

OFFICIAL MICROSOFT LEARNING PRODUCT

20488B

**Developing Microsoft® SharePoint® Server
2013 Core Solutions**

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2013 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at

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

Product Number: 20488B

Part Number (if applicable): X18-85650

Released: 10/2013

MICROSOFT LICENSE TERMS MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

- a. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
- b. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
- c. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
- d. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of a MPN Member, or (iii) a Microsoft full-time employee.
- e. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
- f. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
- g. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.
- h. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.
- i. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
- j. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.
- k. "MPN Member" means an active silver or gold-level Microsoft Partner Network program member in good standing.

- l. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
- m. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.
- n. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.
- o. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. **USE RIGHTS.** The Licensed Content is licensed not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

a. **If you are a Microsoft IT Academy Program Member:**

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
- ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 - 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 - 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 - 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,**provided you comply with the following:**
- iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
- iv. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
- v. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
- vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

- vii. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
- viii. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
- ix. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

b. If you are a Microsoft Learning Competency Member:

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
- ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 - 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
 - 2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 - 3. you will provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,
provided you comply with the following:
- iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
- iv. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
- v. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
- vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
- vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for your Authorized Training Sessions,
- viii. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
- ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
- x. you will only provide access to the Trainer Content to Trainers.

c. **If you are a MPN Member:**

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
- ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,
provided you comply with the following:
- iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
- iv. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
- v. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
- vi. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
- vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
- viii. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
- ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
- x. you will only provide access to the Trainer Content to Trainers.

d. **If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

e. **If you are a Trainer.**

- i. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

- ii. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of “*customize*” refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.

2.3 **Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.

2.4 **Third Party Programs and Services.** The Licensed Content may contain third party programs or services. These license terms will apply to your use of those third party programs or services, unless other terms accompany those programs and services.

2.5 **Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content’s subject matter is based on a pre-release version of Microsoft technology (“**Pre-release**”), then in addition to the other provisions in this agreement, these terms also apply:

- a. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
- b. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
- c. **Pre-release Term.** If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest (“**Pre-release term**”). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.

- 4. SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
- access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
 - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
 - modify or create a derivative work of any Licensed Content,
 - publicly display, or make the Licensed Content available for others to access or use,
 - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
 - work around any technical limitations in the Licensed Content, or
 - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
- 5. RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.
- 6. EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
- 7. SUPPORT SERVICES.** Because the Licensed Content is “as is”, we may not provide support services for it.
- 8. TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
- 9. LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.
- 10. ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.
- 11. APPLICABLE LAW.**
- a. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

b. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

- 12. LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
- 13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
- 14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection des consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit local, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised September 2012

Welcome!

Thank you for taking our training! We've worked together with our Microsoft Certified Partners for Learning Solutions and our Microsoft IT Academies to bring you a world-class learning experience—whether you're a professional looking to advance your skills or a student preparing for a career in IT.

- **Microsoft Certified Trainers and Instructors**—Your instructor is a technical and instructional expert who meets ongoing certification requirements. And, if instructors are delivering training at one of our Certified Partners for Learning Solutions, they are also evaluated throughout the year by students and by Microsoft.
- **Certification Exam Benefits**—After training, consider taking a Microsoft Certification exam. Microsoft Certifications validate your skills on Microsoft technologies and can help differentiate you when finding a job or boosting your career. In fact, independent research by IDC concluded that 75% of managers believe certifications are important to team performance¹. Ask your instructor about Microsoft Certification exam promotions and discounts that may be available to you.
- **Customer Satisfaction Guarantee**—Our Certified Partners for Learning Solutions offer a satisfaction guarantee and we hold them accountable for it. At the end of class, please complete an evaluation of today's experience. We value your feedback!

We wish you a great learning experience and ongoing success in your career!

Sincerely,

Microsoft Learning
www.microsoft.com/learning

Microsoft | Learning

¹ IDC, Value of Certification: Team Certification and Organizational Performance, November 2006

Acknowledgments

Microsoft Learning wants to acknowledge and thank the following for their contribution toward developing this title. Their effort at various stages in the development has ensured that you have a good classroom experience.

