint 2013 have been done to adapt the SharePoint platform for hosted environments in the cloud, such as SharePoint Online in the Office 365 environment. This is a big change for developers because the SharePoint platform has been split in two. There is the older, familiar SharePoint platform in scenarios in which a company has deployed SharePoint on-premises. And now there is a second SharePoint platform in which developers are called upon to provide business solutions for hosted environments such as SharePoint Online.

Understanding the impact of SharePoint Online on the SharePoint platform

With its first two releases, Microsoft generated revenue from SharePoint technologies by using only the on-premises model. More specifically, Microsoft made money by selling SharePoint Server as a traditional software product that requires the customer to purchase a server-side license for each server and a client access license (CAL) for each user.

Starting in the SharePoint 2007 life cycle, Microsoft began to sell hosted versions of SharePoint that were bundled together with other services such as Microsoft Exchange, under the name of Business Productivity Online Standard Suite (BPOS). In the SharePoint 2010 life cycle, Microsoft changed the name of their bundled hosting service from BPOS to Office 365.

SharePoint 2013 represents the version in which Microsoft got serious about adapting the SharePoint platform for hosted environments such as SharePoint Online. This is evidenced by significant investments on the part of Microsoft to re-architect many core aspects of the SharePoint platform that had been causing scalability issues in SharePoint Online with SharePoint 2010.

Microsoft made one other big decision that is having a profound impact on every developer that works with the SharePoint platform. With SharePoint 2013, Microsoft has introduced a new strategy for developing business solutions based on the new *SharePoint app model*, which is a 180-degree turn from anything that has existed before.

With SharePoint 2013, there are now two separate and distinct styles in which you can develop a business solution. First, there is the original style of SharePoint development based on SharePoint solutions. Second, there is the new style of development based on SharePoint apps. This means that you must decide between creating a SharePoint solution and creating a SharePoint app each time you start a new development project on the SharePoint platform. So which one should you choose? The answer to that question is easy: it depends.

Examining SharePoint Foundation architecture

At its core, SharePoint Foundation 2013 is a provisioning engine—that is, its fundamental design is based on the idea of using web-based templates to create sites, lists, and libraries to store and organize content. Templates are used to create both new websites and various elements inside a website, such as lists, pages, and Web Parts.

SharePoint Foundation is particularly valuable to companies and organizations faced with the task of creating and administering a large number of websites, because it dramatically reduces the amount of work required. Someone in the IT department or even an ordinary business user can *provision* (a fancy word for *create*) a site in SharePoint Foundation in less than a minute by filling in a browser-based form and clicking the OK button. Creating a new page or a new list inside a site is just as easy.

SharePoint Foundation takes care of all the provisioning details behind the scenes by adding and modifying records in a SQL Server database. The database administrator doesn't need to create a new database or any new tables. The ASP.NET developer doesn't need to create a new ASP.NET website to supply a user interface. And the system administrator doesn't need to copy any files on the front-end web server or configure any Internet Information Services (IIS) settings. It all just works. That's the magic of the SharePoint platform.

The architecture of SharePoint Foundation was specifically designed to operate in a web farm environment. Figure 1-1 shows a basic diagram of a simple web farm with two front-end web servers and a database server. In scenarios that have multiple web servers, a network load balancer is used to take incoming HTTP requests and determine which front-end web server each request should be sent to.

SharePoint Foundation 2013 and SharePoint Server 2013 are available only in 64-bit versions. They can be installed on a 64-bit version of Windows Server 2012 or Windows Server 2008 R2. Unlike SharePoint 2010, Microsoft does not support installing SharePoint 2013 on a client operating system such as Windows 7 or Windows 8. However, you can run Windows 8 and enable Microsoft Hyper-V, which will allow you to create virtual machines (VMs) based on Windows Server 2012 or Windows Server 2008 R2. Therefore, you can install SharePoint 2013 on a VM running on Windows 8.

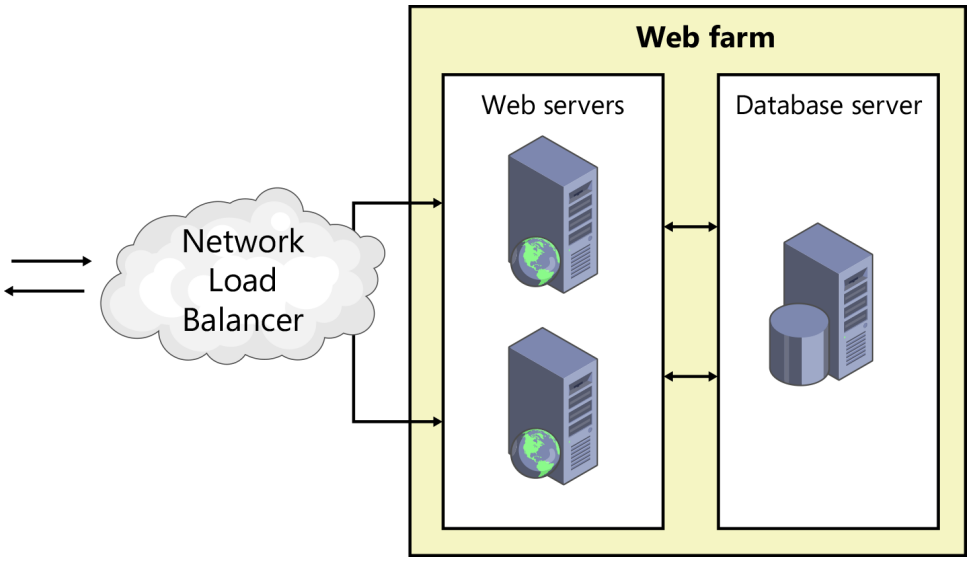


FIGURE 1-1 SharePoint Foundation is designed to scale out by using a farm of web servers.

SharePoint Foundation takes advantage of IIS on front-end web servers to listen for incoming HTTP requests and to manage the server-side worker processes by using the IIS application pool infrastructure. The version of IIS depends upon the operating system. Windows Server 2012 will use IIS 8.0, whereas Windows Server 2008 R2 will use IIS 7.5. The runtime environment of SharePoint Foundation runs within a worker process launched from the IIS application pool executable named w3wp.exe. As shown in Figure 1-2, SharePoint Foundation 2013 is built on .NET Framework 4.5.

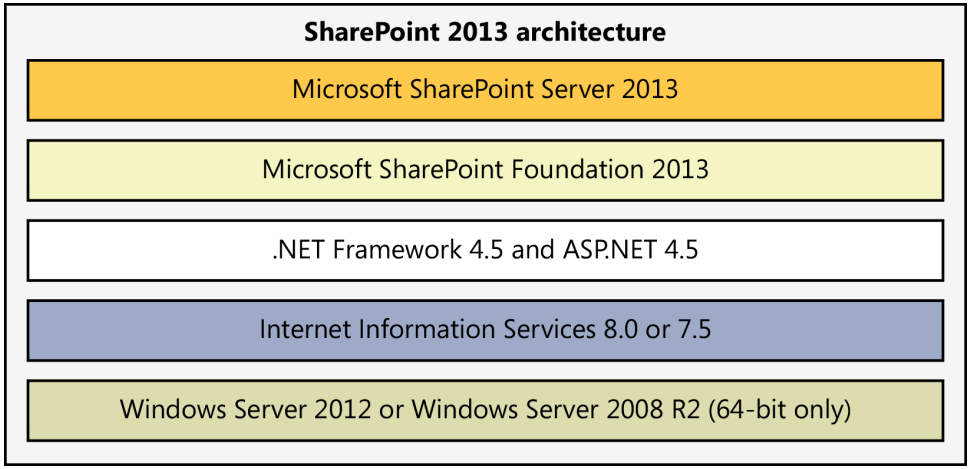


FIGURE 1-2 The SharePoint Foundation runtime loads into an IIS application pool running ASP.NET 4.5.

Understanding SharePoint farms

Every deployment of SharePoint Foundation is based on the concept of a farm. Simply stated, a *SharePoint farm* is a set of one or more server computers working together to provide SharePoint Foundation functionality to clients. For simple scenarios, you can set up an on-premises farm by installing SharePoint 2013 and configuring everything you need on a single server computer or a single VM. An on-premises farm in a typical production environment runs SQL Server on a separate, dedicated database server and can have multiple front-end web servers, as shown in Figure 1-3. As you will learn later in this chapter, a farm can also run one or more application servers in addition to a database server and a set of web servers.

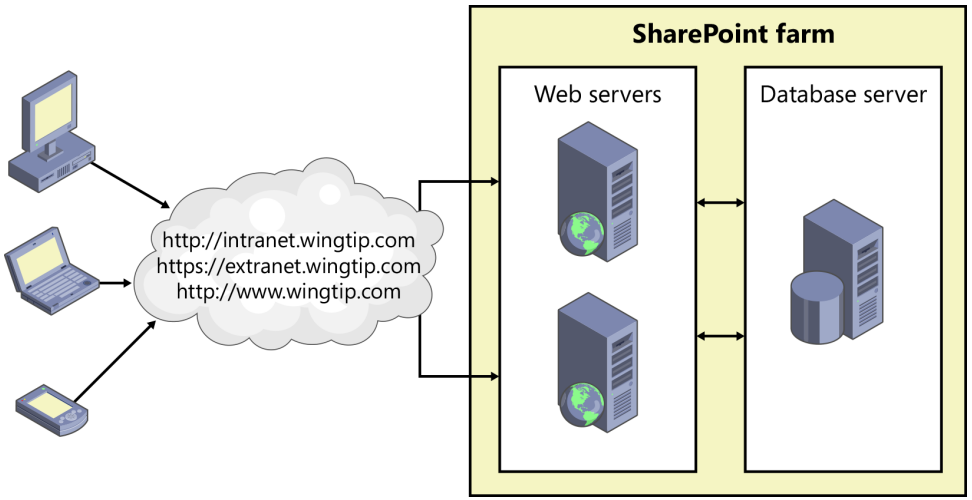


FIGURE 1-3 A SharePoint farm is a set of servers running SharePoint that are all associated by a single configuration database.

Each SharePoint farm runs a single SQL Server database known as the *configuration database*. SharePoint Foundation creates a configuration database whenever it creates a new farm, in order to track important farm-wide information. For example, the configuration database tracks which web servers are associated with the farm, as well as which users have been assigned administrative permissions within SharePoint Foundation at the farm level.

When you are creating a SharePoint 2013 development environment with an on-premises farm, it is typical to install and configure SharePoint 2013 as a single-server farm by using either Windows Server 2012 or Windows Server 2008 R2. You have the option of installing a version of SharePoint 2013 on a native installation of Windows Server or on a virtual machine (VM). For example, you can install a 64-bit version of Windows 8 as a host operating system and configure it to run Hyper-V. Hyper-V allows you to create a VM on which you can install a 64-bit version of Windows Server 2012 and SharePoint Server 2013.

As a SharePoint developer, you must remember that farms come in all different shapes and sizes. Although it is common to write and test your code on a single-server farm, this type of environment is probably not the type of farm in which your code will be deployed. It can be a big mistake to assume that your target SharePoint production environment is just like your development environment.

Many companies that are invested in on-premises SharePoint development categorize their farms into three different types. SharePoint developers write and debug SharePoint solutions in *development farms*. *Staging farms* simulate a more realistic environment and are used to conduct quality assurance testing on SharePoint solutions. For example, the servers in a staging farm should be built without installing developer tools such as Microsoft Visual Studio 2012. After a SharePoint solution or a SharePoint app has been thoroughly tested in a staging farm, it can be deployed in a *production farm*, where its functionality is made available to users.

Working with SharePoint 2013 Central Administration

As a SharePoint developer, you must wear many hats. One hat you frequently wear is that of a SharePoint farm administrator. You should become familiar with the administrative site that SharePoint Foundation automatically creates for each farm. This administrative site is known as *SharePoint 2013 Central Administration*, and its home page is shown in Figure 1-4.

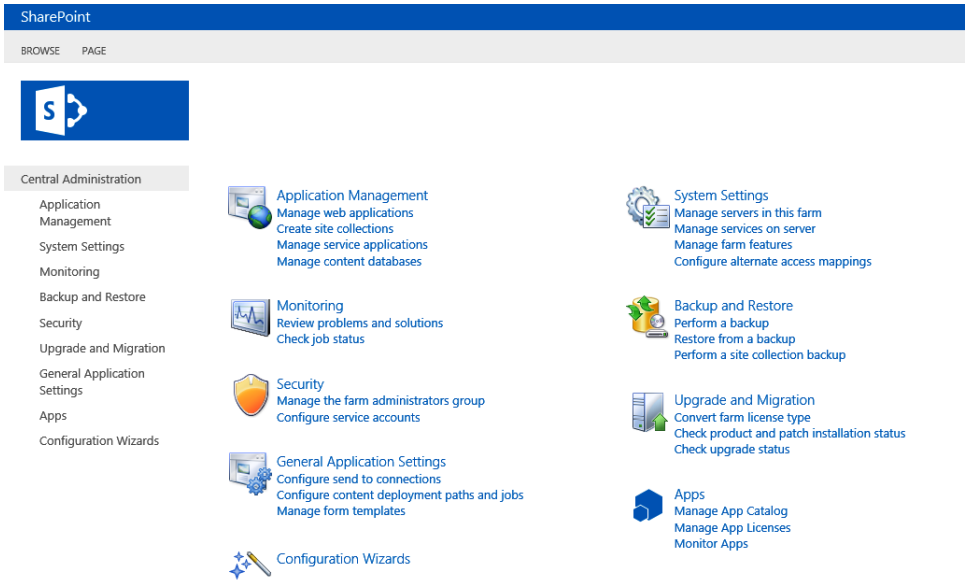


FIGURE 1-4 SharePoint developers should become familiar with SharePoint 2013 Central Administration.

Figure 1-4 shows the home page of SharePoint 2013 Central Administration in an on-premises farm with SharePoint Server 2013 installed. If you only install SharePoint Foundation instead of SharePoint Server 2013, you will not find as many links to administrative pages, because quite a few are only installed with SharePoint Server 2013. Also note that SharePoint 2013 Central Administration is extensible. If you need to create a SharePoint solution for administrative purposes, you canintegrate your work into SharePoint 2013 Central Administration by adding custom links and custom administration pages.

Scenario: Introducing Wingtip Toys

Many of the example configurations and code samples in this book are based on Wingtip Toys, a company that was fictitiously founded in 1882 by Henry Livingston Wingtip. Wingtip Toys has a long and proud history of producing the industry's most unique and inventive toys for people of all ages. Wingtip Toys has set up an intranet using SharePoint internally to provide a means of collaboration between its trinket design scientists, its manufacturing team, and its remote sales force. It has also erected an extranet, using SharePoint to interact with partners and toy stores around the world. Finally, Wingtip Toys has decided to use SharePoint to create its Internet-facing site to advertise and promote its famous line of toys and novelties.

Creating web applications

SharePoint 2013 is built on top of Internet Information Services (IIS). SharePoint 2013 is completely dependent upon IIS because it uses IIS websites to listen for and process incoming HTTP requests. Therefore, you need to understand exactly what an IIS website really is.

An IIS website provides an entry point into the IIS web server infrastructure. For example, the default website that is automatically created by IIS listens for incoming HTTP requests on port 80. You can create additional IIS websites to provide additional HTTP entry points using different port numbers, different IP addresses, or different host headers. In this book's scenario, we'll use host headers to create HTTP entry points for domain names such as *http://intranet.wingtiptoys.com*.

SharePoint Foundation creates an abstraction on top of IIS that is known as a *web application*. At a physical level, a SharePoint web application is a collection of one or more IIS websites configured to map incoming HTTP requests to a set of SharePoint sites. The web application also maps each SharePoint site to one or more specific *content databases*. SharePoint Foundation uses content databases to store site content such as list items, documents, and customization information.

Warning: Don't touch the SharePoint databases

When developing for SharePoint 2013, you're not permitted to directly access the configuration database or any of the content databases. For example, you must resist any temptation to write data access code that reads or writes data from the tables inside these databases. Instead, you should write code against one of the SharePoint 2013 APIs to reach the same goal, and leave it to SharePoint 2013 to access the configuration database and content database behind the scenes.

SharePoint Foundation leverages the ASP.NET 4.0 support in IIS to extend the standard behavior of an IIS website. It does this by configuring IIS websites to run SharePoint-specific components in the ASP.NET pipeline by using *HttpModule* objects. This integration with ASP.NET allows SharePoint Foundation to take control over every request that reaches an IIS website that has been configured as a SharePoint web application.

Keep in mind that every SharePoint web application runs as one large ASP.NET application. Consequently, SharePoint Foundation adds a standard ASP.NET web.config file to the root directory of each IIS website associated with a web application. When you create a web application in SharePoint Foundation, it creates an IIS website with a root folder containing a web.config file at the following location:

```
C:\inetpub\wwwroot\wss\VirtualDirectories
```

The fact that there is a one-to-many relationship between a web.config file and SharePoint sites can be counterintuitive for developers who are migrating from ASP.NET. A single SharePoint site is unlike an ASP.NET site because it can't have its own web.config file. That means that a single web.config file in SharePoint Foundation supplies configuration information for every site in a web application. This is true even in scenarios where the number of sites in a web application reaches into the hundreds or thousands.

A SharePoint on-premises farm typically runs two or more web applications. The first web application is created automatically when the farm is created. This web application is used to run SharePoint 2013 Central Administration. You need at least one additional web application to create the sites that are used by business users. The IT staff at Wingtip Toys decided to configure their production farm with three different web applications used to reach employees, partners, and customers, as shown in Figure 1-5.

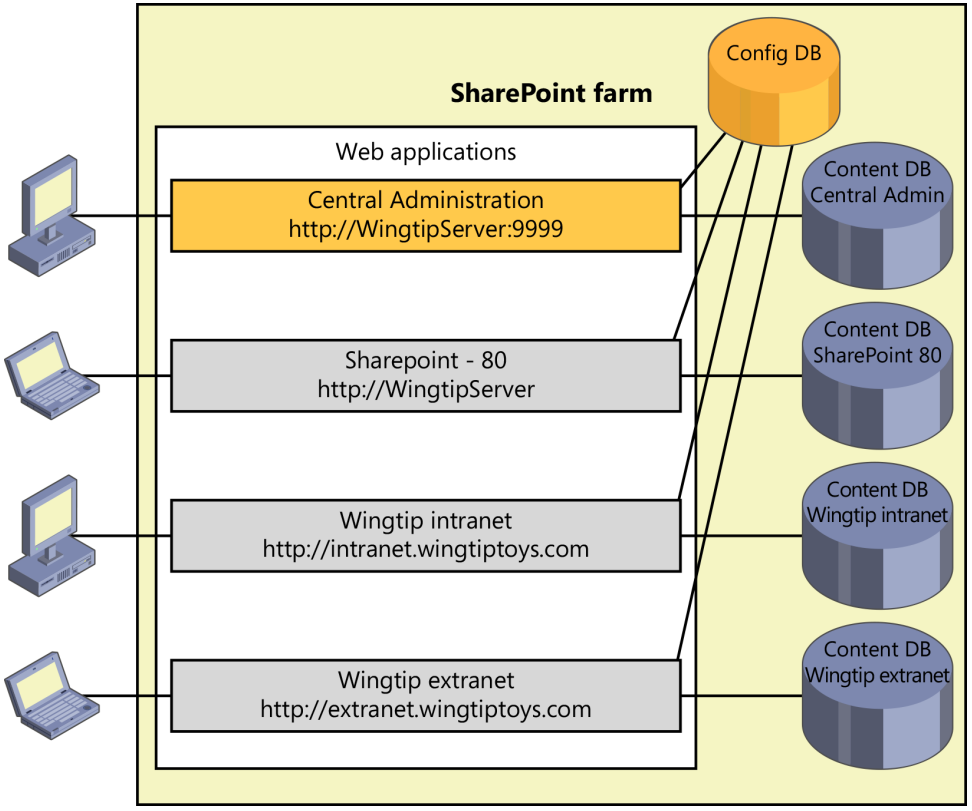


FIGURE 1-5 Each web application has one or more content databases.

Understanding web applications and user authentication

The first thing to understand is that the SharePoint platform itself does not supply the actual code to authenticate users. Instead, the SharePoint platform relies on external user authentication systems such as Windows Server and Active Directory or the built-in support in ASP.NET for forms-based authentication (FBA). After an external system has authenticated a user and created a security token, the SharePoint platform is then able to create a profile around that security token to establish and track the user's identity inside the SharePoint security system.

The manner in which SharePoint authenticates users is configured at the web application level. When you create a SharePoint web application, you have the option of creating it in either claims mode or classic mode. Classic authentication mode is the older style of user authentication that was used in SharePoint 2007, where user identity is tracked by using native Windows security tokens. Though classic mode is still supported in SharePoint 2013 for older scenarios, its use is deprecated and should be avoided. That means new web applications should be configured to use claims-based security.