Jason Lee – Subject Matter Expert

Jason Lee is a Principal Technologist with Content Master where he has been working with Microsoft products and technologies for several years, specializing in SharePoint development and architecture. Jason holds a PhD in computing and is currently MCPD and MCTS certified. You can read Jason's technical blog at www.jrjlee.com.

Mike Sumsion – Subject Matter Expert

Mike Sumsion is a Senior Technologist with Content Master, where he has worked with a range of Microsoft technologies. Mike has worked with SharePoint products since Microsoft Office SharePoint Server 2007, and has produced a variety of courseware, hands-on-labs, and other documentation. Michael has an honors degree in Computer Science, and is currently MCTS certified. You can read Mike's blog at www.mikesumsion.com.

Alistair Matthews - Subject Matter Expert

Alistair Matthews is the Principal Consultant for Web Dojo Ltd., a UK consultancy company specializing in Microsoft Technologies. Alistair concentrates on SharePoint and ASP.NET and has extensive experience of other Microsoft products including SQL Server, Windows Azure, and the .NET Framework, which he has gained over a 15 year career. Alistair holds an MA in Natural Sciences from Cambridge University and lives the telecommuting dream in Cornwall, UK.

Lin Joyner - Content Developer

Lin Joyner is a Principal Technologist with Content Master, where she has been working with and writing about Microsoft technologies for more than 13 years. She is an experienced .NET Framework application developer and has previously been an MCT and MCSD.

Paul Barnes - Content Developer

Paul Barnes is a Graduate Technologist with Content Master where he has been working with Microsoft products and technologies, specializing in Microsoft Visual C#. Paul holds a BSc in computing and is currently working towards MCSD certification.

Scot Hillier – Technical Reviewer

Scot Hillier is an independent consultant and Microsoft SharePoint Most Valuable Professional (MVP) focused on creating solutions for Information Workers with SharePoint, Office, and related technologies. He is the author/coauthor of 18 books on Microsoft technologies including "Inside SharePoint 2013" and "App Development in SharePoint 2013". Scot splits his time between consulting on SharePoint projects, speaking at SharePoint events like the Microsoft SharePoint Conference, and delivering training for SharePoint developers through Critical Path Training. Scot is a former U. S. Navy submarine officer and graduate of the Virginia Military Institute. He can be reached at scot@scothillier.net.

Contents

Module 1: SharePoint as a Developer Platform

Lesson 1: Introducing the SharePoint Developer Landscape	page 2
Lesson 2: Choosing Approaches to SharePoint Development	page 13
Lesson 3: Understanding SharePoint 2013 Deployment and Execution Models	page 24
Lab: Comparing Web Parts and App Parts	page 30

Module 2: Working with SharePoint Objects

Lesson 1: Understanding the SharePoint Object Hierarchy	page 2
Lesson 2: Working with Sites and Webs	page 13
Lab A: Working with Sites and Webs	page 21
Lesson 3: Working with Execution Contexts	page 25
Lab B: Working with Execution Contexts	page 30

Module 3: Working with Lists and Libraries

Lesson 1: Using List and Library Objects	page 2
Lesson 2: Querying and Retrieving List Data	page 16
Lab A: Querying and Retrieving List Data	page 27
Lesson 3: Working with Large Lists	page 31
Lab B: Working with Large Lists	page 35

Module 4: Designing and Managing Features and Solutions

Lesson 1: Understanding Features and Solutions	page 2
Lesson 2: Configuring Features and Solutions	page 12
Lesson 3: Working with Sandboxed Solutions	page 23
Lab: Working with Features and Solutions	page 31

Module 5: Working with Server-Side Code

Lesson 1: Developing Web Parts	page 2
Lesson 2: Using Event Receivers	page 7
Lesson 3: Using Timer Jobs	page 13
Lesson 4: Storing Configuration Data	page 21
Lab: Working with Server-Side Code	page 28

Module 6: Managing Identity and Permissions

Lesson 1: Understanding Identity Management in SharePoint 2013	page 2
Lesson 2: Managing Permissions in SharePoint 2013	page 11
Lab A: Managing Permissions Programmatically in SharePoint 2013	page 17
Lesson 3: Configuring Forms-Based Authentication	page 19
Lesson 4: Customizing the Authentication Experience	page 28

Lab B: Creating and Deploying a Customs Claims Provider	page 34
Module 7: Introducing Apps for SharePoint	
Lesson 1: Overview of Apps for SharePoint	page 2
Lesson 2: Developing Apps for SharePoint	page 16
Lab: Creating a Site Suggestions App	page 26
Module 8: Client-Side SharePoint Development	
Lesson 1: Using the Client-Side Object Model for Managed Code	page 2
Lab A: Using the Client-Side Object Model for Managed Code	page 9
Lesson 2: Using the Client-Side Object Model for JavaScript	page 14
Lesson 3: Using the REST API with JavaScript	page 26
Lab B: Using the REST API with JavaScript	page 33
Module 9: Developing Remote-Hosted SharePoint Apps	
Lesson 1: Overview of Remote-Hosted Apps	page 2
Lesson 2: Configuring Remote-Hosted Apps	page 9
Lab A: Configuring a Provider-Hosted SharePoint App	page 15
Lesson 3: Developing Remote-Hosted Apps	page 19
Lab B: Developing a Provider-Hosted SharePoint App	page 25
Module 10: Publishing and Distributing Apps	
Lesson 1: Understanding the App Management Architecture	page 2
Lesson 2: Understanding App Packages	page 5
Lesson 3: Publishing Apps	page 9
Lab A: Publishing an App to a Corporate Catalog	page 13
Lesson 4: Installing, Updating, and Uninstalling Apps	page 16
Lab B: Installing, Updating, and Uninstalling Apps	page 20
Module 11: Automating Business Processes	
Lesson 1: Understanding Workflow in SharePoint 2013	page 2
Lesson 2: Building Workflows by Using Visio 2013 and SharePoint Designer 2013	page 7
Lab A: Building Workflows in Visio 2013 and SharePoint Designer 2013	page 16
Lesson 3: Developing Workflows in Visual Studio 2012	page 20
Lab B: Creating Workflow Actions in Visual Studio 2012	page 25
Module 12: Managing Taxonomy	
Lesson 1: Managing Taxonomy in SharePoint 2013	page 2
Lesson 2: Working with Content Types	page 10
Lab A: Working with Content Types	page 20
Lesson 3: Working with Advanced Features of Content Types	page 24

Lab B: Working with Advanced Features of Content Types	page 30
Module 13: Managing Custom Components and Site Lifecycles	
Lesson 1: Defining Custom Lists	page 2
Lesson 2: Defining Custom Sites	page 7
Lesson 3: Managing SharePoint Sites	page 16
Lab: Managing Custom Components and Site Lifecycles	page 22
Module 14: Customizing User Interface Elements	
Lesson 1: Working with Custom Actions	page 2
Lesson 2: Using Client-Side User Interface Components	page 14
Lab A: Using the Edit Control Block to Launch an App	page 22
Lesson 3: Customizing the SharePoint List User Interface	page 26
Lab B: Using jQuery to Customize the SharePoint List User Interface	page 34
Module 15: Working with Branding and Navigation	
Lesson 1: Creating and Applying Themes	page 2
Lesson 2: Branding and Designing Publishing Sites	page 9
Lab A: Branding and Designing Publishing Sites	page 18
Lesson 3: Tailoring Content to Platforms and Devices	page 24
Lesson 4: Configuring and Customizing Navigation	page 31
Lab B: Configuring Farm-Wide Navigation	page 41
Lab Answer Keys	
Module 1 Lab: Comparing Web Parts and App Parts	page 1
Module 2 Lab A: Working with Sites and Webs	page 1
Module 2 Lab B: Working with Execution Contexts	page 6
Module 3 Lab A: Querying and Retrieving List Data	page 1
Module 3 Lab B: Working with Large Lists	page 5
Module 4 Lab: Working with Features and Solutions	page 1
Module 5 Lab: Working with Server-Side Code	page 1
Module 6 Lab A: Managing Permissions Programmatically in SharePoint 2013	page 1
Module 6 Lab B: Creating and Deploying a Custom Claims Provider	page 4
Module 7 Lab: Creating a Site Suggestions App	page 1
Module 8 Lab A: Using the Client-Side Object Model for Managed Code	page 1
Module 8 Lab B: Using the REST API with JavaScript	page 9
Module 9 Lab A: Configuring a Provider-Hosted SharePoint App	page 1
Module 9 Lab B: Developing a Provider-Hosted SharePoint App	page 5
Module 10 Lab A: Publishing an App to a Corporate Catalog	page 1
Module 10 Lab B: Installing, Updating, and Uninstalling Apps	page 4