The claims-based authentication mode was introduced in SharePoint 2010; it allows the SharePoint platform to use a single, unified format for all the security tokens that are created during the user

authentication process. More specifically, the user authentication tokens are converted into a special format for caching known as a *FedAuth token*. Within developer circles, a FedAuth token is also commonly referred to as a *claims token*.

Let's walk through the authentication process in a SharePoint web application in a scenario in which the user is authenticated with Windows authentication. The first part of the authentication process involves creating a native Windows security token. In the second part of the authentication process, SharePoint Foundation will convert the Windows security token into a FedAuth token by using a local service known as the *Security Token Service (STS)*.

You also have the option of configuring a web application in an on-premises farm to support forms-based authentication by using an ASP.NET authentication provider. In this style of authentication, SharePoint Foundation once again calls upon the STS to create a FedAuth token for the FBA user during the user authentication process.

In SharePoint 2010, the FedAuth tokens created during the user authentication process are cached in memory on a per–web server basis and can be reused across multiple requests from the same user. SharePoint 2013 further optimizes the caching of FedAuth tokens with a new platform-level service known as the Distributed Cache Service, which can be configured to maintain a farm-wide cache of FedAuth tokens.

SharePoint Foundation's use of claims-based authentication and FedAuth tokens provides another noteworthy point of flexibility. It opens up the number of identity providers that can be integrated with a SharePoint farm to provide user authentication. In addition to supporting Windows authentication and FBA, claims-based security makes it possible to configure a SharePoint web application to authenticate users by using external identity providers that support an XML-based industry standard known as *Security Assertion Markup Language (SAML)*. More specifically, SharePoint 2013 supports identity providers that support the SAML 1.1 specification. Examples of supported providers include Windows Azure Access Control Service (ACS), Windows Live ID, Google Single Sign-on, and Facebook.

Now that you have learned the fundamentals of how web applications provide the support for user authentication, let's examine how you might configure a set of web applications in a real-world scenario. For example, imagine a scenario in which the IT staff at Wingtip Toys must decide how many web applications should be created in their production farm.

The Wingtip Toys IT staff decided to create the first web application for the exclusive use of Wingtip employees, all of whom have their own Active Directory user accounts. Therefore, the first web application was configured for intranet usage by requiring Integrated Windows authentication and by prohibiting anonymous access.

The Wingtip Toys IT staff decided to create a second web application so they could create sites that could be made accessible to external users such as partners and vendors. The key characteristic of these external users is that they will never have their own Active Directory user accounts and, therefore, cannot be authenticated by using Windows authentication. Therefore, the Wingtip Toys IT staff decided to configure the second web application to support user authentication using FBA, so that these external users can be authenticated without any need for Active Directory user accounts.

The Wingtip Toys IT staff decided to create a third web application to host any SharePoint site that requires anonymous access, such as their public website hosted at *http://www.wingtiptoys.com*. Although they configured this web application to allow visitors from the Internet to view their public website anonymously, they also wanted to make logging onto the site an available option so that customers could create member accounts and customer profiles. Therefore, they configured this web application with a trust to Windows Live ID. When customers attempt to log onto the Wingtip Toys public website, they are redirected to the Windows Live ID site and prompted to enter their Windows Live ID credentials. After the customer is authenticated by Windows Live ID, he is then redirected back to the Wingtip Toys public website with an established identity.

Understanding service applications

A SharePoint farm must provide an efficient way to share resources across sites running in different web applications. It must also provide the means for offloading processing cycles for certain types of processes from front-end web servers to dedicated application servers. To meet this requirement, SharePoint Foundation uses an architecture based on *service applications* that was introduced in SharePoint 2010. Service applications are used to facilitate sharing resources across sites running in different web applications and different farms. The service application architecture also provides the means for scaling a SharePoint farm by offloading processing cycles from the front-end web servers over to dedicated application servers in the middle tier.

A key benefit of the service application architecture is that you can treat a service application as a moveable entity. After you create a service application, you can configure it for several different deployment scenarios. In a simple two-tier farms, the service application can be configured to run on one or more of the web servers in the farm, as shown on the left in Figure 1-6. In scenarios that require the ability to scale to thousands of users, the same service application can be configured to run on a dedicated application server such as the one shown on the right in Figure 1-6. In scenarios that require even greater scale, a service application can be configured to run within its own dedicated farm of application servers.

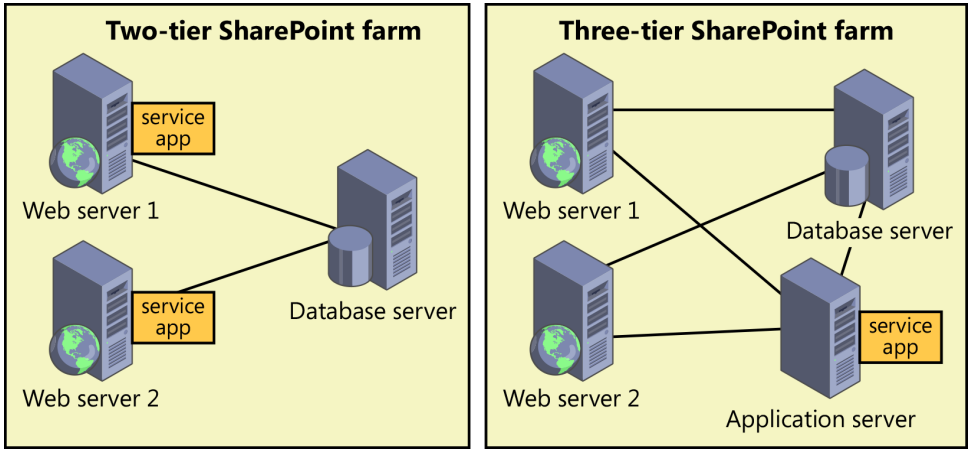


FIGURE 1-6 SharePoint farms run service applications in addition to web applications.

The service application architecture of the SharePoint platform was created with extensibility in mind. Any developer with the proper knowledge and motivation can develop a service application that can be deployed within a SharePoint 2013 farm. However, this is not an easy undertaking. A service application targeting a SharePoint platform must be written to a specific set of requirements. For example, a service application must query the configuration database about its current deployment configuration and adjust its behavior accordingly. A service application must be written in such a way that it can be deployed and configured using nothing more than Windows PowerShell.

When a service application runs across the network on a dedicated application server, it relies on a proxy component that must be written to run on the web server. The service application proxy is created and configured along with the service application. The service application proxy provides value by abstracting away the code required to discover where the service application lives on the network. The service application proxy provides additional value by encapsulating the Windows Communication Foundation (WCF) code used to execute web service calls on the target service application.

The proxy-based design of service applications provides flexibility in terms of deployment and configuration. For example, you can configure a proxy in one farm to communicate with a service application in another farm. The proxy simply consults the configuration database and discovers the correct address for the application server running the service application. The implication here is that the new service application architecture makes it much easier to share resources across farms while still controlling what services are made available and how they are consumed.

As a SharePoint developer creating business solutions, it is unlikely that you would ever find the need or have the proper incentives to develop a custom SharePoint service application. However, you still need to understand how service applications work and how they fit into the high-level architecture of SharePoint Foundation. For example, SharePoint Server 2013 delivers a good deal of its functionality through service applications.

The key point here is that you must learn how to create and configure service applications and service application proxies to properly build out a local on-premises farm for SharePoint development. This can be done most easily by using the Farm Configuration Wizard, which is available in Central Administration. However, using a custom Windows PowerShell script allows you to create service applications and service application proxies with far more control and flexibility than is afforded by the Farm Configuration Wizard.

Building an environment for SharePoint development

If you plan on developing SharePoint solutions or SharePoint apps that will be used within private networks such as a corporate LAN, it makes sense to build out a development environment with a local SharePoint 2013 farm. Critical Path Training provides a free download called the *SharePoint Server 2013 Virtual Machine Setup Guide*, which provides you with step-by-step instructions to install all the software you need and to build out a local SharePoint 2013 farm. You can download the guide from *http://criticalpathtraining.com/Members*.

Creating service applications in SharePoint Server 2013

SharePoint Server 2013 is nothing more than a layer of software that's been written to run on SharePoint Foundation. Every installation of SharePoint Server 2013 begins with an installation of SharePoint Foundation. After installing SharePoint Foundation, the installation for SharePoint Server 2013 then installs its own templates, components, and service applications. The Standard edition of SharePoint Server 2013 only supports a subset of the features and services available in the Enterprise edition of SharePoint Server 2013.

Adding to the complexity is that the feature set of SharePoint Online does not exactly match that of the on-premises version of SharePoint Server 2013. Therefore, you can really break SharePoint 2013 out into four distinct platforms that all vary in some degree from one another:

- SharePoint Foundation 2013
- SharePoint Server 2013 Standard edition
- SharePoint Server 2013 Enterprise edition
- SharePoint Online

To help you understand which service applications are available in each variation of the SharePoint 2013 platform, Table 1-2 lists some of the SharePoint 2013 service applications in addition to the editions of SharePoint 2013 that support each of these service applications.

TABLE 1-2 Service applications included with SharePoint 2013 platform

Name	Foundation	Standard	Enterprise	Online
Access Services	No	No	Yes	Yes
Access Services 2010	No	No	Yes	No
App Management Service	Yes	Yes	Yes	Yes
Business Data Connectivity Service	Yes	Yes	Yes	Yes
Excel Services Application	No	No	Yes	Yes
Machine Translation Service	No	No	Yes	Yes
PerformancePoint Service Application	No	No	Yes	No
PowerPoint Automation Services	No	Yes	Yes	Yes
Managed Metadata Service Application	No	Yes	Yes	Yes
Search Service Application	Yes	Yes	Yes	Yes
Secure Store Service	No	Yes	Yes	Yes
Site Subscription Settings Service	Yes	Yes	Yes	Yes

Name	Foundation	Standard	Enterprise	Online
State Service	Yes	Yes	Yes	Yes
User and Health Data Collection Service	Yes	Yes	Yes	Yes
User Profile Service Application	No	Yes	Yes	Yes
Visio Graphics Service	No	No	Yes	Yes
Word Automation Services	No	Yes	Yes	Yes
Work Management Service Application	No	Yes	Yes	Yes
Workflow Service Application	Yes	Yes	Yes	Yes

Managing sites

Now that you understand the high-level architecture of a SharePoint farm, you need to know how SharePoint Foundation creates and manages sites within the scope of a web application. Let's start by asking a basic question: What exactly is a SharePoint site?

This question has many possible answers. For example, a site is an endpoint that is accessible from across a network such the Internet, an intranet, or an extranet. A site is also a storage container that allows users to store and manage content such as list items and documents. In addition, a site is a customizable entity that allows privileged users to add pages, lists, and child sites as well as install SharePoint apps. Finally, a site is a securable entity whose content is accessible to a configurable set of users.

As a developer, you can also think of a site as an instance of an application. For example, the scientists at Wingtip Toys use a SharePoint site to automate the business process of approving a new toy idea. When Wingtip scientists have new ideas for a toy, they describe their ideas in Microsoft Word documents, which they then upload to a document library in the site. The approval process is initiated whenever a scientist starts a custom approval workflow on one of those documents.

A site can also be used as an integration point to connect users to back-end data sources such as a database application or a line-of-business application such as SAP or PeopleSoft. The Business Connectivity Services that ship with SharePoint 2013 make it possible to establish a read-write connection with a back-end data source. One valuable aspect of the Business Connectivity Services architecture is that this external data often appears to be a native SharePoint list. There are many user scenarios and developer scenarios in which you can treat external data just as you would treat a native SharePoint list.

Understanding the role of site collections

Every SharePoint site must be provisioned within the scope of an existing web application. However, a site can't exist as an independent entity within a web application. Instead, every site must also be created inside the scope of a site collection.

A *site collection* is a container of sites. Every site collection has a top-level site. In addition to the top-level site, a site collection can optionally contain a hierarchy of child sites. Figure 1-7 shows a web application created with a host header path of *http://intranet.wingtiptoys.com* that contains four site collections. The first site collection has been created at the root of the web application and contains just a single, top-level site. Note that the top-level site, the site collection, and the hosting web application all have the same URL.

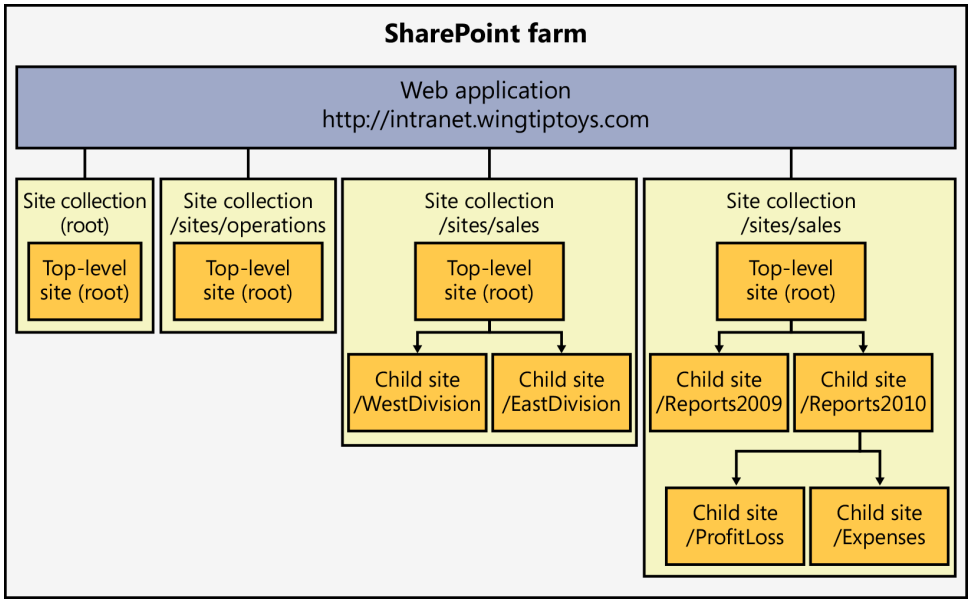


FIGURE 1-7 Each site collection has a top-level site and can optionally contain a hierarchy of child sites.

Only one site collection within a web application can be created at the same URL as the hosting web application itself. The other three site collections shown in Figure 1-7 have been created at URLs that are relative to the host header path of the hosting web application. The site collection created at the relative path of /sites/operations has just a top-level site. The site collection created at the relative path of /sites/sales contains one level of child sites below the top-level site. The last site collection on the right, which has been created at the relative path of /sites/financials, contains a more complex hierarchy with three levels.

When a company begins using SharePoint Foundation or SharePoint Server 2013, one of the first questions that comes up is how to partition sites across site collections. For example, should you create one big site collection with lots of child sites, or should you create many individual site collections? This decision is usually best made after thinking through all the relevant issues discussed in the next few paragraphs. You must gain an understanding of how partitioning sites into site collections affects the scope of administrative privileges, security boundaries, backup and restore operations, and site design.

You could be asking yourself why the SharePoint Foundation architecture requires this special container to hold its sites. For starters, site collections represent a scope for administrative privileges. If

you've been assigned as a site collection administrator, you have full administrative permissions within any existing site and any future site created inside that site collection.

Think about the requirements of site management in a large corporation that's provisioning thousands of sites per year. The administrative burden posed by all these sites is going to be more than most IT staffs can deal with in a timely manner. The concept of the site collection is important because it allows the IT staff to hand off the administrative burden to someone in a business division who takes on the role of the site collection administrator.

Let's walk through an example. The Wingtip Toys IT staff is responsible for provisioning new site collections, and one of the Wingtip business divisions submits a request for a new site. Imagine the case where the Wingtip Sales Director has put in a request to create a new team site for his sales staff. A Wingtip IT staff member would handle this request by creating a new site collection with a team site as its top-level site.

When creating the new site collection, the Wingtip IT staff member would add the Wingtip Sales Director who requested the site as the site collection administrator. The Wingtip Sales Director would have full administrative privileges inside the site collection and could add new users, lists, and pages without any further assistance from the Wingtip IT staff. The Wingtip Sales Director could also add child sites and configure access rights to them independently of the top-level site.

A second advantage of site collections is that they provide a scope for membership and the configuration of access rights. By design, every site collection is independent of any other site collection with respect to what security groups are defined, which users have been added as members, and which users are authorized to perform what actions.

For example, imagine that the Wingtip IT staff has provisioned one site collection for the Sales department and a second site collection for the Finance department. Even though some users within the Finance department have administrative permissions within their own site collection, there's nothing they can do that will affect the security configuration of the Sales site collection. SharePoint Foundation sees each site collection as an island with respect to security and permissions configuration.

A third reason for site collections is that they provide a convenient scope for backup and restore operations. You can back up a site collection and later restore it with full fidelity. The restoration of a site collection can take place in the same location where the backup was made. Alternatively, a site collection can be restored in a different location—even inside a different farm. This technique for backing up a site collection and restoring it in another location provides one possible strategy for moving sites and all the content inside from one farm to another.

A final motivation for you to start thinking about in terms of site collections is that they provide a scope for many types of site elements and for running custom queries. For example, the server-side object model of SharePoint Foundation provides you with the capability to run queries that span all the lists within a site collection. However, there is no query mechanism in the SharePoint server-side object model that spans across site collections. Therefore, if your application design calls for running queries to aggregate list data from several different sites, it makes sense to add sites to the same site collection when they contain lists that must be queried together.

Imagine a case in which the West Division of the Wingtip Sales team has four field offices. The Wingtip Sales Director could create a child site for each field office below a site that was created for the West Division. Now assume that each child site has a Contacts list that is used to track sales leads. By using programming techniques shown later in this book, you can execute queries at the scope of the West Division site that would aggregate all the Contacts items found across all of its child sites. You could execute the same query at a higher scope and get different results. For example, if you executed the same query scoped to the top-level site, it would aggregate all the Contacts found throughout the site collection, including both the West Division and the East Division.

Understanding host-named site collections (HNSCs)

The traditional way to manage the URLs of site collections is to create the hosting web application with a host header path such as http://intranet.wingtiptoys.com. The site collections created inside this type of web application are known as path-based site collections because they all must be created with a URL that starts with the same host header path. When you create path-based site collections, you must create the URL for each site collection by starting with the host header path defined by the hosting web application:

- *http://intranet.wingtiptoys.com*
- *http://intranet.wingtiptoys.com/sites/operations*
- *http://intranet.wingtiptoys.com/sites/sales*
- *http://intranet.wingtiptoys.com/sites/financials*

There is a second approach, which provides more flexibility when you are managing the URLs for the site collections with a web application. This approach requires you to create the hosting web application without the traditional host header path. When you create a new web application without the host header path, you then have the ability to create site collections with unique host names. This type of site collection is known as a host-named site collection (HNSC).

Consider the following scenario. Imagine you are required to create a set of site collections using the following domain names:

- *http://operations.wingtiptoys.com*
- *http://sales.wingtiptoys.com*
- *http://financials.wingtiptoys.com*

If you use the older, traditional approach of creating path-based site collections, you would have to create a separate web application to host each of these site collections. However, this approach is going to become problematic because it cannot be scaled due to the fact that you are limited in how many web applications can be created within a single farm. However, if you use an approach based on host-named site collections, you can create all these site collections and many more with unique domain names within a single web application.

Note that the creation of host-named collections can be a little tricky at first. That's because a host-named site collection cannot be created through Central Administration. You must create a host-named site collection by using Windows PowerShell.

Customizing sites

SharePoint Foundation provides many user options for configuring and customizing sites. If you're logged onto a site as the site collection administrator, site administrator, or a user granted Designer permissions, you can perform any site customization options supported by SharePoint Foundation. If you're logged onto a site without administrative privileges in the role of a contributor, however, you won't have the proper permissions to customize the site. Furthermore, if you're logged on as a contributor, SharePoint Foundation uses *security trimming* to remove the links and menu commands that lead to pages with functionality for which you don't have permissions.

If you're logged onto a standard team site as a site administrator, you should be able to locate and open the Site Actions menu by clicking the small gear icon in the upper-right corner of the page, as shown in Figure 1-8. Note that the gear icon of the SharePoint Site Action menu is easy to confuse with the gear icon displayed by Windows Internet Explorer, which provides a menu that can be used to configure browser settings. Remember that the lower gear icon is specific to SharePoint and the one above it is specific to Internet Explorer.

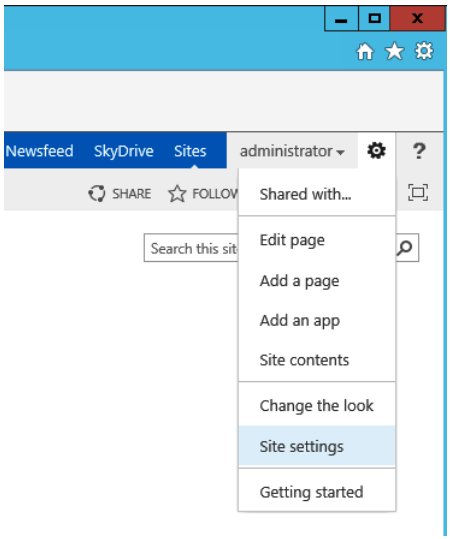


FIGURE 1-8 SharePoint sites provide the Site Actions menu to users with the correct permissions.

The Site Actions menu provides commands that allow you to edit the current page; to create new pages, lists, and document libraries; to view site contents; to change the look and feel of the current site; and to navigate to the Site Settings page shown in Figure 1-9.

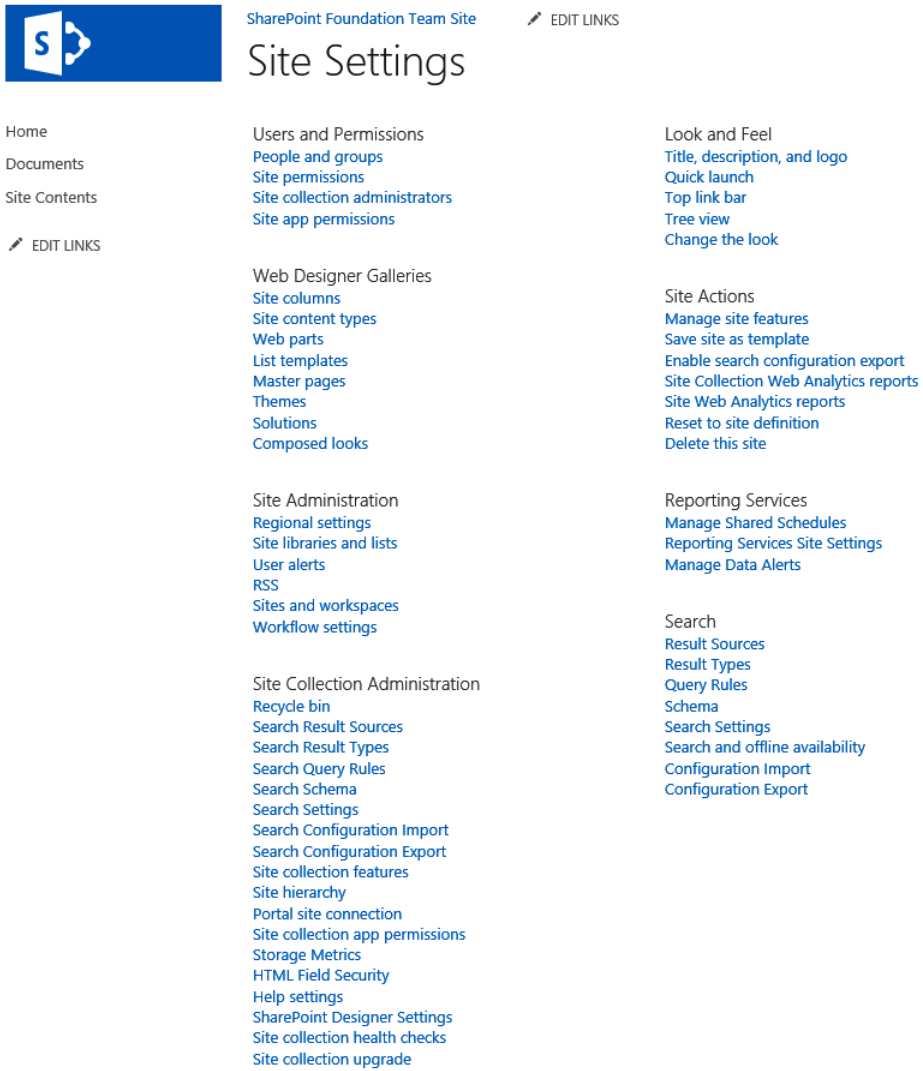


FIGURE 1-9 The Site Settings page is accessible to site administrators on any site.

The Site Settings page provides links to pages that allow you to perform various administrative and customization tasks. Notice that the Site Settings page for a top-level site contains one section for Site Administration and a second section for Site Collection Administration. The Site Settings page for child sites doesn't include the section for Site Collection Administration.

Figure 1-9 shows several sections of links, including Users and Permissions, Look and Feel, Web Designer Galleries, Site Actions, Site Administration, and Site Collection Administration, all of which provide links to various other administrative pages for the current site. If you're new to SharePoint Foundation, you should take some time to explore all the administrative pages accessible through the Site Settings page. Also keep in mind that Figure 1-9 shows only the links on the Site Settings page

of a team site running within a SharePoint Foundation farm. If the site were running in a SharePoint Server 2013 farm, there would be additional links to even more site administration pages that are not part of the standard SharePoint Foundation installation.

Creating and customizing pages

The support for wiki page libraries and Web Parts is an aspect of SharePoint Foundation that enables business users to make quick changes to the content on pages in a SharePoint site. Business users with no experience in web design or HTML can quickly add and customize webpages. A good example of this can be seen when creating a new SharePoint 2013 team site. As part of the provisioning process, SharePoint Foundation automatically creates a new wiki library at the SitePages path off the root of the site, and it adds a wiki page named Home.aspx. It additionally configures Home.aspx to be the home page of the site, so it becomes the first page users see when navigating to the site.

Customizing the home page is simple for any user who has the proper permissions. The user can enter edit mode by using either the Site Actions menu or the ribbon. When in edit mode, the user is free to simply type text or copy and paste from another application. The Insert tab on the ribbon also makes it easy for the user to add tables, links, and images.

Web Part technology also plays a prominent role in page customization. Web Parts are based on the idea that the SharePoint platform and developers supply a set of visual components that users can add and move around in their pages. Every site collection has a Web Part Gallery, which contains a set of Web Part template files. This set of Web Part template files determines which types of Web Parts can be added to pages within the site collection.

Although earlier versions of SharePoint technologies supported Web Parts, they were not as flexible as SharePoint Foundation because Web Parts could be added only to Web Part pages. Starting with SharePoint 2010, SharePoint Foundation has made it possible to add Web Parts anywhere inside a wiki page. When you're editing the content of a wiki page, you can place the cursor wherever you want and add a new Web Part by using the Insert tab on the ribbon. The new Web Part appears inline along with your other wiki content.

Creating and customizing lists

The Site Actions menu provides an Add A Page menu command for creating new pages and an Add An App menu command for creating new lists and document libraries. If you click the Add An App menu command in the Site Actions menu, SharePoint Foundation displays the Add An App page, which allows you to create a new list or document library, as shown in Figure 1-10.

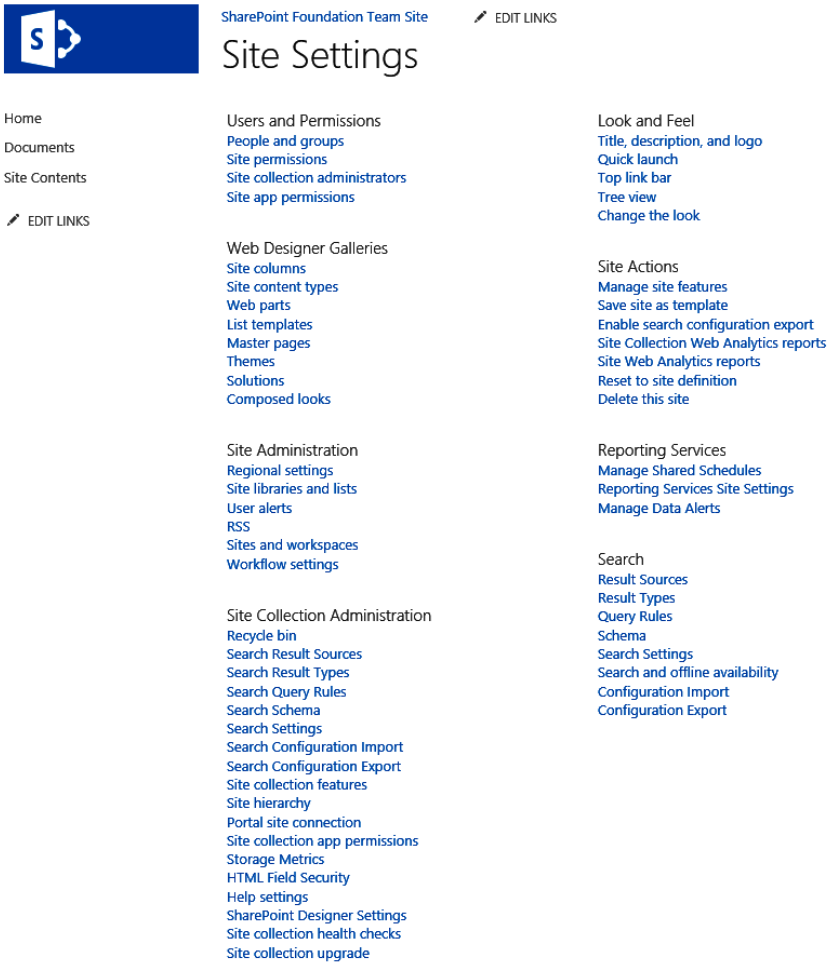


FIGURE 1-10 From the Add An App page, you can create new lists and document libraries.

In addition to list templates, the standard collaboration features of SharePoint Foundation also include templates for creating several different types of document libraries. Besides the standard document library type, there are also more specialized document library types for wiki page libraries, picture libraries, and InfoPath form libraries.

What's appealing to SharePoint users is that after they create a new list, it's immediately ready to use. SharePoint Foundation provides instant gratification by including page templates as part of the list template itself, making it possible to create each new list and document library with a set of pages that allow users to add, view, modify, and delete items and documents.

After a list has been created, SharePoint Foundation gives a user the flexibility to further customize it. SharePoint Foundation provides a List Settings page for each list and document library. Figure 1-11 shows a typical List Settings page. It provides a set of links to secondary pages that allow the user to modify properties of a list such as its title and description and to configure other important aspects of

the list, including versioning, workflow, and security permissions. The List Settings page also provides links to add and manage the set of columns behind the list.

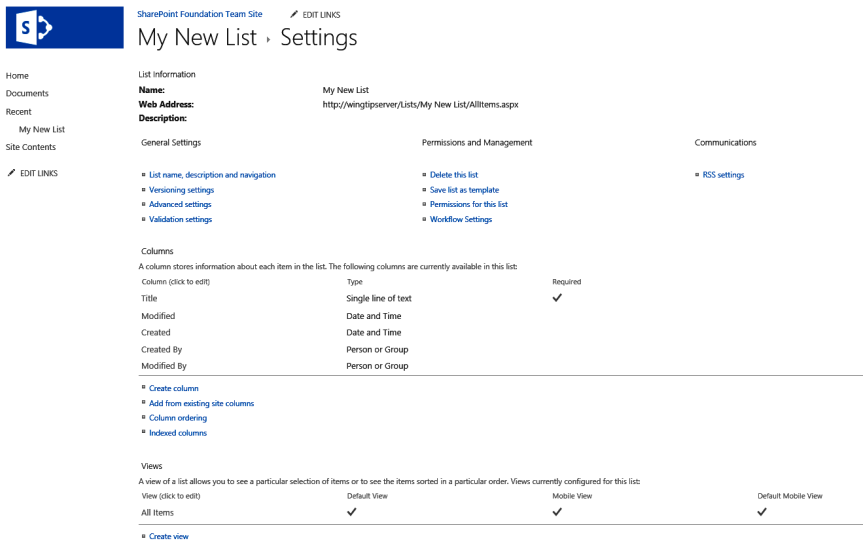


FIGURE 1-11 The List Settings page allows you to modify list properties and to add columns.

SharePoint Foundation provides many built-in list templates to track information about common business items such as tasks, contacts, and scheduled events. For business scenarios in which the list data that needs to be tracked doesn't conform to a built-in list template, SharePoint Foundation makes it easy for a user to create a custom list with a unique set of columns for these ad hoc situations.

SharePoint Foundation provides a list template named Custom List. When you create a new list from this template, it will initially contain a single column named Title. A user can add columns with just a few mouse clicks. Each added column is based on an underlying field type. SharePoint Foundation supplies a rich set of built-in field types for columns whose values are based on text, numbers, currency, dates, and yes/no values.

Using SharePoint Designer 2013

Microsoft SharePoint Designer 2013 is a specialized site customization tool. It is a rich desktop application that is often easier to use for customizing a site than a browser is. SharePoint Designer 2013 is a free product that can be downloaded from the following URL:

http://www.microsoft.com/en-us/download/details.aspx?id=35491

If you have used a previous version of SharePoint Designer, you might be surprised to find that the editor window for customizing pages no longer supplies a Design View. The Design View feature of the page editor has been discontinued in SharePoint Designer 2013. This means that you only have a Code View editor when working on site pages, master pages, and page layouts.

SharePoint Designer 2013 is primarily designed to assist users who have been granted Designer permissions or have been put in the role of site collection administrator or site administrator. The tool makes it quick and easy to examine the properties and structure of a site and to perform common site tasks such as adding security groups and configuring permissions. Many users will also prefer the experience of SharePoint Designer 2013 over the browser when it comes to creating new lists and adding columns.

SharePoint Designer 2013 also allows a user to perform site customizations that aren't possible through the browser. The ability to create and customize custom workflow logic by using a new set of workflow designers provides a great example. By using SharePoint Designer 2013, an experienced user can create and design complex workflows on targets such as sites, lists, and document libraries.

In Chapter 12, "SharePoint workflows," you will learn that SharePoint Designer 2013 supports the creation of custom workflows in both the new SharePoint 2013 format as well as the older SharePoint 2010 format. You will learn how custom workflows created with SharePoint Designer can be packaged and reused across site collections, web applications, and farms.

Understanding site customization vs. SharePoint development

In one sense, SharePoint Foundation lessens the need for professional software developers because it empowers users to create and customize their own sites. In minutes, a user can create a SharePoint site, add several lists and document libraries, and customize the site's appearance to meet the needs of a particular business situation. An identical solution that has all the rich functionality that SharePoint Foundation provides out of the box would typically take an ASP.NET development team weeks or months to complete.

In another sense, SharePoint Foundation provides professional developers with new and exciting development opportunities. As with any other framework, the out-of-the-box experience with SharePoint Foundation takes you only so far. At some point, you'll find yourself needing to create custom list types and write code for custom SharePoint components such as Web Parts and event handlers. What is attractive about SharePoint Foundation as a development platform is that it was designed from the ground up with developer extensibility in mind.

As you begin to design software for SharePoint 2013, it is critical that you differentiate between *customization* and *development*. SharePoint Foundation is very flexible for users because it was designed to support high levels of customization. As we've pointed out, you no longer need to be a developer to build a complex and highly functional website. Today, many sophisticated users are capable of customizing SharePoint sites for a large number of business scenarios. Site customization has its limitations, however. SharePoint Foundation records every site customization by modifying data within a content database, whether a new list is created or an existing list is customized with new columns and views. All types of site customization that can be performed by using SharePoint Designer 2013 are recorded this way.

The fact that all site customization is recorded as a modification to the content database is both a strength and a weakness for SharePoint Foundation. It is a strength because it provides so much flexibility to users and site administrators doing ad hoc customizations. It is a weakness from the perspective of a professional software developer because customization changes are hard to version and can also be hard or impossible to make repeatable across site collections and farms.

Think about a standard ASP.NET development project in which all the source files you're working with live within a single directory on your development machine. After you've finished the site's initial design and implementation, you can add all the site's source files to a source control management system such as Team Foundation Server.

By using a source control management system, you can formalize a disciplined approach to deploying and updating an ASP.NET site after it has gone into production. You can also elect to push changes out to a staging environment where your site's pages and code can be thoroughly tested before they are used in the production environment.

As a developer, you should ask yourself the following questions: How do I conduct source control management of customization changes? How do I make a customization change to a list definition or a page instance and then move this change from a development environment to a staging environment and finally to a production environment? How do I make a customization change within a site and then reuse it across a hundred different sites? Unfortunately, these questions have tough answers, and usually you'll find that a possible solution isn't worth the trouble.

Fortunately, as a developer, you can work at a level underneath the SharePoint Foundation customization infrastructure. To be more specific, you can create a SharePoint farm solution that allows you to work with the low-level source files to create underlying templates for items such as pages and lists. These low-level source files don't live inside the content database; instead, they live within the file system of the front-end web server.

Working at this level is complex and has a steep learning curve. Even so, this low-level approach lets you centralize source code management and have a more disciplined approach to code sign-off when moving functionality from development to staging to production. This approach also makes versioning and reuse of code far more manageable across multiple sites, web applications, and farms.

For the remainder of this book, we differentiate between customization and development according to these criteria. SharePoint site customizations are updates to a site accomplished by making changes to the content database, generally through the web browser or SharePoint Designer 2013. A site customization never requires the front-end web server to be touched.

SharePoint development, on the other hand, often involves working with farm solutions that include files that must be deployed to the file system of the front-end web server. In Chapter 3, "Server-side solution development," we introduce SharePoint solutions and discuss best practices for how to package a development effort for deployment within a SharePoint 2013 farm. In Chapter 4, "SharePoint apps," we introduce the alternative development approach of creating SharePoint apps. You will learn that the two approaches are quite different.

Windows PowerShell boot camp for SharePoint professionals

SharePoint 2013 is the second version of SharePoint technologies in which Microsoft supports administration through Windows PowerShell scripts. In earlier versions of SharePoint, farm administrators must use a command-line utility named stsadm.exe to run interactive commands from the console window and to write MS-DOS–style batch file scripts to automate common administrative tasks such as creating, backing up, or restoring a new site collection.

SharePoint Foundation still installs the stsadm.exe utility, but it is primarily included to support backward compatibility with pre-existing scripts migrated from earlier versions. Microsoft recommends using the Windows PowerShell support for writing, testing, and executing scripts that automate the same types of administrative tasks that you can accomplish by using stsadm.exe, plus a whole lot more.

The Windows PowerShell support for SharePoint Foundation adds a new required skill for every farm administrator and every developer moving to SharePoint 2010 or SharePoint 2013. You're now required to be able to read, write, and execute Windows PowerShell scripts to automate tasks such as creating a new web application or a new site collection.

Given the expected percentage of readers without any prior experience with Windows PowerShell, we decided to conclude Chapter 1 with a fast and furious Windows PowerShell boot camp. Our goal here is to get you up to speed on Windows PowerShell so that you can start reading, writing, executing, and debugging Windows PowerShell scripts. So fasten your seat belt.

Learning Windows PowerShell in 21 minutes

Working with Windows PowerShell is much easier than writing MS-DOS–style batch files. It's easier because the Windows PowerShell scripting language treats everything as an object. You can create and program against .NET objects as well as COM objects. Furthermore, Windows PowerShell has first-rate support for calling out to EXE-based utilities and passing parameters to execute specific commands.

There are two common ways in which you can use Windows PowerShell. First, you can execute commands interactively by using the Windows PowerShell console window. Second, you can write scripts to automate administration tasks. Then you can execute these scripts either on demand or through some type of scheduling mechanism.

Let's first get familiar with the Windows PowerShell console window. In Windows Server 2012, press the Windows logo key and then type **PowerShell**. In Windows Server 2008 R2, you can launch the Windows PowerShell console window from the following path from the Windows Start menu:

Start\All Programs\Accessories\Windows PowerShell\Windows PowerShell

When the Windows PowerShell console appears, type and execute the following three commands interactively:

1. Type **cd** and then press Enter. This sets the current location to the root of drive C.

2. Type **cls** and then press Enter. This clears the console window.

3. Type **2 + 2** and then press Enter. This performs a mathematical calculation and displays the result.

If you followed these steps correctly and executed each of the three commands, your console window should look like the one in Figure 1-12.

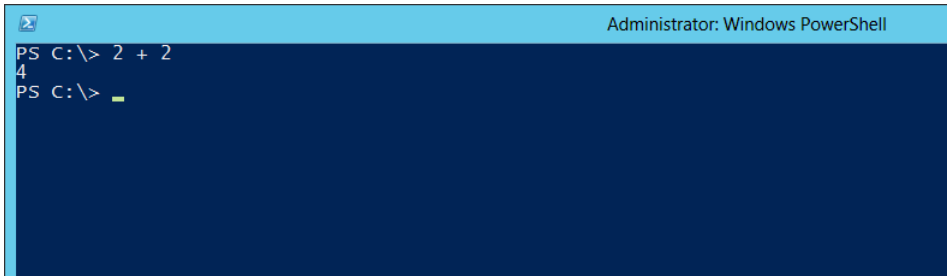


FIGURE 1-12 You can execute commands interactively from the Windows PowerShell console window.

Congratulations! You've just completed your first lesson. Now you know how to execute a command interactively from the Windows PowerShell console window. You simply type the command at the cursor in the Windows PowerShell console window and press Enter.