Module 11 Lab A: Building Workflows in Visio 2013 and SharePoint Designer 2013	page 1
Module 11 Lab B: Creating Workflow Actions in Visual Studio 2012	page 7
Module 12 Lab A: Working with Content Types	page 1
Module 12 Lab B: Working with Advanced Features of Content Types	page 5
Module 13 Lab: Managing Custom Components and Site Lifecycles	page 1
Module 14 Lab A: Using the Edit Control Block to Launch an App	page 1
Module 14 Lab B: Using jQuery to Customize the SharePoint List User Interface	page 5
Module 15 Lab A: Branding and Designing Publishing Sites	page 1
Module 15 Lab B: Configuring Farm-Wide Navigation	page 7

About This Course

This section provides a brief description of the course, audience, suggested prerequisites, and course objectives.

Course Description

This course teaches students the core skills that are common to almost all SharePoint development activities. These include working with the server-side and client-side object models, developing and deploying features, solutions, and apps, managing identity and permissions, querying and updating list data, managing taxonomy, using workflow to manage business processes, and customizing the user interface.

Audience

This course is intended for professional developers who develop solutions for SharePoint products and technologies in a team-based, medium-sized to large development environment. While some familiarity with SharePoint solution development is required, candidates are not expected to have prior experience with the new features in SharePoint Server 2013.

The ideal candidate is a technical lead with at least four years of SharePoint and web development experience. The candidate is responsible for designing custom code for projects that are deployed to or interact with SharePoint environments. This includes:

- Selecting an appropriate approach and building customizations in SharePoint.
- Creating and implementing a strategy for solution packaging, deployment, and upgrading.
- Identifying SharePoint data and content structures for customizations.
- Performing diagnostics and debugging.
- Planning and designing applications for scalability and performance.
- Identifying and mitigating performance issues of customizations.
- Understanding authentication and authorization.
- Experience with Windows PowerShell.
- Broad familiarity with SharePoint capabilities.
- Familiarity with Online Services such as Azure and SharePoint Online.

Student Prerequisites

This course requires that you meet the following prerequisites:

- A working knowledge of using Visual Studio 2010 or Visual Studio 2012 to develop solutions.
- A basic working knowledge of SharePoint solution development, either in SharePoint 2013 or in earlier versions of SharePoint.
- A working knowledge of Visual C# and the .NET Framework 4.5.
- A basic understanding of ASP.NET and server-side web development technologies, including request/response and the page lifecycle.
- A basic understanding of AJAX and asynchronous programming techniques.
- A basic working knowledge of client-side web technologies including HTML, CSS, and JavaScript.

- Familiarity with approaches to authentication and authorization, including claims-based authentication.

Course Objectives

After completing this course, students will be able to:

- Identify SharePoint development opportunities and make appropriate design choices.
- Work with core server-side objects.
- Query and update list data.
- Design and manage features and solutions.
- Develop code for customer server-side components.
- Manage and customize authentication and authorization.
- Explain the capabilities and design choices for apps for SharePoint.
- Use the client-side object model and the REST API.
- Develop provider-hosted and auto-hosted apps.
- Distribute and deploy apps for SharePoint.
- Create custom workflows to automate business processes.
- Use fields and content types to manage taxonomy.
- Create custom sites and lists and manage the site lifecycle.
- Customize the appearance and behavior of user interface elements.
- Customize navigation and branding.