Windows PowerShell is based on reusable libraries containing functions known as *cmdlets* (pronounced "command lets"). Cmdlets have names that follow the convention of a common verb followed by a noun. For example, the built-in Windows PowerShell libraries provide a cmdlet named *Get-Process*, which returns a collection of objects representing the Windows processes running on the current machine:

```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     94       9     2780      13448    99     0.33   6592 conhost
     83       9     1588       3652    42     0.03   8188 csrss
    130      13     2020       5608    31     0.02   1312 dfssvc
   1126     768  1172456     484268  1693    56.23   1176 DistributedCacheService
   5261    6161    87624      86352   128     0.88   1636 dns
   1197      76    27332      82044   398    11.73   3696 explorer
    531      39     9136      34820   183     2.08   5348 iexplore
    712      53    46252      56792   598     2.06   1504 Microsoft.ActiveDirectory.WebServices
   2045     743   517044     492284  -631    19.44   6748 noderunner
   1954     757   557248     503572  -649    17.98   6796 noderunner
    461     238   343372     383888  1201   152.95   4828 OWSTIMER
    594      39   113076     119668   653     3.03   6676 powershell
    867     473  1674072      94884   552   284.56   1852 sqlservr
     82       8     1388       5296    38     0.02   2016 sqlwriter
   1536     657   378000     210600  1767     8.16   3852 w3wp
   1715     794   916880     758228 -1882    29.83   5244 w3wp
   1672     657   226404     192840  1772     7.33   5360 w3wp
    335      32    32612      36616   547    17.83   2816 WSSADMIN
```

Pipelining is an important concept to understand when you are executing cmdlets. The basic idea is that every cmdlet returns an object or a collection of objects. Pipelining allows you to take the output results of one cmdlet and pass it as an input parameter to a second cmdlet. The second cmdlet can execute and then pass its output results to a third cmdlet, and so on. You create a pipeline by typing a sequence of cmdlets separated by the | (pipe) character:

```
cmdlet1 | cmdlet2 | cmdlet3
```

Let's examine a common scenario in which you need to create a pipeline of two cmdlets to filter a collection of objects. First you call *Get-Process* to return a collection of objects, and then you use pipelining to pass this collection of objects to the *Where-Object* cmdlet:

```
PS C:\> Get-Process | Where-Object {$_.ProcessName -like "w*"}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    977     135   173372     180504  1511     4.94   2176 w3wp
    773     123   161220     164464  1485     3.36   5112 w3wp
    270      31    25052      17860   496     0.14   2568 WSSADMIN
```

The *Where-Object* cmdlet takes a predicate expression enclosed in curly braces as a parameter. Inside these curly braces, you can use *$_* to refer to an object as it's being filtered. The predicate expression in this example is *{$_.ProcessName -like "w*"}*. The filter returns all processes whose process name starts with "w".

Windows PowerShell cmdlets such as *Where-Object* use standard Windows PowerShell comparison operators. You should memorize these operators because you'll be using them regularly as you work with Windows PowerShell. Table 1-3 lists some commonly used Windows PowerShell comparison operators.

TABLE 1-3 Commonly used Windows PowerShell comparison operators

Operator	Purpose
-lt	Less than
-le	Less than or equal to
-gt	Greater than
-ge	Greater than or equal to
-eq	Equal to
-ne	Not equal to
-like	Like, using wildcard matches
-notlike	Not like, using wildcard matches

You should understand that Windows PowerShell comparison operators that work with strings are case insensitive by default. However, these operators can be made case sensitive by adding a *c* immediately after the hyphen. For example, *-ceq* represents the case-sensitive *equal-to* operator.

Writing Windows PowerShell scripts

Now that you've seen how to execute cmdlets from the Windows PowerShell console window, it's time to move on to Windows PowerShell scripting. Windows PowerShell scripts are text files that have an extension of .ps1. You can create and edit a Windows PowerShell script by using any text editor, including Notepad.

Before you can begin writing and testing Windows PowerShell scripts, you might be required to adjust the Windows PowerShell script execution policy on your developer workstation. The reason for this step is that Windows PowerShell is configured out of the box to prohibit or to prompt the user during script execution.

You should take note that the installation of SharePoint 2013 actually changes the Windows PowerShell execution policy of the local machine. By default, the Windows PowerShell execution policy is set to restricted, which means that scripts have to be digitally signed before they can be run. However, the installation of SharePoint 2013 lowers the execution policy from restricted to unrestricted, which allows scripts to run even when they are not digitally signed.

On a developer workstation, it's common to disable the default execution constraints so that you can write and test scripts without security errors. You make this adjustment by calling the standard Windows PowerShell cmdlet named *Set-ExecutionPolicy* from the Windows PowerShell console to set the current machine's execution policy to *"bypass"*:

```
Set-ExecutionPolicy "bypass"
```

After you've correctly adjusted the Windows PowerShell execution policy, you can write your first script. Open Notepad and type in the following one-line script:

```
Write-Host "Hello World"
```

Now you need to save the file for the script with a .ps1 extension. First, create a new directory named Scripts on your local drive C. Next, save your new Windows PowerShell script file as c:\Scripts\Script1.ps1. Now that you've saved the Windows PowerShell script file with a .ps1 extension, you can execute the script to test your work.

Let's first execute the script through the Windows PowerShell console window. In the console window, move to the new directory by executing *Set-Location c:\Scripts*. Now you can execute the script by typing **.\Script1.ps1** and pressing Enter. When you do this, the message *Hello World* should appear in the Windows PowerShell console window.

Now let's create a Windows batch file so that you can execute the script without having to use the Windows PowerShell console window. Just create a new text file named RunIt.bat in the same directory as Script1.ps1, and call powershell.exe and pass the *-Command* parameter with the following syntax to execute the script:

```
powershell.exe -Command "& {.\Script1.ps1}"
pause
```

Notice that this example batch file also added a *pause* operation at the end. This can be handy because it keeps the Windows PowerShell console window open so that you can view the output of your Windows PowerShell script.

Finally, you should learn how to directly execute a Windows PowerShell script without any assistance from an MS-DOS batch file. If you right-click a Windows PowerShell script such as Script1.ps1 in Windows Explorer, you'll find a Run With PowerShell menu command. If you execute this command, the Windows operating system takes care of executing the Windows PowerShell script for you.

Executing Windows PowerShell scripts by using the Run With PowerShell command is quick and easy, but it doesn't leave the Windows PowerShell console window open when it's done. If you like using this technique but you still want to display the Windows PowerShell console window afterward, you can simply add the *Read-Host* cmdlet at the bottom of your script, which results in the Windows PowerShell console window remaining open until you press the Enter key:

```
Write-Host "Hello World"
Read-Host
```

The Windows PowerShell Integrated Scripting Environment (ISE)

Although you can use any text editor to write Windows PowerShell scripts, you'll probably prefer to use a powerful new utility, the *Windows PowerShell Integrated Scripting Environment (ISE)*, which is included with the Windows Server operating system.

The Windows PowerShell ISE will be immediately familiar to anyone with experience in Visual Studio. You can type a script in the top window and then press the F5 key to execute the script in debug mode. The Windows PowerShell ISE allows you to debug by setting breakpoints and to single-step through your code. After you've launched the Windows PowerShell ISE, type the following script into the top window and then press F5:

```
$sum1 = 2 + 2
$sum2 = 3 + 4
$sum3 = $sum1 + $sum2
Write-Host $sum3
```

This example shows how to create a new variable in a Windows PowerShell script. You simply create a new variable name, which begins with the $ character. You don't need to define variables before you use them, as you do in C#. Instead, you just create a variable when you begin using it.

Now, let's write a Windows PowerShell control-of-flow construct. In this case, we create a new string array by using the proper Windows PowerShell syntax, and then write a *foreach* loop to enumerate each string:

```
$band = "Paul", "John", "George", "Ringo"

foreach($member in $band) {
  Write-Host $member
}
```

One aspect of Windows PowerShell that will instantly appeal to .NET developers is that you can create and program against any .NET object. For example, imagine you want to create an object from the *DateTime* class of the .NET Framework. You do this by executing the *New-Object* cmdlet and passing the class name and initialization values as parameters:

```
$date = New-Object -TypeName System.DateTime -ArgumentList @(1882,7,4,0,0,0)
$message = "Wingtip Toys, Inc. was founded on " + $date.ToLongDateString()
Write-Host $message
```

The preceding script produces the following output:

```
Wingtip Toys, Inc. was founded on Tuesday, July 04, 1882
```

In addition to creating new .NET objects, Windows PowerShell allows you to call the static methods and static properties of classes in the .NET Framework. You do this by typing the namespace-qualified class name in square brackets, like this: *[System.DateTime]*. After you type the class name, you add the *::* operator (two colons) and then the call to a static member:

```
$today = [System.DateTime]::Today
Write-Host $today.ToLongDateString()
Write-Host $today.ToString("MM/dd/yy")
Write-Host $today.AddDays(100).ToString("MMMM d")
```

If you're feeling nostalgic, you can even use Windows PowerShell to create and program against COM objects. For example, let's say you want to write a Windows PowerShell script that launches Internet Explorer and navigates to a specific URL. The Windows operating system provides a built-in COM interface that allows you to launch and control Internet Explorer:

```
$ie = New-Object -ComObject "InternetExplorer.Application"
$ie.Navigate("http://intranet.wingtiptoys.com")
$ie.Visible = $true
```

Windows PowerShell snap-ins for SharePoint

Windows PowerShell installs a set of core libraries containing cmdlets such as *Write-Host, Get-Process*, and *Where-Object*. Environments such as SharePoint Foundation add their own library of custom cmdlets by installing and registering a special type of an assembly DLL known as a *Windows PowerShell snap-in*. When you install SharePoint 2013, a Windows PowerShell snap-in named *Microsoft.SharePoint.PowerShell* is installed. However, this snap-in doesn't automatically load into every Windows PowerShell session. Instead, you have to ensure that the *Microsoft.SharePoint.PowerShell* snap-in is loaded before you begin to call the cmdlets specific to SharePoint.

SharePoint Foundation provides a specialized version of the Windows PowerShell console known as the *SharePoint 2013 Management Shell*. The main difference between the standard Windows PowerShell console window and the SharePoint 2013 Management Shell console has to do with which Windows PowerShell providers get loaded automatically. More specifically, the SharePoint 2013 Management Shell automatically loads the *Microsoft.SharePoint.PowerShell* snap-in, whereas the standard Windows PowerShell console does not. In general, you can't always rely on the SharePoint

snap-in *Microsoft.SharePoint.PowerShell* being loaded automatically, so you need to learn how to load it explicitly within a Windows PowerShell script.

Let's say you've just launched the standard Windows PowerShell console window and you attempt to execute one of the cmdlets built into SharePoint Foundation, such as *Get-SPWebApplication*. The call to this cmdlet will fail unless you've already loaded the *Microsoft.SharePoint.PowerShell* Windows PowerShell snap-in. Before calling the *Get-SPWebApplication* cmdlet, you need to load the SharePoint Management Windows PowerShell snap-ins for SharePoint by using the *Add-PSSnapin* cmdlet:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell
Get-SPWebApplication
```

Executing these two cmdlets in sequence displays the current collection of web applications for the current farm, excluding the web application for SharePoint 2010 Central Administration:

```
DisplayName                    Url
-----------                    ---
Wingtip Intranet               http://intranet.wingtiptoys.com/
Wingtip Extranet               http://extranet.wingtiptoys.com/
Wingtip Public Web site        http://www.wingtiptoys.com/
```

Now let's write a Windows PowerShell script to create a new web application. You can do this by calling the *New-SPWebApplication* cmdlet. The call requires quite a few parameters. Note that the following script creates a "classic mode" web application, which is no longer supported through the Central Administration interface in SharePoint 2013:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"

$name = "Wingtip Intranet Web App"
$port = 80
$hostHeader = "intranet.wingtiptoys.com"
$url = "http://intranet.wingtiptoys.com"
$appPoolName = "SharePoint Default Appl Pool"
$appPoolAccount = Get-SPManagedAccount "WINGTIP\SP_Content"

New-SPWebApplication -Name $name -Port $port -HostHeader $hostHeader -URL $url '
                     -ApplicationPool $appPoolName '
                     -ApplicationPoolAccount $appPoolAccount
```

Notice that the call to the *New-SPWebApplication* cmdlet in the preceding script breaks across multiple lines for clarity. When you write scripts, however, you must place the entire call to a cmdlet and all its parameters on a single line. That is, of course, unless you know the special trick of using the grave accent (`) to add line breaks within a call to a cmdlet inside a Windows PowerShell script, as shown in the preceding example.

As you can imagine, writing and executing scripts like this can save you quite a bit of time in a production farm because the need to perform the same tasks manually through SharePoint 2013 Central Administration is eliminated. Scripts like this also provide a great way to create consistency in how you create web applications across farms.

We'll finish with one more example. Let's write a script to create a new site collection in the web application created earlier, which has a team site as its top-level site. You can accomplish this by calling the *New-SPSite* cmdlet:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell

$title= "Wingtip Intranet"
$url = "http://intranet.wingtiptoys.com"
$owner = "WINGTIP\Administrator"
$template = "STS#0"

New-SPSite -URL $url -Name $title -OwnerAlias $owner -Template $template
```

When you create a new site collection by using the *New-SPSite* cmdlet, you must specify the URL and title and provide a user account to be configured as the site collection administrator. You can also specify a template by using the *Template* parameter, which is applied on the top-level site. In this example, a template of *STS#0* has been applied to create the top-level site as a standard team site.

Now you've written a script to create a new site collection. The first time you run it, it works great. But what happens when you run it a second time? The second attempt to call the *New-SPSite* cmdlet fails because a site collection already exists at the target URL.

During development, there's a common scenario in which you must continually delete and re-create a site to effectively test and debug your code. Before deleting a site collection, your script should check to determine whether a target site collection already exists at the target URL by using the *Get-SPSite* cmdlet. If the site collection already exists, you can delete it with the *Remove-SPSite* cmdlet:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell

$title= "Wingtip Intranet"
$url = "http://intranet.wingtiptoys.com"
$owner = "WINGTIP\Administrator"
$template = "STS#1"

# delete target site collection if it exists
$targetSite = Get-SPSite | Where-Object {$_.Url -eq $url}
if ($targetSite -ne $null) {
  Remove-SPSite -Identity targetSite -Confirm:$false
}

# create new site collection
New-SPSite -URL $url -Name $title -OwnerAlias $owner -Template $template
```

Remember that cmdlets such as *New-SPSite* return objects that you can program against. For example, imagine you want to update the title of the top-level site after the site collection has been created. A site collection object exposes a *RootWeb* property that allows you to access the top-level site. The site object provides a *Title* property that you can modify with a new title. You must call the site object's *Update* method to write your changes back to the content database:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell

$title= "Wingtip Dev Site"
$url = "http://intranet.wingtiptoys.com"
$owner = "WINGTIP\Administrator"
$template = "STS#0"

# delete target site collection if it exists
$targetSite = Get-SPSite | Where-Object {$_.Url -eq $url}
if ($targetSite -ne $null) {
  Remove-SPSite -Identity targetSite -Confirm:$false
}

$sc = New-SPSite -URL $url -Name $title -OwnerAlias $owner -Template $template
$site = $sc.RootWeb
$site.Title = "My New Site Title"
$site.Update
```

You've just seen an example of writing code against the server-side object model of SharePoint Foundation. Unfortunately, the Windows PowerShell ISE isn't able to provide IntelliSense in the same manner that Visual Studio does. However, the Windows PowerShell ISE still has valuable editing and debugging features that are easy to learn and use. You should become familiar with this tool because it provides a quick way to script out changes to the local farm in your development workstation or in a production environment.

Summary

SharePoint 2013 mainly consists of two products: SharePoint Foundation and SharePoint Server 2013. Having a solid understanding of SharePoint Foundation is essential even for developers who are only building software for SharePoint Server 2013. That's because SharePoint Foundation provides the underlying infrastructure on which SharePoint Server 2013 is built.

SharePoint Foundation represents different things to different people. To users, SharePoint Foundation provides the infrastructure for web-based business solutions that scale from simple team-collaboration sites to enterprise-level applications. To site collection administrators, SharePoint Foundation provides the capability to customize sites by adding lists and document libraries and by customizing many aspects of a site's appearance through the browser or by using a customization tool such as SharePoint Designer 2013.

To a company's IT staff, SharePoint Foundation provides a scalable and cost-effective solution for provisioning and managing a large number of sites in a web farm environment. It also provides a reliable mechanism to roll out applications and to version these applications over time.

To a developer, SharePoint Foundation represents a rich development platform that adds value on top of the underlying ASP.NET platform. Developers build software solutions targeting SharePoint Foundation by using features and components such as Web Parts, event handlers, and workflows. Now that you've studied the SharePoint developer roadmap and made it through our Windows PowerShell boot camp, you're ready to dive into the fundamentals of SharePoint 2013 development.

CHAPTER 2

SharePoint development practices and techniques

Before you can start building a custom Microsoft SharePoint solution you will have to make sure you set up your development environment correctly. Because the hardware requirements for SharePoint 2013 are again a lot more demanding than they were for SharePoint 2010, setting up a new development environment might well mean that you have to acquire new hardware. There might be quite a bit of time between the moment that you order the hardware, whether from an external vendor or from an internal department, and when you can actually start using the hardware. This means that it's important to start planning your SharePoint customizations early, so that waiting on the hardware will not interfere with your project planning.

When you have gotten the hardware, you will have to install your development environment. It is important to do this meticulously, to follow best practices and to make sure you document the entire configuration. Documentation is important if you have to create a second environment, or if you have to recreate your development environment.

When your SharePoint environment has been set up properly, you will need proper specifications so that you can start designing your solution. You will have to decide what type of solution will best suit your skills, the environment into which the solution will have to be deployed, and the functionality that you have to create. SharePoint 2013 introduces a new development approach, which means that you can now not only create farm solutions and sandboxes solution, but you can also create SharePoint apps. SharePoint 2013 also introduces a third application programming interface (API) by making Representational State Transfer (REST) APIs available that allow you to use simple HTTP requests and responses to perform CRUD (create, read, update, delete) operations on SharePoint data.

All these additions give you more options, but they also require you to make more choices, and it is important to make deliberate and well-informed choices to make sure that you end up with the best solution that you could possibly build for your specific situation and scenario. This chapter talks you through a lot of the choices and can help you make the right decisions.

Setting up a developer environment

Whenever you are looking at building a custom solution for any platform, one of the things you will have to determine is what environment you will use to build your custom solution. This is no different when you want to create a custom solution for SharePoint. Determining the best way to set up your development environment has always been difficult for SharePoint, and SharePoint 2013 adds even more complexity to it, with extended hardware requirements and two new types of servers.

Let's start by looking at the different server roles that you can choose from.

- Domain controller
- Database server
- SharePoint server
  - Web server
  - Application server
- Office Web Apps server
- Windows Azure workflow server

Although it is possible to build a development environment by using a standalone server installation of SharePoint on a single server without a domain controller or separate computer that is running Microsoft SQL Server, for practical reasons you will at least need a domain controller, a database server, and a SharePoint server. For certain types of SharePoint apps you might not need a SharePoint development environment, because these apps can be hosted on a generic web server that doesn't have SharePoint installed on it. However, you should test what your app looks like in a SharePoint environment before you add the app to the production environment, so you should always use some sort of test or development environment.

If your development environment is installed in an existing domain, you don't have to build your own domain controller; you can simply use an existing one. If you are creating your own domain, you will have to create a domain controller as well. You can create a single server and use that as the domain controller, the database server, and the SharePoint server. Be aware, though, that some things don't work on a domain controller and some things have to be configured differently. It is important to keep this in mind while developing and testing custom solutions on your development server.

In SharePoint 2010, Microsoft Office Web Apps came in a separate installation that had to be installed on at least one of the SharePoint servers in the farm. After installation, they could be configured as service applications. In SharePoint 2013, this is no longer the case. Office Web Apps is now its own product. Office Web Apps has to be installed in its own separate farm, and it cannot be installed on a server that also has SharePoint installed on it, because Office Web Apps will completely take over the Internet Information Services (IIS) on the server. You can install Office Web Apps on one or more servers and connect the Office Web Apps farm to the SharePoint farm. Having Office Web Apps installed in its own farm on one or more servers means that it is now more scalable. The Office Web

Apps farm can be connected to one or more SharePoint farms. This means that one Office Web Apps farm can support the SharePoint servers of several developers.

With SharePoint 2010, you automatically got the SharePoint 2010 workflow host, which was based on Windows Workflow Foundation 3. Windows Workflow Foundation was a native part of SharePoint, but the way in which it was implemented meant that customers who were serious about using workflow in SharePoint almost always ran into issues with scalability. SharePoint 2013 uses a new workflow service, which is built on the Windows Workflow Foundation components of the Microsoft .NET Framework 4.5. The new workflow service is called Workflow Manager and, like Office Web Apps, is a separate installation that should be installed on separate servers. After you have created a Workflow Manager farm consisting of one or more servers, you can connect this farm to your SharePoint 2013 farm. As with Office Web Apps, creating a separate workflow farm means that your environment will be a lot easier to scale out and a lot more suitable for use in a serious workflow solution or a large enterprise. Your old SharePoint 2010 workflows will still work, because SharePoint 2013 automatically installs the SharePoint 2010 workflow engine.

To summarize, if you want to have all SharePoint 2013 functionality available to you in your development environment, you will need at least three servers:

- A domain controller/database server/SharePoint server
- An Office Web Apps server
- A Workflow Manager server

You can, of course, have many more: you could split out your domain controller, database server, and SharePoint server; you could have separate SharePoint web and application servers; and you can have as many Office Web Apps and Workflow Manager servers as you want. How many servers you use will mostly depend on the size of the solution that you are building, the type of functionality that you need, and—let's face it—your budget.

Deciding between virtual and physical

An important decision that you have to make when you start to think about your development environment is whether you will be using virtual or physical servers. You could choose to install a supported server operating system (we'll get into more detail on that soon) directly on your computer, either by connecting to an existing domain or turning the computer into a domain controller and installing SQL Server and SharePoint on it. You can no longer install SharePoint on a client operating system such as Windows 7 or Windows 8 as you could with SharePoint 2010. However, unlike Windows 7, Windows 8 does support Hyper-V, which means that you can create your virtual machines in Hyper-V on your Windows 8 computer. The Windows 8 version of Hyper-V is officially called Client Hyper-V.

As long as you only work on a single project, and you only need a single server (so you don't need Workflow Manager or Office Web Apps), you can run your development environment directly on your computer. However, creating your development environment by using virtual servers is a far more flexible solution. You can either host the virtual servers on your own computer or on a server somewhere in the network, or even in the cloud. With today's hardware requirements (especially the

memory) and considering the fact that you might need more than one server, running your development environment on your computer won't be a feasible solution for most people, so in a lot of cases development servers are hosted in a network somewhere. If you are using your development environment on a daily basis, it is recommended that you make sure that your servers are hosted somewhere relatively close to you to minimize latency issues and frustrations.

Running the development environment on a virtual server has a few advantages:

- Using virtual servers as a development environment means that you can use a different virtual server for each project you're working on. When a developer works on more than one project, it is better not to have the configuration and custom solutions from these projects in a single environment. Settings or solutions from one project might influence the behavior of the solutions from the second project, which means that you have no way of knowing what is causing problems and you can't determine how the solution will behave in the production environment.
- Another advantage of working with virtual servers is the fact that it's easy to create snapshots and to go back to them. By using snapshots, the developer can run tests and, depending on the outcome of a test, decide to go back to a snapshot of a previous situation. He can then make some changes to the solution and run the same tests again.

  Also, when project work goes on for a long time, environments sometimes get messy from testing different solutions and settings, and going back to a snapshot is a very easy way to clean that up. Using snapshots also means that you can go back to a previous state if a solution that you deployed or a script that you ran messed up your environment.
- Using virtual servers to create development environments also makes it easier to set up a new development environment when a new developer is added to the project. Later on in this chapter we will talk in more detail about having a team of developers work on a single project.
- In most cases, using virtual servers is also a lot cheaper than using physical servers. If you have a large physical server, you can run several virtual servers on it. This means you can save on hardware costs. Also, if you don't need all your servers at the same time, they can share the resources, and if you need a new server, you can very quickly set it up, instead of having to order hardware and wait till it arrives.

Understanding hardware and software requirements

As with every SharePoint version, SharePoint 2013 has its own hardware requirements. Table 2-1 shows an overview of the hardware requirements for SharePoint 2013. As you can see, the amount of memory needed to run a SharePoint Server development environment has again increased significantly. There are a couple of things to note:

- *Single server* means that both SharePoint and its databases are running on the same server.
- A single server development installation of SharePoint Server 2013 is listed as requiring 24 gigabytes (GB) of RAM. However, the amount of RAM it really needs heavily depends on what services you are running in the environment. For instance, if you are actively using search, you

probably need 24 GB, or at least something close to that. However, if you are only using web applications and some of the lighter service applications, you can get away with having a lot less memory.

- The storage on the system drive has to be at least 80 GB. It is very important to note that this does not include the storage that is needed to store the databases that contain the content from your SharePoint environment, and it doesn't include the storage that is needed to store, for instance, the SharePoint logs. Make sure that you have enough storage on your system; storage is cheap, and it's very annoying to have to go into your development server every day to try and free up some storage so that at least your server will keep running.

TABLE 2-1 Hardware requirements for SharePoint 2013

Type of installation	RAM	Processor	Storage on system drive
Single server development installation of SharePoint Foundation 2013	8 GB	64-bit, 4 cores	80 GB
Single server development installation of SharePoint Server 2013	24 GB	64-bit, 4 cores	80 GB
SharePoint server in a SharePoint Server 2013 development environment	12 GB	64-bit, 4 cores	80 GB
Database server in a SharePoint 2013 development environment	8 GB	64-bit, 4 cores	80 GB

SharePoint 2013 also comes with its own software requirements. For a SharePoint 2013 server, the following software is required:

- The 64-bit edition of Windows Server 2008 R2 Service Pack 1 (SP1) Standard, Enterprise, or Datacenter or the 64-bit edition of Windows Server 2012 Standard or Datacenter
- Hotfix: The SharePoint parsing process crashes in Windows Server 2008 R2 (KB 2554876)
- Hotfix: FIX: IIS 7.5 configurations are not updated when you use the ServerManager class to commit configuration changes (KB 2708075)
- Hotfix: WCF: process may crash with "System.Net.Sockets.SocketException: An invalid argument was supplied" when under high load (KB 2726478)
- The prerequisites installed by the Microsoft SharePoint Products Preparation Tool
- Hotfix: ASP.NET (SharePoint) race condition in .NET 4.5 RTM:
  - Windows Server 2008 R2 SP1 (KB 2759112)
  - Windows Server 2012 (KB 2765317)

For a database server in a SharePoint 2013 farm, the following software is required:

- The 64-bit edition of Microsoft SQL Server 2012, or the 64-bit edition of SQL Server 2008 R2 Service Pack 1
- The 64-bit edition of Windows Server 2008 R2 Service Pack 1 (SP1) Standard, Enterprise, or Datacenter or the 64-bit edition of Windows Server 2012 Standard or Datacenter
- Hotfix: The SharePoint parsing process crashes in Windows Server 2008 R2 (KB 2554876)
- Hotfix: FIX: IIS 7.5 configurations are not updated when you use the ServerManager class to commit configuration changes (KB 2708075)
- Hotfix: ASP.NET (SharePoint) race condition in .NET 4.5 RTM:
  - Windows Server 2008 R2 SP1 (KB 2759112)
  - Windows Server 2012 (KB 2765317)
- .NET Framework version 4.5

When setting up your development environment, you should always aim to make sure that it's as much like the production environment as possible.

Delivering high-quality solutions

To deliver high-quality solutions it is best for the development environment to be as much like the production environment as possible. Theoretically this is true for all aspects of the environment: hardware, software, configuration, and data. In most cases, however, the hardware of a development environment cannot be the same as the hardware of a production environment. This is fine, as long as you are aware of the differences and what the impact of them might be on your test results.

So that accurate tests can be performed in a development environment, the software should be the same as the software in the production environment. You should use the same version of Windows Server and SharePoint and a similar version of SQL Server. If the production environment has SharePoint Server installed, make sure the development environment doesn't have SharePoint Foundation installed. If the production server has a Windows service pack installed on it, make sure you install the same service pack in the development environment. It also works the other way around; if the service pack will not be installed in the production environment, do no install it in the development environment either. If one of the environments gets a SharePoint service pack or cumulative update installed on it, make sure all environments get that same service pack or cumulative update installed on them.

The way in which you configure your development server should also be as much like the production environment as possible. The best thing is to try and get access to the build guide for the production environment and use that to set up your development environment.

Examples of settings that are important when configuring your development environment are:

- Using the default SQL instance or different instances
- The authentication type:
  - NT LAN Manager (NTLM)
  - Kerberos
  - Windows claims
  - Security Assertion Markup Language (SAML) claims
- Using host headers on your web application, or Host Header Site Collections
- HTTP or HTTPS
- The number of web applications
- The way in which the farm, application pool, and services accounts are configured and the level of permissions they have. Make sure you have the same number of managed accounts in your development environment as in the production environment.

In order to get accurate test results, it is also very helpful to have representative sample data and test users. The data will help you perform the same type of actions that a user would. If you are able to load enough sample data into your development environment, it will also help you test the scalability of your solution, at least to a certain extent. Most custom solutions perform very well with only a couple of documents, users, or sites, but when there are tens of thousands it might be a completely different story. Even if you can't test on the scale of your production environment, you should always keep in mind what numbers your solution will have to cope with after it's in production. It is always a good idea to at least make sure that you test whether your application will keep working past the list view threshold. The *list view threshold* is a web application setting that can be adjusted in Central Administration that tells SharePoint how many items can be requested from the database in a single query. The default list view threshold is 5,000.

As a developer, you will usually log in with an account that has administrative permissions. It's the only way in which you can properly develop custom full trust solutions. Do make sure that you are not logged on as the SharePoint farm account. When you are testing your solution, it is very important to not only test it using your administrative account, but also with accounts that have Read, Contribute, and Site Owner permissions. A lot of custom SharePoint solutions will work fine when run by an administrator, but need more work when a reader or contributor of a site should be able to work with them as well. For instance, the List view threshold (the throttling feature that specifies the maximum number of list or library items that a database operation, such as a query, can process at the same time) will not be applied if you are logging onto your SharePoint environment as a local administrator, which means that you cannot test the behavior of large lists properly.

Automating SharePoint administration by using Windows PowerShell scripts

Windows PowerShell scripts can be used to automate SharePoint installation and management. When you are using Windows PowerShell scripts to install SharePoint, it is easy to repeat the installation in exactly the same way. This is very useful when you have to create multiple development environments or multiple servers in a production environment, or when a farm has to be rebuilt after a system failure. Be aware that not all steps of the installation can be scripted by using Windows PowerShell, so you will still have to make sure that all steps are documented as well.

Using Windows PowerShell to manage SharePoint is very useful for repeatable tasks. When you use a saved script every time, the chances of human errors causing serious problems during maintenance decrease. Windows PowerShell can also be used to fully automate maintenance steps. It would, for example, be possible to create a Windows PowerShell script that creates a new site collection. The next step would be to add a couple of parameters and then automatically start the script. You could, for instance, start the script whenever a new project or customer is added to a Customer Relationship Management (CRM) system.

Though it is often convenient to use Windows PowerShell to install or configure SharePoint, in some cases you don't have a choice because some functionality doesn't show up in the user interface and can only be configured by using Windows PowerShell. Examples of this are the multitenancy features. The multitenancy features are a set of features that allow SharePoint to work as a hosting platform. They allow for operational service management of SharePoint for one or more divisions, organizations, or companies. Using the multitenancy features allows SharePoint to separate data, features, administration, customizations, and operations. In order to set up multitenancy in an environment, you have to set up site subscriptions, partitioned service applications, tenant administration sites, and (optionally) host header site collections and feature packs. All these features can only be configured by using Windows PowerShell.

If you need to install development environments on a regular basis—for instance, because you are working on different projects, or because you are working on a long-running project and developers are coming and going—it is worthwhile to create a Windows PowerShell script to install a development environment. Even if you would just use Windows PowerShell to configure SharePoint, this will save you a lot of time. It will also make sure that your development environments are always configured in exactly the same way. In addition to the fact that doing a scripted installation is often faster, this approach also allows you to do work on something else while the script is running.

There are two different tools in which you can write and run Windows PowerShell scripts: the Windows PowerShell console window or the Windows PowerShell Integrated Scripting Environment (ISE). To make the Windows PowerShell ISE available on your server, you need to install the Windows PowerShell Integrated Scripting Environment (ISE) Windows feature. You can do this by opening up the Server Manager, clicking Add Features, and selecting the ISE feature.

If you are using the console environment, you can either use the general console environment or the SharePoint 2013 Management Shell. You can access the general console environment by selecting

it from the Windows Start menu. If it's not on the first page, you can get to it from the Windows Start menu by simply starting to type **PowerShell**. This will give you the option to select one of the available Windows PowerShell tools and consoles.

Don't pick the 32-bit version (x86); SharePoint is a 64-bit product. You can also go to the Windows Start menu to select the SharePoint 2013 Management Shell. You can find this in the same way as the general console; go to the Start menu, start typing **PowerShell**, and select the SharePoint 2013 Management Console.

These are effectively the same environment, except for the fact that in the SharePoint Management Shell the Microsoft.SharePoint.PowerShell snap-in has already been loaded. If you are using the standard console, you will have to load the snap-in yourself, by using the following command:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
```

You add the *-ErrorAction "SilentlyContinue"* mainly because the console will throw an error if the snap-in has already been loaded. You can ignore this, so it will look nicer if you hide the error. You can also play it safe and check to see whether the snap-in is already loaded before attempting to load it, by using the following:

```
$snap = Get-PSSnapin | Where-Object {$_.Name -eq 'Microsoft.SharePoint.PowerShell'}
if ($snap -eq $null) {
  Add-PSSnapin Microsoft.SharePoint.PowerShell
}
```

If you are creating a larger script, it is probably easier to open up the Windows PowerShell ISE, because this provides a better editing environment. You will have to load the Microsoft.SharePoint.PowerShell snap-in into the ISE as well. You can do this by using exactly the same script used for the console. If you find yourself using the ISE a lot, you can also add the snap-in automatically when the ISE starts, by adding it to the Windows PowerShell profile. The profile is a Windows PowerShell script file that runs every time you start Windows PowerShell. It has a .ps1 extension like any normal Windows PowerShell file, and you can put any valid Windows PowerShell cmdlet in it. The only way in which the profile file differs from a normal script file is in its name and location.

If you want to use the profile, you will first have to figure out whether a profile already exists on the server. You can do this by using the *Test-Path* cmdlet:

```
Test-Path $profile
```

If the profile already exists, the *Test-Path* cmdlet will return *True*; if it doesn't exist, it will return *False*. You can also just run the *$profile* cmdlet and use Windows Explorer to browse to the path that it returns. If the file isn't there, the profile doesn't exist:

```
$profile
```

You can use the *New-Item* cmdlet to create a profile if one doesn't already exist. *-path $profile* passes in the full path, and *-type file* tells the cmdlet that you are trying to create a file:

```
New-Item -path $profile -type file
```

When you open the profile, you will notice that it is completely empty. You can add to it any script that you want to always be executed before you start working on your scripts. This could, for instance, be a command telling Windows PowerShell to always go to a default location:

```
Set-Location C:\scripts
```

Or you can add the Microsoft.SharePoint.PowerShell snap-in:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
```

Using Windows PowerShell to deploy a custom solution

As a developer, you may also find Windows PowerShell very useful for creating a deployment script that can be used to install your custom solution in a SharePoint environment. This will allow you to distribute your custom solution across test, user acceptance, and production environments a lot more easily. It will mean that administrators don't have to perform a lot of manual steps and that you as a developer don't have to describe all these steps in a deployment manual.

When using Windows PowerShell to write a deployment script, you will have to take into account that a previous version of your solution might already be installed in the environment that you are deploying to. This means that you first have to retract and remove the solution before you can install and deploy the solution. Retracting a solution forces SharePoint to delete most of the files it copied during deployment as well as to uninstall features and delete assemblies from the global assembly cache. After you've retracted a solution, you can then remove it, which deletes the solution package file from the configuration database.

One thing to be aware of is that SharePoint doesn't clean up after itself very well when you retract a solution. For instance, SharePoint doesn't explicitly deactivate features before it retracts the solution that they are deployed in. Because of this, it is a best practice to make sure that you deactivate all features in a solution before retracting the solution. Another thing to keep in mind is that SharePoint doesn't always delete all files that were deployed using a solution when the solution is retracted. When it doesn't, this often is for a good reason (for instance, because it could cause errors if the files were to be deleted), but it is something to keep in mind because it can cause quite a bit of cluttering, especially in a development or test environment where solutions are installed and retracted all the time.

You can retract a solution package by using the *Uninstall-SPSolution* cmdlet. When calling *Uninstall-SPSolution*, you should pass the *-Identity* parameter and the *-Local* parameter in the same manner as when calling *Install-SPSolution*. You should also pass the *-Confirm* parameter with a value of *$false* because failing to do so will cause the cmdlet to prompt the user, which can cause problems if the script is not monitored while it runs. After you've retracted the solution, you can then remove it by calling *Remove-SPSolution*, which instructs SharePoint Foundation to delete the solution package file from the configuration database:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
$SolutionPackageName = "WingtipDevProject1.wsp"
Uninstall-SPSolution -Identity $SolutionPackageName -Local -Confirm:$false
Remove-SPSolution -Identity $SolutionPackageName -Confirm:$false
```

These calls to *Uninstall-SPSolution* and *Remove-SPSolution* will fail if the solution package isn't currently installed and deployed. Therefore, it makes sense to add a call to *Get-SPSolution* and conditional logic to determine whether the solution package is currently installed and deployed before attempting to retract or remove it:

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
$SolutionPackageName = "WingtipDevProject1.wsp"
$solution = Get-SPSolution | where-object {$_.Name -eq $SolutionPackageName}
# check to see if solution package has been installed
if ($solution -ne $null) {
  # check to see if solution package is currently deployed
  if($solution.Deployed -eq $true){
    Uninstall-SPSolution -Identity $SolutionPackageName -Local -Confirm:$false
    Remove-SPSolution -Identity $SolutionPackageName -Confirm:$false
  }
}
```

Now that you've made sure that there's no old version of your solution installed on the farm, you can add and deploy your solution. Listing 2-1 shows the complete Windows PowerShell script. This script can be used to deploy a solution that cannot be scoped to a web application. Also, the DLL file in this solution will be deployed to the global assembly cache. If a solution can be scoped to a web application, the *–AllWebApplications* parameter can be used to deploy the solution to all web applications, or the *–WebApplication* parameter can be used to specify a specific web application that the solution should be deployed to.

LISTING 2-1 A Windows PowerShell script to uninstall and install a solution

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
$solution = Get-SPSolution | where-object {$_.Name -eq $SolutionPackageName}
if ($solution -ne $null) {
  if($solution.Deployed -eq $true){
    Uninstall-SPSolution -Identity $SolutionPackageName -Local -Confirm:$false
  }
  Remove-SPSolution -Identity $SolutionPackageName -Confirm:$false
}
Add-SPSolution -LiteralPath $SolutionPackagePath
Install-SPSolution -Identity $SolutionPackageName -Local -GACDeployment
```

Configuring SharePoint service applications

In SharePoint Server 2010 the concept of service applications was introduced. SharePoint contains several different service applications, and all of them provide a specific piece of functionality to your SharePoint farm if they are enabled. All service applications can be shared across web applications, and some service applications can even be shared across farms. Let's establish the terminology first.

A *service application* itself is the logical container of the service. We use the term *service application* to describe the services architecture in SharePoint. It is also what is exposed in the Central Administration site through the Manage Service Applications page. For most service applications, there can be more than one instance of the service application in a single farm.

The *service instance* is the actual implementation of the service, the binaries. A service instance could include Windows Services, configuration, registry settings, timer jobs, and more. The bits that make up the service instance are deployed to every SharePoint server in the farm.

The *service machine instance* of a particular service application is the server or servers in the farm on which the service for that service application runs. You can check where a service is running and start or stop a service on a particular server by going to the Services On Server page in the Central Administration site. On this page, you can select a server and then start the services you want to run on that particular server. When a service runs on more than one server in the farm, software round-robin load balancing is provided by SharePoint. Not all service applications have an associated service machine instance. Most service applications can have more than one associated service machine instance, but some can only have one. Not all services you see on the Services On Server page are service machine instances of a service application.

The *service application endpoint* is created when you start a service. Starting the service and thus creating a service machine instance creates an Internet Information Services (IIS) virtual application in the SharePoint Web Services IIS website. The virtual application includes a Windows Communication Foundation (WCF) or .asmx web service. This web service is the service application endpoint. Each service application must have its own service application endpoint.

A *service application proxy* (also called *service connection* or *service association*) is a virtual link between a web application and a service application. The service application proxy also enables cross-farm services.

A *proxy group* is a group of service application proxies that are selected for one or more web applications. By default, all service application proxies are included in the default proxy group. When you create a web application, you can do one of the following:

- Select the default proxy group.
- Create a custom proxy group by selecting which service application proxies you want to link to the web application. These service application proxies will then be included in the proxy group.

The custom proxy group for one web application cannot be reused with a different web application.

There are three ways in which you can configure service applications:

- By selecting services when you run the SharePoint Products Configuration Wizard
- By adding services one by one on the Manage Service Applications page in SharePoint 2013 Central Administration
- By using Windows PowerShell

It is not recommended that you use the SharePoint Product Configuration Wizard to configure service applications. Using the wizard will create the service applications with a set of default settings that might not be suitable for your environment. If you use the wizard it is also very easy (as easy as selecting a check box) to create too many service applications. You should always just create the service applications that you need in your farm. Every service application consumes a certain amount of resources, so creating a service application that you don't need means that you are burning valuable resources on your server.

To create a service application from the Manage Service Applications page in Central Administration, you start by clicking the New button on the Manage Service Applications page, as shown in Figure 2-1. Click Managed Metadata Service to create a managed metadata service application.

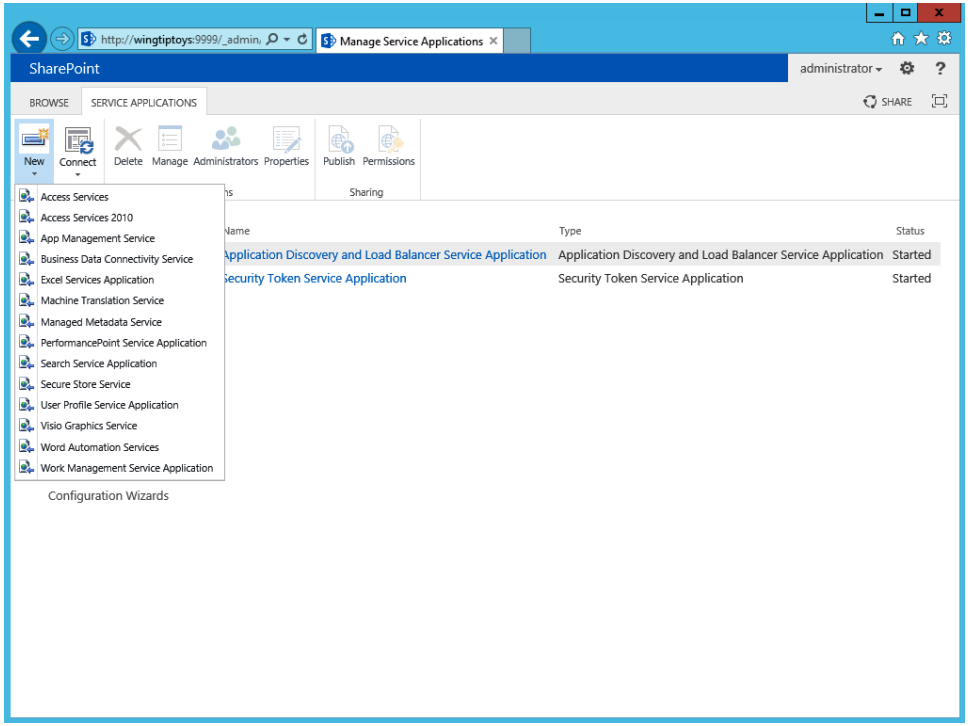


FIGURE 2-1 The Manage Service Applications page in Central Administration

On the next page, you enter a name for the service application and for the database that will store the contents and configuration of the managed metadata service application that you are creating, as shown in Figure 2-2. In this example, the name of the service application is Managed Metadata Service Application. The name of the database is ManagedMetadata. If this is the first service application that you are creating, you will also have to create an application pool that it can use. The name of the application pool in this example is SharePoint Web Services Default. This is the same name that the wizard would have used for the application pool that it creates if you use it to create service applications. In most cases, this application pool can be used for most if not all of your service applications. The account that is used in this example is the WINGTIPTOYS\spservices account. Be aware that the account that you use as the application pool account must be a managed account. Go to the Configure Managed Accounts page in Central Administration to create a new managed account. All managed accounts should be dedicated service accounts. Selecting Add This Service Application To The Farm's Default List means that SharePoint will add the service application to the default proxy group after you click OK.

Figure 2-3 shows the Manage Service Applications page after the managed metadata service application has been created. Creating the managed metadata service application through the Manage Service Applications page also automatically creates the managed metadata service application proxy. Most service applications automatically create their proxy when they are created through the Central Administration user interface. When Windows PowerShell is used to create the service application, you will almost always have to create the service application proxy yourself.

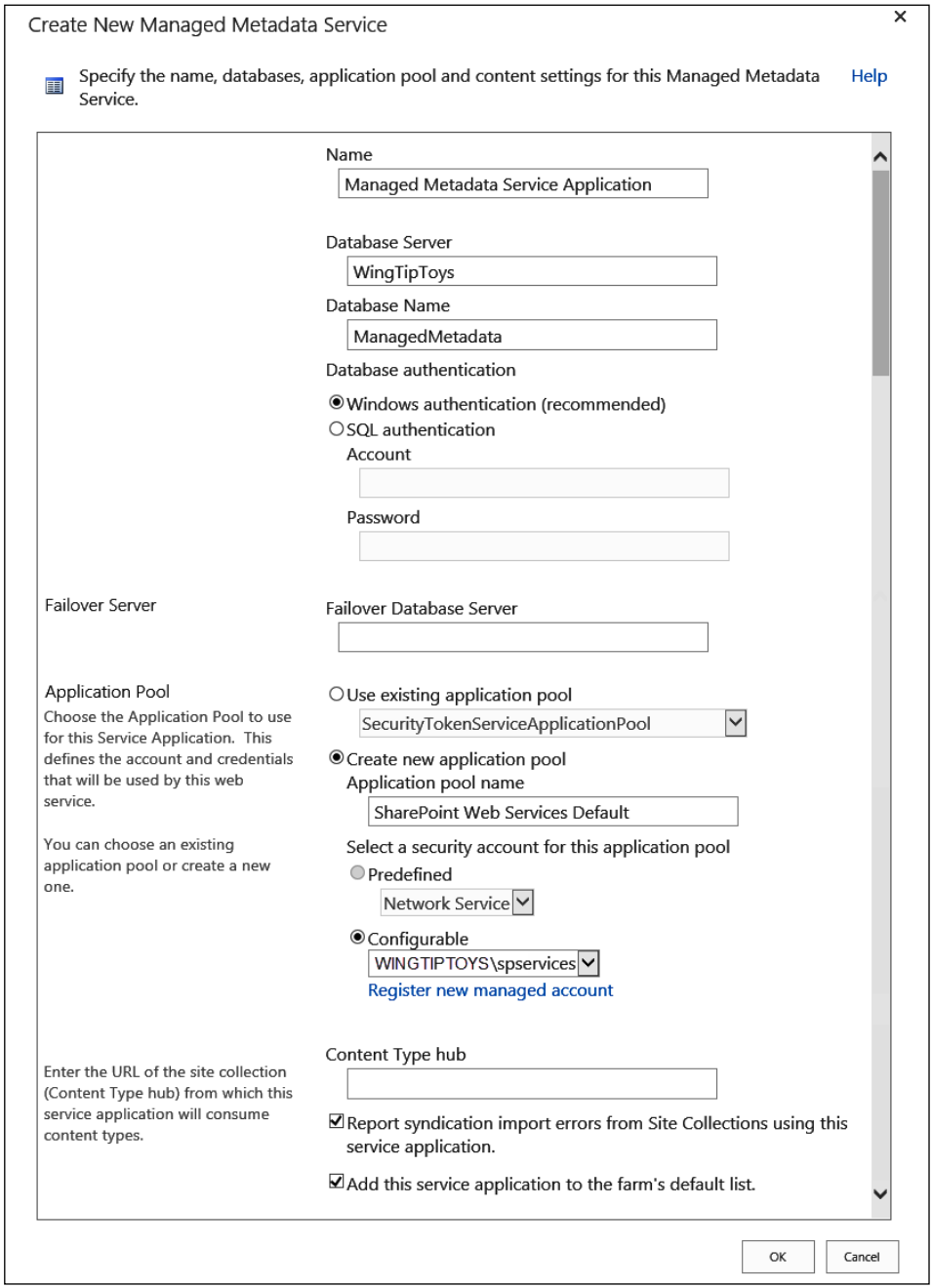


FIGURE 2-2 Creating a Managed Metadata Service Application

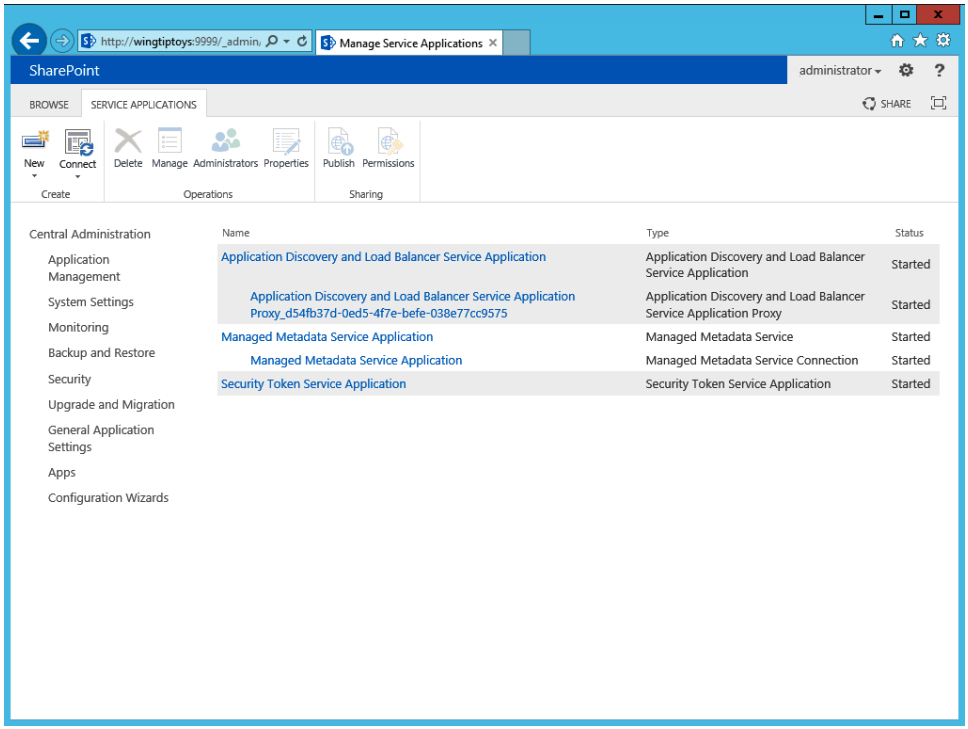


FIGURE 2-3 The Manage Service Applications page with the Managed Metadata Service Application and proxy

Some service applications start their service or services automatically, but for most service applications you will have to go into the Manage Services On Server page in Central Administration (shown in Figure 2-4). For the managed metadata service application, you will have to start the Managed Metadata Web Service on at least one server in the farm. In most development environments you will only have one SharePoint server, so you can start the service only on that server. Starting the service will also create a new IIS virtual application in the SharePoint Web Services IIS website. The name of the virtual application is a GUID, and the application will include the MetadataWebService.svc web service.

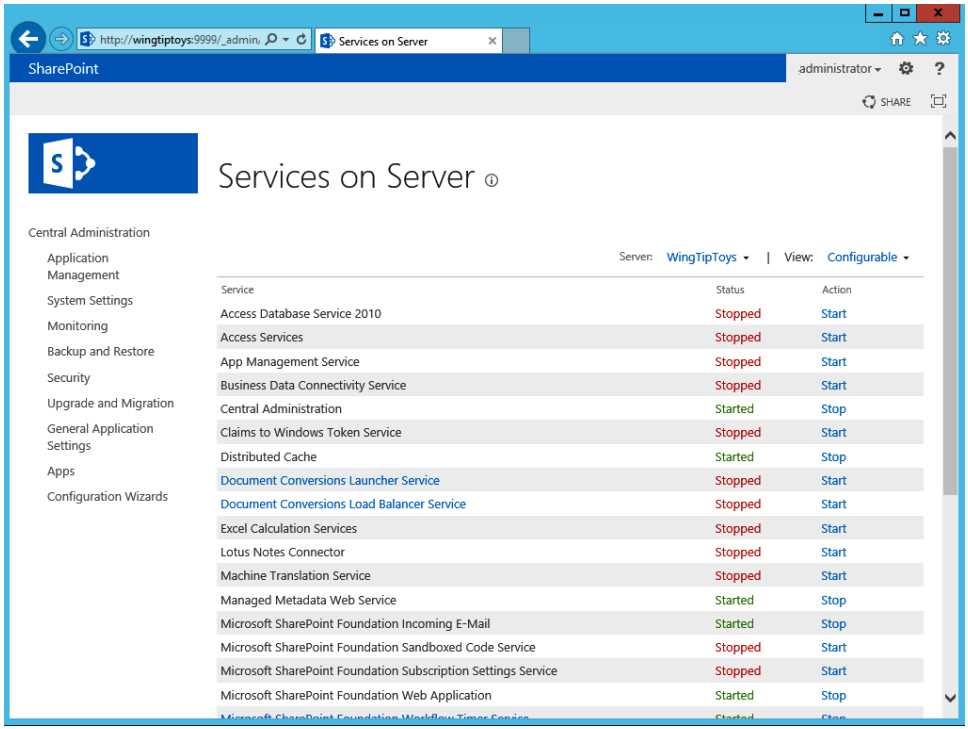


FIGURE 2-4 The Manage Services On Server page

Listing 2-2 shows the Windows PowerShell script that will create the service application, the application pool, and the service application proxy and that will start the managed metadata web service.

LISTING 2-2 A Windows PowerShell script to configure the Managed Metadata Service Application

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"

$saAppPoolName = "SharePoint Web Services Default"
$appPoolUserName = "WINGTIPTOYS\spservices"

# Gets Application Pool, or creates one
$saAppPool = Get-SPServiceApplicationPool -Identity $saAppPoolName -EA 0
if($saAppPool -eq $null)
{

  Write-Host "Creating Application Pool"
  # Create Application Pool
  $saAppPoolAccount = Get-SPManagedAccount -Identity $appPoolUserName
  $saAppPool = New-SPServiceApplicationPool -Name $saAppPoolName ´
    -Account $saAppPoolAccount
}

$mmsInstanceName = "MetadataWebServiceInstance"
$mmsName = "Managed Metadata Service Application"
$mmsDBName = "ManagedMetadata"

Write-Host "Creating Managed Metadata Service Application & proxy"
$mms = New-SPMetadataServiceApplication -Name $mmsName ´
    -ApplicationPool $saAppPoolName -DatabaseName $mmsDBName
$proxy = New-SPMetadataServiceApplicationProxy -Name "$mmsName Proxy "´
    -ServiceApplication $mms -DefaultProxyGroup
Write-Host "Starting Managed Metadata Web Service"
Get-SPServiceInstance | where {$_.GetType().Name ´
    -eq $mmsInstanceName} | Start-SPServiceInstance
Write-Host "Managed Metadata Service Application successful configured!"
```

Using debugging tools

While creating a custom solution, you can and probably will use Microsoft Visual Studio to debug your code in your development environment if you experience any issues or unexpected behavior. You might even use Visual Studio to do some debugging to simply get a better understanding of what's happening behind the scenes in SharePoint. The Visual Studio Debugger is not the only way to understand and troubleshoot SharePoint and your custom components, though. Other tools that you can use include:

- Unified Logging Service (ULS) and Windows event logs
- The Developer Dashboard
- Fiddler and other network monitoring tools

Working with ULS and Windows event logs

The ULS logs are SharePoint's own dedicated log files. Whenever there is problem with a SharePoint environment, the first place a SharePoint developer or administrator should look for information is in the ULS logs. One of the advantages of the ULS logs is that they can be used for troubleshooting in all types of environments. Regardless of whether problems are occurring in a development environment, in a test or integration environment, or in a production environment, ULS logs should contain valuable pointers to what's happening.

By default, the ULS logs are stored on the file system of every SharePoint server in the *<Program Files Directory>*\Common Files\Microsoft Shared\Web Server Extensions\15\LOGS folder, which is the LOGS folder under the SharePoint root folder. By going into Configure Diagnostic Logging on the Monitoring page in Central Administration, it is possible to specify the folder where the ULS logs are stored. Administrators can change the number of days logs are kept and the total amount of disk space that can be used by the logs. If you are troubleshooting a server that wasn't configured by you and the logs are not in the LOGS folder in the SharePoint root folder, you can browse to the Configure Diagnostic Logging page to find out where the log files are stored on the server. The page can also be used to change the severity of the events that are logged both to the ULS logs and to the Windows event logs. Flood protection can be enabled to make sure that SharePoint events won't flood the Windows event logs.

The ULS logs are text files that are quite difficult to read and that usually contain a lot of data. To more easily read events in the ULS logs and also to search, sort, and filter the ULS logs, the ULS Viewer is a must-have tool for everyone who has to troubleshoot SharePoint. The ULS Viewer can be downloaded from MSDN at *http://archive.msdn.microsoft.com/ULSViewer* and is an .exe file that has to be run on the SharePoint server. It will enable you to start and stop traces and to search through the logs by using a well-organized user interface instead of a text file.

Troubleshooting information for a SharePoint environment can also be found in the Windows event logs. On the server, the Windows event logs can be consulted by opening up the Event Viewer. There is some overlap in the information between the ULS logs and the Windows Event Viewer, but both also contain specific information that can be valuable for finding the source of the problem.

When an error occurs in SharePoint, an error message will be displayed in the user interface that contains what is referred to as a correlation ID. A *correlation ID* is a GUID that uniquely identifies a particular request. The correlation ID is generated by the SharePoint web server that receives the request. Every event that is part of the request is tagged with the same ID, and the ID even persists across different servers in the farm. For instance, if a request was sent to a SharePoint web server, it will be generated on that server and it will mark all entries in the ULS log that are part of the request with that particular correlation ID. If, as part of the request, some managed metadata has to be requested from the Managed Metadata service that runs on a dedicated application server, the same correlation ID can be found in the ULS logs on that application server. You can even use the correlation ID to trace the request on the server that is running SQL Server by using SQL Profiler to filter out requests related to the ID.

When an end user encounters an error in a SharePoint environment, that user will usually see an error message that contains a correlation ID. Even though the ID is of no use to the user himself, users can be asked to include the ID when they place a call to a helpdesk. Having the ID of the user's faulty request can help administrators and developers find out what went wrong with the user's request and help solve the issue.

Correlation IDs aren't just generated for faulty requests; they are generated for all requests. To find the correlation ID for a successful request, you can use the Developer Dashboard.

Using the Developer Dashboard

The Developer Dashboard was introduced in SharePoint 2010 to show performance and tracing information for a SharePoint page in a control on the page itself. In SharePoint 2013, the Developer Dashboard has been dramatically improved. The dashboard is no longer a control on a page; it opens in a separate dedicated window. The dashboard also no longer just contains information about the latest request but contains information about several requests, so that you can compare them if you want to and more easily get an overview. The information on the Developer Dashboard is a lot more detailed than it was in SharePoint 2010. For instance, you can now easily see the SQL requests and the time it took to process them, the different scopes and execution times, service calls, and also all ULS log entries that are related to the selected request. All this can really help you to identify any potential problems related to a request, because you have all the information SharePoint collected about the requests in a single place.

By default, the Developer Dashboard is disabled. You can enable it by using Windows PowerShell. The Windows PowerShell cmdlet only supports *On* or *Off*; the *OnDemand* parameter has been deprecated, although *On* now pretty much acts the way *OnDemand* did in SharePoint 2010; it displays an icon in the upper-right corner that allows you to open up the Developer Dashboard. The Windows PowerShell cmdlet to turn on the Developer Dashboard is displayed in Listing 2-3.

LISTING 2-3 Changing the mode of the Developer Dashboard

```
Add-PSSnapin Microsoft.SharePoint.PowerShell -ErrorAction "SilentlyContinue"
$DevDashboardSettings = [Microsoft.SharePoint.Administration.SPWebService]:: ´
ContentService.DeveloperDashboardSettings
$DevDashboardSettings.DisplayLevel = 'On'
$DevDashboardsettings.Update()
```

Figure 2-5 shows the Developer Dashboard after the welcome page of an out-of-the-box team site has been loaded. On the dashboard, you can see different tabs for server information, scopes, SQL info, ULS information, and a lot more. The Server Info tab contains the total execution time for the page, the current user, whether the page is published, and the correlation ID. The SQL tab also shows the execution time for all database queries and for all methods. By using the dashboard you can not only see how long a page took to load just the out-of-the-box functionality on it, but you can also see how long it takes your custom component to load. You can identify whether your code executes any expensive methods or database queries.

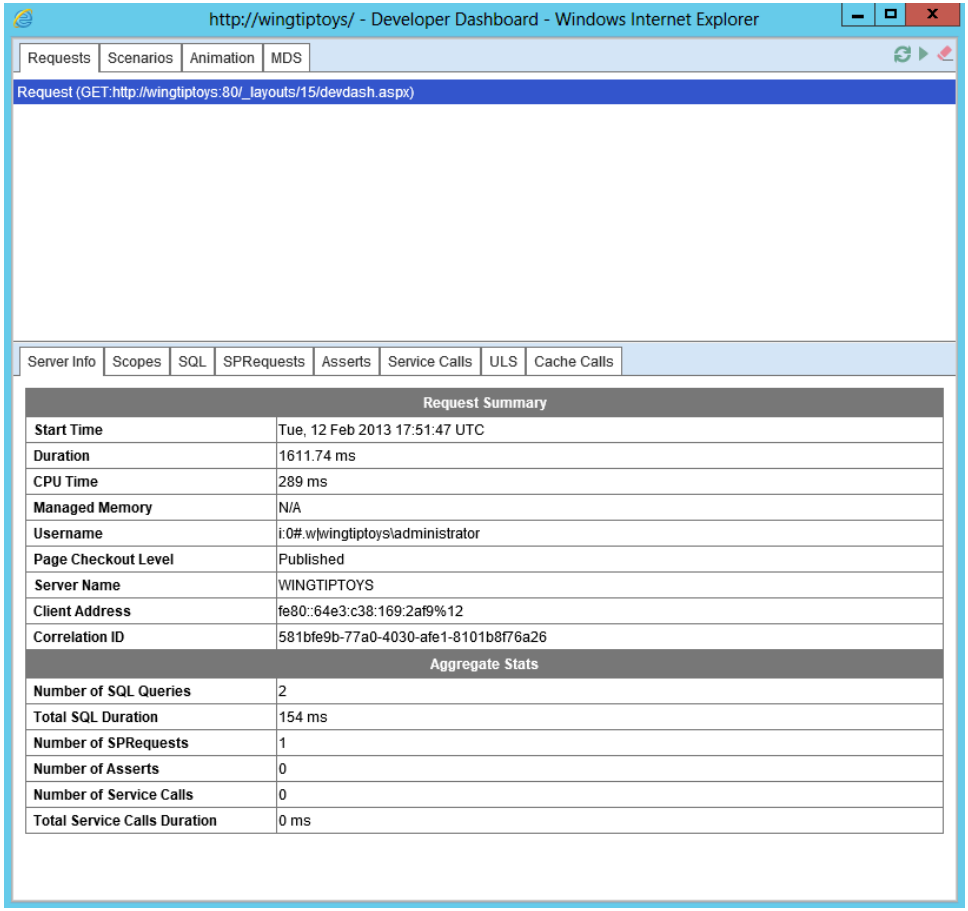


FIGURE 2-5 The Developer Dashboard

To write information from your own custom solution to the Developer Dashboard you can either execute your code in an *OnInit* or *Render* override, or you can wrap your code in an *SPMonitoredScope* block. Only code from farm solutions can send information to the Developer Dashboard; the contents of sandboxed solutions or apps cannot send information to the dashboard.

Using the SharePoint Developer Tools in Visual Studio 2012

The release of SharePoint 2007 was a significant milestone for SharePoint as a development platform because in this version, Microsoft introduced support for features and solution packages. Soon after SharePoint 2007 was released, however, it became clear within Microsoft and throughout the industry that more and better developer productivity tools were needed. With SharePoint 2010, Microsoft extended the developer platform by introducing the SharePoint Developer Tools in Visual Studio 2010. These new tools made developing for SharePoint 2010 much faster and easier because they automated

grungy tasks and hid many of the low-level details that developers had to worry about when developing for SharePoint 2007. For example, every SharePoint project in Visual Studio 2010 is created with built-in support to generate its output as a solution package. The SharePoint Developer Tools also integrate commands into the Visual Studio 2010 environment that make it easy to deploy and retract the solution package for a SharePoint project during testing and debugging.

With the introduction of SharePoint 2013 and Visual Studio 2012, Microsoft has further improved the SharePoint Developer Tools. For the project types that were available for SharePoint 2010 there are now also SharePoint 2013 versions. There are also Office add-in project types that can be used to create add-ins for the Microsoft Office 2013 and Office 2010 applications. The biggest change, however, is that the SharePoint Developer Tools in Visual Studio 2012 contain two project types that allow you to create two different types of apps:

- Apps for Office 2013
- Apps for SharePoint 2013

You can find the app for SharePoint 2013 C# in the New Project dialog box within Visual Studio 2012 by navigating to Visual C#\Office/SharePoint\Apps, as shown in Figure 2-6.

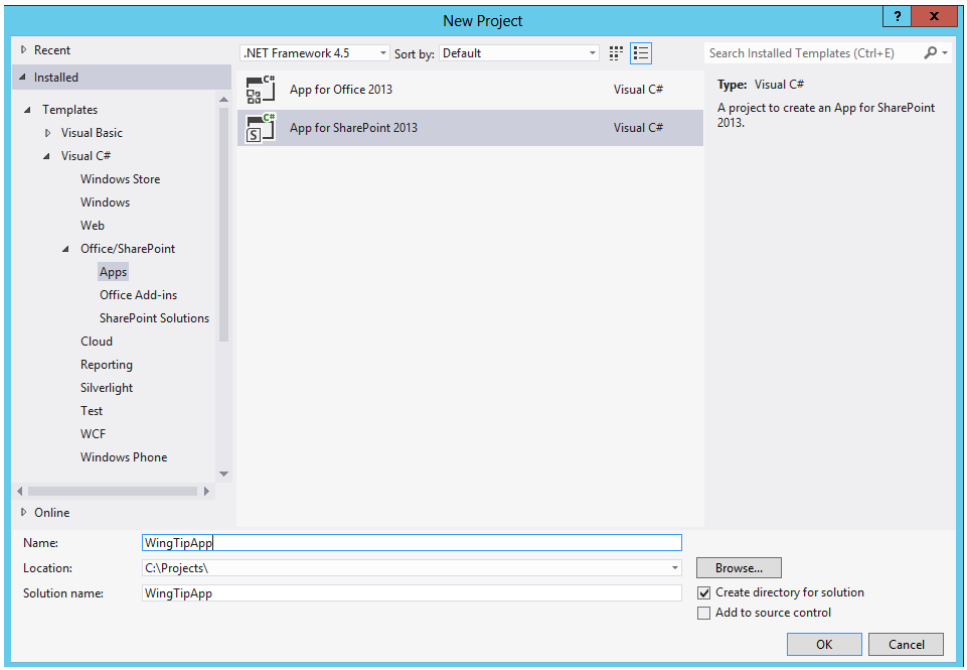


FIGURE 2-6 Selecting the App for SharePoint 2013 project type to create an app

When you click OK you are asked to confirm the name of the app and specify the site to which you want Visual Studio to deploy the app and how you want to host your app, as shown in Figure 2-7.

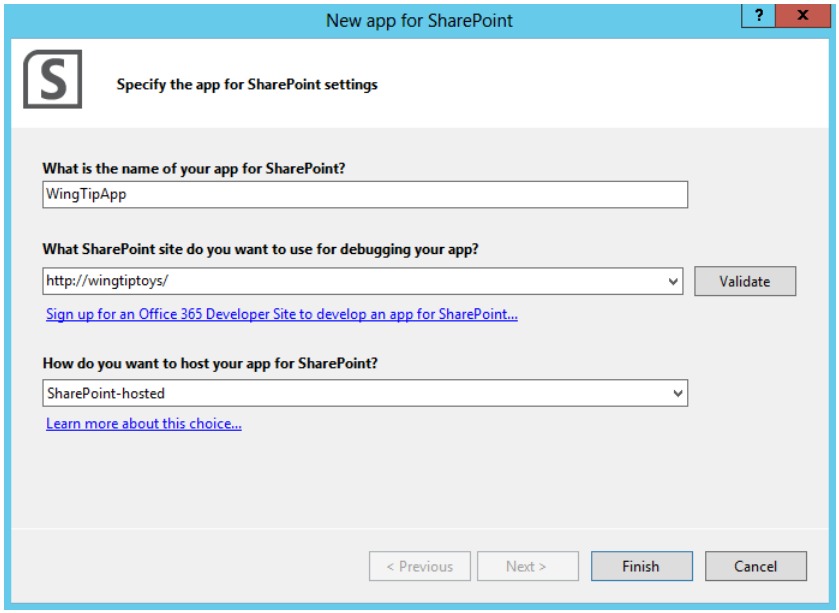


FIGURE 2-7 Specifying a name, test site URL, and trust level for an app

SharePoint apps will be discussed in detail in Chapter 4, "SharePoint apps." For now, you will create a SharePoint-hosted app, which means that all of the app will be deployed to a SharePoint site. When you click the Finish button in the SharePoint Customization Wizard, Visual Studio takes a few seconds to create and configure the new project. Figure 2-8 shows what the new SharePoint project looks like in Solution Explorer.

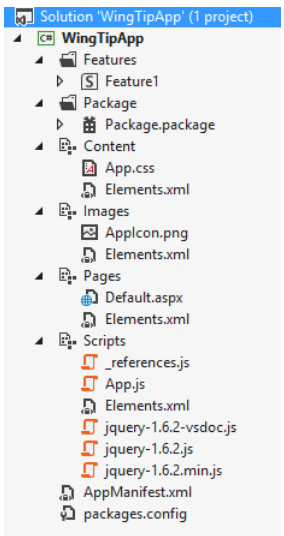


FIGURE 2-8 An app's Features node, Package node, and some standard content

A SharePoint app is created with some default content. There is a style sheet, an app icon image, a default page, and several JavaScript files. The AppManifest.xml file contains metadata such as the name and the title of the app, the app icon, and the scopes at which the app needs to have permissions. You use the *Features* node of the App for SharePoint 2013 project to add new features to the app. Notice that the *Features* node of a SharePoint app contains one feature, called Feature1, when the app is created. Feature1 contains the default content and is web scoped. A feature in a SharePoint app can only be web scoped, it is not supported to use a SharePoint app to deploy a site, web application, or farm-scoped feature. You use the *Package* node to track project-wide settings related to building the project into a solution package .wsp file and an app package .app file.

Just like normal SharePoint projects, SharePoint apps also have three special menu commands to support deployment and packaging of the app: *Deploy*, *Package*, and *Retract*. These menu commands are available when you right-click the top-level project node in Solution Explorer. You can run the *Package* command to build a SharePoint project into a solution package. You can use the *Deploy* command to run a sequence of deployment steps that deploy the solution package in the local farm so that you can test and debug your work. The *Retract* command reverses the act of deployment by retracting the solution package from the local farm. When you click the Deploy button, Visual Studio deploys the app to the site you listed in the Customization Wizard. You can find the app on the Site Contents page and in the navigation pane on the left side of the page, under the Recent heading, as shown in Figure 2-9. Clicking the app name or icon will open the app's default.aspx page.

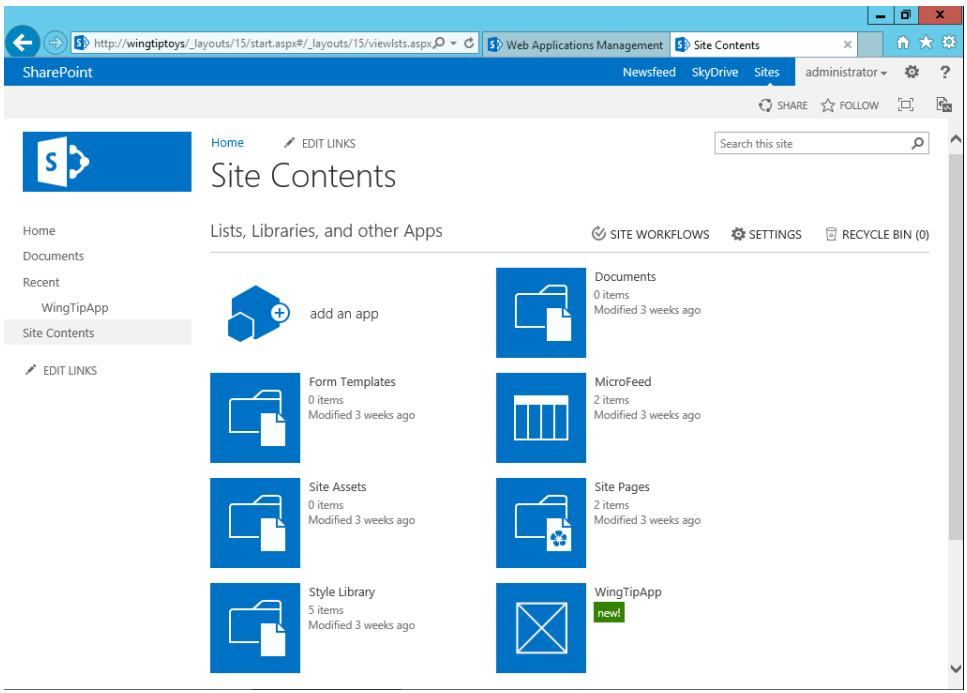


FIGURE 2-9 The WingTip app on the Site Contents page and in the navigation pane on the left side of the page

Choosing a development approach

At least as important as knowing how to create a solution is knowing when to create a solution and what type of solution to create. Even though this is a book about custom development, the best approach when using SharePoint is to use out-of-the-box functionality. Customizations are the number-one cause of problems with SharePoint environments. Knowing this, you have to make sure that when you create a custom solution, the same functionality couldn't be achieved by using out-of-the-box functionality. If you do have to create a custom solution, you have to make sure that you create the right type of solution and that you build the solution in a way that uses the least amount of resources from the SharePoint server.

When you open up Visual Studio, the first thing you have to decide is what type of project you want to create. With the new SharePoint 2013 app model added into the mix, you can now choose between a SharePoint 2013 farm solution, a SharePoint 2013 sandboxed solution, and a SharePoint 2013 app. There isn't one right project type; the best type depends on what kind of customizations you want to build and to what type of environment you want to deploy the customizations.

The best option is always the project type that puts the least amount of load on the SharePoint server. However, the solution also has to be maintainable and upgradable. The solution should not be overly complex, so that it can be maintained by others. The solution design should follow best practices to ensure that it can be upgraded to a next version of SharePoint. Of course, you also have to make sure that the solution can be deployed to the environment that you are creating the customizations for. To summarize, when you create SharePoint customizations, the following things have to be taken into account:

- Put the least possible load on the SharePoint server.
- Keep the customizations as simple as possible.
- Make sure you follow best practices, so the solutions don't block the SharePoint environment from being upgraded.
- Ensure that the solution can be deployed to the target environment.

Because a SharePoint app can't deploy server-side code to the SharePoint server, that will always be the project type that puts the least amount of load on the SharePoint server. Sandboxed solutions can deploy server-side code to the SharePoint server; however, a sandboxed solution can only use a limited amount of server resources before it is shut down. A farm solution can technically use all the resources that are available on the server, and because of that, a farm solution can bring down an entire SharePoint farm. This is not just a theoretical scenario; it actually happens on a regular basis to both small and really large SharePoint environments.

It is likely that the version of the customizations that you are building is not the last version. Either the customizations will be a huge hit and users will ask for more functionality, or they aren't what users were expecting and they need modifications to fit the user's needs. To make the solution easy to maintain, make it as simple as possible. This means that even though your first choice should be to build a SharePoint app, because that would consume the least amount of SharePoint server resources,

you shouldn't do it at all costs. If creating an app would create a solution that is significantly more complex, the best solution is probably to build a sandboxed or farm solution.

If the SharePoint environment is successful, it's very likely that at some point the environment will have to be upgraded to the next version of SharePoint. Some SharePoint customizations can block the upgrade of a SharePoint environment. For instance, if a solution makes unsupported changes to the database or to SharePoint files on the file system, it won't be possible to upgrade the environment. Also, when certain customizations such as site definitions are involved, it will be significantly more difficult to move the contents of the SharePoint farm to a cloud-hosted environment. Because Microsoft has positioned the SharePoint app model as the SharePoint customization type of the future, and because almost all parts of the SharePoint app live outside of SharePoint, creating apps is your safest bet when you want to guarantee that your environment can upgrade without being hindered by the customizations.

Using farm solutions means that the solution will have to be upgraded when the SharePoint farm gets upgraded. Sandboxed solutions live in the content database and can only affect the site collection that they are deployed in. This means that the impact they can have on an upgrade to a new version of SharePoint is a lot smaller than for farm solutions. However, sandboxed solutions are deprecated in SharePoint 2013. This means that although both sandboxed solutions that were created for SharePoint 2010 and new sandboxed solutions created for SharePoint 2013 are still fully supported, Microsoft is planning to remove support for sandboxed solutions at some point in the future. This doesn't necessarily have to be in the next release; it could be in the release after that, or an even later release. Investing in large-scale sandboxed solutions is probably not a good idea, though.

If your environment is a cloud-hosted environment such as Office 365, you might not have much of a choice, because you won't be allowed to deploy full trust solutions, and not all customizations can be created by using apps. In those cases, you will probably still want to create new sandboxed solutions. The advice would then be to try and create the solutions in such a way that you can remove the sandboxed solution without the entire site breaking down. When Microsoft then at some point removes the support for sandboxed solutions, at least your existing content is still accessible.

When you are upgrading from SharePoint 2010 to SharePoint 2013, you can choose how you want to upgrade you custom solutions:

- You can deploy your SharePoint 2010 solutions as is. Microsoft has designed SharePoint 2013 to make sure that your solutions keep working. This means that the entire solution should work when a site is in SharePoint 2010 mode. When a site is in SharePoint 2013 mode, most of the solution should work, but some things might not work. Things that don't work are mostly related to functionality that is no longer available in SharePoint. For instance, custom solutions that use the SharePoint 2010 Web Analytics features will not work in a SharePoint 2013 environment, because the Web Analytics Service Application and all related functionality has been removed. In most cases, you will not just deploy your existing solutions as is. The only case in which it makes sense to not make any changes and just deploy an existing SharePoint 2010 solution as is to a SharePoint 2013 environment is when the solution is only there to keep existing SharePoint 2010 sites working in SharePoint 2010 mode in the SharePoint 2013 environment. In all other cases, you will at least want to recompile your solution.

- The second way to move SharePoint 2010 solutions to a SharePoint 2013 environment is to open the SharePoint 2010 solution in Visual Studio 2012. The solution can then be recompiled against the SharePoint 2013 DLLs. While you are doing this, you can decide to make some minor changes to the solution. You should definitely remove any references to functionality that no longer exists in SharePoint 2013, such as the Web Analytics functionality. Another example of something that will definitely not work in a SharePoint 2013 mode are SharePoint 2010 visual designs. Any master pages and style sheets that were created for SharePoint 2010 will not work in sites that are in SharePoint 2013 mode. Normally SharePoint will just not use the SharePoint 2010 designs in SharePoint 2013 mode sites. However, if you had a stapling feature in SharePoint 2010 that stapled a custom design to sites in the environment, especially if it stapled the design to all sites in the environment by stapling it to the GLOBAL site definition, you will want to remove the stapling feature before moving the solution to SharePoint 2013. Stapling a SharePoint 2010 design to the GLOBAL site definition and deploying it to SharePoint 2013 can make it impossible to create any fully functional sites, either by using out-of-the-box site definitions or custom web templates.
- The third approach that can be taken when moving solutions from SharePoint 2010 to SharePoint 2013 is to rebuild the solution to use the new SharePoint 2013 functionality where possible. Rebuilding the solution could mean replacing custom features with new out-of-the-box functionality. In a lot of cases, the aim should be to minimize the amount of customizations in an environment, which means that cutting customizations in favor of new out-of-the-box functionality is a very valid change to invest in. Do keep in mind, though, that in order to be able to upgrade existing sites you might need to have certain SharePoint 2010 customizations deployed in your environment, even if you don't want to actively use them anymore in your SharePoint 2013 environment. There are also cases in which a customization can't be replaced by out-of-the-box functionality but could be replaced by SharePoint apps, by an application that runs external to SharePoint and that uses the SharePoint Web Services or the vastly improved SharePoint Client Object Model. This should only be done if replacing the functionality by using a SharePoint app or an external solution doesn't make the solution significantly more complex. If that is not the case, rebuilding functionality as a SharePoint app or as an external service will give you practice and experience in using the new development options, and it will make sure that your solution is easier to deploy in hosted environments. Assuming that the SharePoint app model is here to stay, it will also make sure that your solution becomes more future proof.

Using the SharePoint APIs

In SharePoint 2013, you can now choose from three different APIs: the server-side object model (SSOM), the client-side object model (CSOM), and the REST API. All three APIs give you the option to build customizations for your SharePoint environment. This section will provide an overview of the three different APIs. Each API will then be used extensively in examples throughout the book. For specific detailed coverage of CSOM and REST, see Chapter 5, "Client-side programming."

Understanding the server-side object model

The core server-side object model of SharePoint Foundation is loaded through an assembly named *Microsoft.SharePoint.dll*. When you reference this assembly within a Visual Studio 2012 project, you can start programming against the classes in the server-side object model, such as *SPSite, SPWeb*, and *SPList*. There are two initial requirements for a Visual Studio project that programs against the server-side object model by using the *Microsoft.SharePoint* assembly. First, the project must be configured to use .NET Framework 4 or 4.5 as its target framework. Pay extra attention if you are upgrading a SharePoint 2010 solution, because that will have been built using the .NET Framework 3.5 as its target framework. The second requirement is that your project must have a platform target setting that is compatible with a 64-bit environment, which is essential for properly loading the *Microsoft.SharePoint* assembly.

Another critical requirement for any application or component that is programmed against the server-side object model is that the application or component must be deployed and run on a SharePoint server in the farm in which you want to use the component. The deployment of applications or components that use the SharePoint server-side object model should always be done by using a SharePoint Solution or .wsp file. To deploy the solution, you will need access to at least one SharePoint server in the farm where the solution should be deployed. In most production environments, this means that you will hand off the solution and a document that describes how to deploy the solution to the administrator of the server. In your development environment, Visual Studio will usually do the deployment for you.

You can also create client applications with Visual Studio 2012 that program against the server-side object model. For example, you can create a standard console application that uses the server-side object model to access a site and the elements inside the site, such as lists and items. However, keep in mind that any client application that depends on the *Microsoft.SharePoint* assembly can be run only when launched on a server that has SharePoint installed on it and that is part of a SharePoint farm. This means that it's not likely that you will encounter real-world scenarios that call for creating client applications that use the server-side object model. Even so, creating simple console applications that program against the *Microsoft.SharePoint* assembly in your development environment can be useful, because it gives you a quick and easy way to write and test code as you begin learning the server-side object model.

Most of the SharePoint Foundation APIs reside in *Microsoft.SharePoint.dll*. However, if you are building a custom solution ,using the server-side object model you might also want to use SharePoint Server APIs and functionality. The bulk of the SharePoint Server APIs reside in *Microsoft.Office.Server.dll*; however, this isn't the only available DLL that contains SharePoint Server APIs. For a full list of SharePoint APIs and the DLLs in which you can find them, see the MSDN page *.NET server API reference for SharePoint 2013* at *http://msdn.microsoft.com/en-us/library/jj193058.aspx*.

Using the client-side object model

SharePoint 2010 introduced the SharePoint Foundation client-side object model, which allows developers to use SharePoint content and objects in their client-side solutions. As a developer, you could now create a very simple solution that would be deployed into a SharePoint site or onto a user's desktop and that could read or manage data in a SharePoint site.

In SharePoint 2010, the client-side object model was only available for SharePoint Foundation objects. In SharePoint 2013, however, the client-side object model has again been vastly improved by making a lot of the SharePoint Server objects available through the client-side object model. In SharePoint 2010 there were three client-side object models, and in SharePoint 2013 there are four. SharePoint 2013 allows you to choose between the Managed, Silverlight, Mobile, and JavaScript object models. Each of the four object models provides an object interface to SharePoint functionality that is based on the objects available in the *Microsoft.SharePoint* namespace. All four client-side object models also have support for at least part of the SharePoint Server 2013 functionality, but not all of them include the same SharePoint Server 2013 components.

The four client-side object models also all have their own usages. Each of the four object models presents an object interface in front of a service proxy. Developers write client-side code by using the object model, but the operations are batched and sent as a single XML request to the Client.svc service. When the XML request is received, the Client.svc service makes calls to the server-side object model on behalf of the client. The results of the server-side calls are then sent back to the calling client in the form of a JavaScript Object Notation (JSON) object.

The object model for Microsoft Silverlight can be used to build Silverlight applications, Web Parts, ASP.NET applications, apps for SharePoint and Office, and Silverlight applications for phones that use SharePoint data or SharePoint objects. A Silverlight application is compiled into an .xap file that can pretty much be stored anywhere. Examples of where .xap files can be deployed are a client computer, the file system of a SharePoint server, a list in a SharePoint library, and an external (web) server. The Silverlight client-side object model is contained in assemblies in the LAYOUTS\ClientBin folder. The following DLLs are available:

- Microsoft.SharePoint.Client.Silverlight.dll
- Microsoft.SharePoint.Client.Silverlight.Runtime.dll
- Microsoft.SharePoint.Client.DocumentManagement.Silverlight.dll
- Microsoft.SharePoint.Client.Publishing.Silverlight.dll
- Microsoft.SharePoint.Client.Search.Applications.Silverlight.dll
- Microsoft.SharePoint.Client.Search.Silverlight.dll

- Microsoft.SharePoint.Client.Taxonomy.Silverlight.dll
- Microsoft.SharePoint.Client.UserProfiles.Silverlight.dll
- Microsoft.SharePoint.Client.WorkflowServices.Silverlight.dll
- Microsoft.Office.Client.Policy.Silverlight.dll
- Microsoft.Office.Client.TranslationServices.Silverlight.dll

The Mobile object model can be used to create applications that run on Windows Phones. The Mobile client-side object model is a special version of the Silverlight client-side object model. The Mobile object model contains most of the same functionality as the Silverlight object model. A couple of areas are missing, but when you are creating a Windows Phone application using the Mobile object model you can use the REST APIs to access these areas. The Mobile client-side object model also contains some functionality that is specific to phones, such as APIs that enable a phone app to register for notifications from the Microsoft Push Notification Service. The Mobile client-side object model can be found in the same folder as the Silverlight client-side object model, in the LAYOUTS\ClientBin folder. The DLLs that are available for the Mobile client-side object model are:

- Microsoft.SharePoint.Client.Phone.dll
- Microsoft.SharePoint.Client.Phone.Runtime.dll
- Microsoft.SharePoint.Client.DocumentManagement.Phone.dll
- Microsoft.SharePoint.Client.Publishing.Phone.dll
- Microsoft.SharePoint.Client.Taxonomy.Phone.dll
- Microsoft.SharePoint.Client.UserProfiles.Phone.dll
- Microsoft.Office.Client.Policy.Phone.dll
- Microsoft.Office.Client.TranslationServices.Phone.dll

The Managed object model can be used to create .NET applications that run on Windows operating systems that aren't phones or SharePoint servers. This means that the Managed object model can be used to create applications that run on client computers, or on Windows web servers not running SharePoint. The Managed object model can be found in the ISAPI folder and is contained in the following DLLs:

- Microsoft.SharePoint.Client.dll
- Microsoft.SharePoint.Client.Runtime.dll
- Microsoft.SharePoint.Client.ServerRuntime.dll
- Microsoft.SharePoint.Client.DocumentManagement.dll
- Microsoft.SharePoint.Client.Publishing.dll
- Microsoft.SharePoint.Client.Search.Applications.dll

- Microsoft.SharePoint.Client.Search.dll
- Microsoft.SharePoint.Client.Taxonomy.dll
- Microsoft.SharePoint.Client.UserProfiles.dll
- Microsoft.SharePoint.Client.WorkflowServices.dll
- Microsoft.Office.Client.Education.dll
- Microsoft.Office.Client.Policy.dll
- Microsoft.Office.Client.TranslationServices.dll
- Microsoft.Office.SharePoint.ClientExtensions.dll

The last client-side object model is the JavaScript object model. The JavaScript object model can be used in inline script or in separate .js files. Using the JavaScript client-side object model is an excellent way to add custom SharePoint code to a SharePoint-hosted app. The JavaScript object model is different from the other three in that it is not contained in a set of DLLs. Instead, it is contained in a JavaScript library, inside of .js files. The many .js files that make up the JavaScript client-side object model are located in the LAYOUTS folder. The core SharePoint functionality can be found in SP.js and in SP.Core.js.

Though the four client-side object models don't contain exactly the same functionality, Microsoft has taken great care to ensure that the four models return objects that behave similarly. This means that if you know how to write code against one of the models, you can easily port that code to either of the other three models. Table 2-2 shows some of the main objects supported by each model alongside the related object from the server-side model.

TABLE 2-2 Equivalent objects in the server and client models

Server model	Managed model	Silverlight model	Mobile model	JavaScript model
SPContext	ClientContext	ClientContext	ClientContext	ClientContext
SPSite	Site	Site	Site	Site
SPWeb	Web	Web	Web	Web
SPList	List	List	List	List
SPListItem	ListItem	ListItem	ListItem	ListItem
SPField	Field	Field	Field	Field

As in the standard code you write against the server-side object model, code written for client object models requires a starting point in the form of a context object. The context object provides an entry point into the associated API that can be used to gain access to other objects. When you have access to the objects, you can interact with the scalar properties of the object freely (for example, *Name, Title, Url*, and so on). Listing 2-4 shows how to create a context in each of the models and return an object representing a site collection. After the site collection object is returned, the *Url* property is examined. Code for the server model is included for comparison.

LISTING 2-4 Creating contexts

```
//Server Object Model
SPSite siteCollection = SPContext.Current.Site;
string url = siteCollection.Url;

//Managed Client Object Model
using (ClientContext ctx = new ClientContext("http://intranet.wingtiptoys.com"))
{
    Site siteCollection = ctx.Site;
    ctx.Load(siteCollection);
    ctx.ExecuteQuery();
    string url = siteCollection.Url;
}

//Silverlight Client Object Model
using (ClientContext ctx =
    new ClientContext("http://intranet.wingtiptoys.com"))
{
    Site siteCollection = ctx.Site;
    ctx.Load(siteCollection);
    ctx.ExecuteQuery();
    string url = siteCollection.Url;
}

//Mobile Client Object Model
using (ClientContext ctx =
    new ClientContext("http://intranet.wingtiptoys.com"))
{
    Site siteCollection = ctx.Site;
    ctx.Load(siteCollection);
    ctx.ExecuteQuery();
    string url = siteCollection.Url;
}

//JavaScript Client Object Model
var siteCollection;
function getSiteCollection
{
    var ctx = new SP.ClientContext("/");
    siteCollection = ctx.get_site;
    ctx.load(site);
    ctx.executeQueryAsync(success, failure);
}

function success {
    string url = siteCollection.get_url;
}

function failure {
    alert("Failure!");
}
```

The *ClientContext* class in the Managed, Silverlight, and Mobile object models inherits from the *ClientContextRuntime* class. By using the *ClientContext* class, you can get a valid run-time context by passing in the URL of a site. In addition, this class provides several members that are needed to access data and invoke methods on the server.

The *SP.ClientContext* class in the JavaScript client object model inherits from the *SP.ClientContext-Runtime* class and provides equivalent functionality to the *ClientContext* class found in the Managed, Silverlight, and Mobile client object models. As with the Managed and Silverlight models, you can get a run-time context in the JavaScript model by using the *SP.ClientContext* class and passing a URL. Unlike the other client object models, however, the JavaScript model also allows you to get a run-time context to the current site by using a constructor with no arguments, so the example above could be rewritten as simply *var ctx = new SP.ClientContext*.

All four client-side object models only communicate with the SharePoint server when the code calls the *ExecuteQuery* or *ExecuteQueryAsync* method. This is to prevent the object models from making too many calls to the SharePoint server and from affecting the SharePoint server's health by querying the server too much. This means that when you are writing your code, you have to really think about when the statements that you are writing actually have to be executed on the server. You will want to minimize traffic to the server, but you will need to communicate with the server if you want to request data from, or send data into, the SharePoint environment.

The *ExecuteQuery* method creates an XML request and passes it to the Client.svc service. The client then waits synchronously while the batch is executed and the JSON results are returned. The *Execute-QueryAsync* method, which is used in the Silverlight and Mobile client object models, sends the XML request to the server, but it returns immediately. Designated success and failure callback methods receive notification when the batch operation is complete.

The JavaScript model works like the Managed and Silverlight models by loading operations and executing batches. In the case of the JavaScript model, however, all batch executions are accomplished asynchronously. This means that you must call the *ExecuteQueryAsync* method and pass in the name of functions that will receive success and failure callbacks, as shown earlier in Listing 2-4.

Using the REST APIs

The most lightweight option for performing relatively simple operations on data in SharePoint lists and sites is to use the REST capabilities that are built into SharePoint 2013. The SharePoint 2013 implementation of a REST web service uses the Open Data Protocol (OData) to perform CRUD operations on data in SharePoint. Using REST allows your code to interact with SharePoint by using standard HTTP requests and responses. Table 2-3 shows the mapping between HTTP verbs and data operations.

TABLE 2-3 Mapping between HTTP verbs and data operations

HTTP verb	Data operation
GET	Retrieve
POST	Create
PUT	Update all fields
DELETE	Delete
MERGE	Update specified fields

The Client.svc web service handles the HTTP request and serves a response in either Atom or JSON format.

To access any object on a site by using a RESTful call, the URL you should use will start with the following construction:

```
http://<server>/<site>/_api
```

To access an actual object within the site you simply add the object to the URL:

```
//Access a site collection
http://<server>/<site>/_api/site

//Access a specific site
http://<server>/<site>/_api/web

//Access a list in a specific site
http://<server>/<site>/_api/web/lists('GUID')
```

You can use the querystring syntax to specify parameters for the methods that you call by using a RESTful HTTP request:

```
//Apply a "blank" site site definition to a SharePoint site
http://<server>/<site>/_api/web/applyWebTemplate?template="STS#1"
```

The query strings can become rather complex, but because of that the queries that can be performed are rather powerful as well. You can select, sort, page, filter, and expand data by using a RESTful query. The filtering allows both numeric and string comparisons as well as date and time comparisons. The next example of a RESTful query requests the FirstName, LastName, and PhoneNumber columns from a list with a specific GUID and filters the items by items where the FirstName starts with an *a*:

```
http://<server>/<site>/_api/web/lists('GUID')/items?$select=FirstName,LastName,
PhoneNumber$filter=startWith(FirstName, a)
```

Summary

In this chapter, all the basics of developing a custom SharePoint solution have been touched on. The first step is to determine what type of development environment you need to create the solution that you want to create, or to complete the project that you are working on. When you have your hardware and the design of the development environment in place, you can install and configure it manually, but you can also use Windows PowerShell to configure your server. Especially if you need to create multiple development environments, scripting the installation and configuration can save you time and will help to ensure that all development environments are identical.

The next step is to determine a development approach. The best approach for your solution depends on the functionality that you want to build and on the environment that the solution should be deployed to. If the solution has to be deployed to a cloud-hosted environment, creating a farm solution is not an option, because you won't be able to deploy it. The most future-proof approach is to create a SharePoint app. However, some of the functionality that you might want to build might not be able to be created by using a SharePoint app (yet). This forces you to make a decision between creating an app that implements as much of the functionality as possible and creating a farm solution that implements the exact functionality that you are looking for. If you deploying a farm solution, you will have to upgrade it if you want to upgrade your environment, and at some point in the future Microsoft might remove support for farm solutions completely. Although it will be a while before this happens, it should already be a consideration when you are determining the development approach you are going to use for your solution.

After your solution is deployed, you might have to debug it. There are several debugging tools that can be used to debug custom SharePoint solutions. The best tool to use depends on the type of problem you are trying to debug and on what type of environment your solution is in. The ULS and Windows Event Logs, and the Developer Dashboard, can give you valuable information from all types of environments and are all useful tools to help you identify the cause of a problem on your farm.

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 authors

Scot Hillier is an independent consultant and Microsoft SharePoint Most Valuable Professional (MVP) focused on creating solutions for information workers with SharePoint, Microsoft Office, and related .NET technologies. He is the author/coauthor of 15 books and DVDs on Microsoft technologies, including *Inside Microsoft SharePoint 2010* (Microsoft Press, 2011). Scot splits his time between consulting on SharePoint projects, speaking at SharePoint events such as Tech Ed, and delivering training for SharePoint developers. Scot is a former US Navy submarine officer and is a graduate of the Virginia Military Institute. Scot can be reached at *scot@shillier.com*.

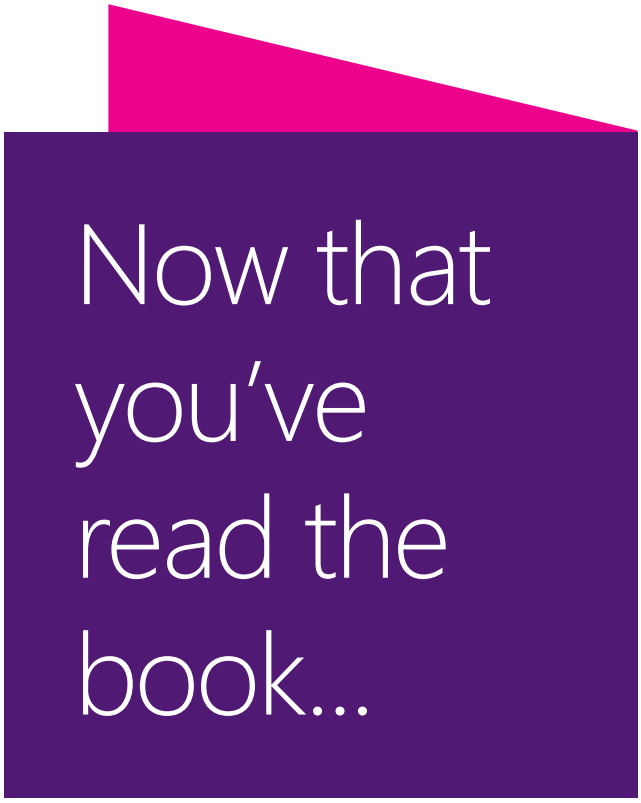
Mirjam van Olst works as a SharePoint Architect for Avanade in the Netherlands. Having worked with different versions of SharePoint since 2004, she has helped companies in different industries and of different sizes to implement successful SharePoint portal, ECM, and search solutions. Mirjam is one of the very few Microsoft Certified Masters for both SharePoint 2007 and SharePoint 2010. Being a strong community advocate, Mirjam is a co-organizer of the Dutch Information Worker User Group (DIWUG). Mirjam is a regular author and editor for the popular DIWUG eMagazine. Mirjam is a regular speaker at both national and international conferences and events and can be found blogging at *http://sharepointchick.com*. Mirjam has been awarded the Microsoft MVP award since 2010. In her spare time Mirjam likes to play tennis and hang out with friends and family.

Ted Pattison has been writing technical books for software developers, speaking at industry conferences, and leading technical training classes for the last 20 years. In March 2003, his professional focus turned to SharePoint technologies when he began to work with the beta of SharePoint Server 2003. As a recognized author and trainer within the industry, Ted was selected by Microsoft early in the beta lifecycle of SharePoint 2007, SharePoint 2010, and SharePoint 2013 to author developer-focused training material for early adopters, and he has been fortunate to gain many close contacts within the SharePoint team at Microsoft over the years. Currently, Ted is the owner and president of Critical Path Training (*www.CriticalPathTraining.com*), a company dedicated to training and education focusing on SharePoint technologies. Ted manages the curriculum at Critical Path Training and also serves as a senior instructor training hundreds of professional developers and IT pros on SharePoint 2013 and Office 365 each year. Ted has been a SharePoint MVP 19 years running since he was originally awarded it as a SharePoint Server MVP in 1994.

Andrew Connell is an independent consultant who enjoys development, writing, and teaching. He has a background in content management solutions and web development that spans back to his time as a student at the University of Florida in the late 1990s, managing class websites. He has consistently focused on the challenges facing businesses today to maintain a current and dynamic online presence without having to rely constantly on web developers or have a proficiency in web technologies. Andrew is an nine-time recipient of Microsoft's MVP award (2005-2013) for Microsoft Content Management Server (MCMS) and SharePoint Server. You can learn from Andrew by taking one of his hands-on courses through Critical Path Training (*www.CriticalPathTraining.com*) or through one of the many on-demand classes he has published though Pluralsight (*www.Pluralsight.com*). He has authored and contributed to numerous MCMS and SharePoint books over the years, including *Professional SharePoint 2007 Web Content Management Development* (Wrox, 2008), *Inside Microsoft SharePoint 2010* (Microsoft Press, 2011), and *Real World SharePoint 2010* (Wrox, 2010) and is the author of numerous articles on the Microsoft Developer Network (MSDN) and in various magazines. Andrew has presented at numerous conferences and taught in the United States, Canada, Australia, England, Spain, Norway, Sweden, and the Netherlands. You can find Andrew on his blog (*http://www.andrewconnell.com*), follow him on Twitter *@andrewconnell*, or email him at *me@andrewconnell.com*.

Wictor Wilén is one of the few Microsoft Certified Architects in the world and Microsoft Certified Solutions Master for SharePoint. He works as Director and Solution Architect at Connecta AB in Sweden. Wictor has worked in the portal and web content management industry since 1998 for consulting companies, founded and sold his own software company and saw the dawn of SharePoint back in 2001. Wictor is an active SharePoint community participant, author, tutor, and frequent speaker at local and international conferences. Since 2010 Wictor has been awarded the SharePoint Server MVP title by Microsoft for his community contributions. He can be found online at *http://www.wictorwilen.se/*. Wictor is based in Stockholm, Sweden.

Kyle Davis is the Solution Architect for the Emerging Technology Group at Catapult Systems. Kyle is a frequent speaker at various Microsoft events and enjoys traveling across the nation sharing best practices and different approaches to solve business needs. Kyle spends most of his time architecting solutions built with emerging technologies and meeting with businesses on how they can do the same. Kyle holds SharePoint MCITP and MCPD certifications and can be followed at *@cldarchitect* on Twitter.



Tell us what you think!

Was it useful?
Did it teach you what you wanted to learn?
Was there room for improvement?

Let us know at http://aka.ms/tellpress

Your feedback goes directly to the staff at Microsoft Press,
and we read every one of your responses. Thanks in advance!

Microsoft