Course Outline

The course outline is as follows:

Module 1, "SharePoint as a Developer Platform"

Module 2, "Working with SharePoint Objects"

Module 3, "Working with Lists and Libraries"

Module 4, "Designing and Managing Features and Solutions"

Module 5, "Working with Server-Side Code"

Module 6, "Managing Identity and Permissions"

Module 7, "Introducing Apps for SharePoint"

Module 8, "Client-Side SharePoint Development"

Module 9, "Developing Remote-Hosted Apps"

Module 10, "Publishing and Distributing Apps"

Module 11, "Automating Business Processes"

Module 12, "Managing Taxonomy"

Module 13, "Managing Custom Components and Site Lifecycles"

Module 14, "Customizing User Interface Elements"

Module 15, "Working with Branding and Navigation"

Course Materials

The following materials are included with your kit:

- **Course Handbook:** a succinct classroom learning guide that provides the critical technical information in a crisp, tightly-focused format, which is essential for an effective in-class learning experience.
 - **Lessons:** guide you through the learning objectives and provide the key points that are critical to the success of the in-class learning experience.
 - **Labs:** provide a real-world, hands-on platform for you to apply the knowledge and skills learned in the module.
 - **Module Reviews and Takeaways:** provide on-the-job reference material to boost knowledge and skills retention.
 - **Lab Answer Keys:** provide step-by-step lab solution guidance.



Course Companion Content on the <http://www.microsoft.com/learning/companionmoc> Site: searchable, easy-to-browse digital content with integrated premium online resources that supplement the Course Handbook.

- **Modules:** include companion content, such as questions and answers, detailed demo steps and additional reading links, for each lesson. Additionally, they include Lab Review questions and answers and Module Reviews and Takeaways sections, which contain the review questions and answers, best practices, common issues and troubleshooting tips with answers, and real-world issues and scenarios with answers.
- **Resources:** include well-categorized additional resources that give you immediate access to the most current premium content on TechNet, MSDN®, or Microsoft® Press®.
- **Student Course files on the <http://www.microsoft.com/learning/companionmoc> Site:** includes the Allfiles.exe, a self-extracting executable file that contains all required files for the labs and demonstrations.



- **Course evaluation:** at the end of the course, you will have the opportunity to complete an online evaluation to provide feedback on the course, training facility, and instructor.
 - To provide additional comments or feedback on the course, send an email to support@mscourseware.com. To inquire about the Microsoft Certification Program, send an email to mcphelp@microsoft.com.

Virtual Machine Environment

This section provides the information for setting up the classroom environment to support the business scenario of the course.

Virtual Machine Configuration

In this course, you will use Microsoft® Hyper-V™ to perform the labs.

Important: At the end of each lab, you must close the virtual machine and must not save any changes. To close a virtual machine (VM) without saving the changes, perform the following steps:

1. On the virtual machine, on the **Action** menu, click **Close**.
2. In the **Close** dialog box, in the **What do you want the virtual machine to do?** list, click **Turn off** and delete changes, and then click **OK**.

The following table shows the role of each virtual machine that is used in this course:

Virtual machine	Role
20488A-LON-SP-xx (where xx is the module number)	Domain controller and SharePoint development computer

Software Configuration

The following software is installed on each VM:

- SQL Server 2012 SP1, with the following features installed:
 - Database Engine Services
 - Full-Text and Semantic Extractions for Search
 - Management Tools – Complete
- SharePoint Server 2013, Enterprise Edition
- Workflow Manager 1.0
- Office 2013 Professional Plus
- Visio 2013
- SharePoint Designer 2013
- Visual Studio 2012 Premium Edition with Update 1
- Office Developer Tools for Visual Studio 2012

Course Files

The files associated with the labs in this course are located in the E:\Labfiles folder on the student computers.

Classroom Setup

Each classroom computer will have the same virtual machine configured in the same way.

Course Hardware Level

To ensure a satisfactory student experience, Microsoft Learning requires a minimum equipment configuration for trainer and student computers in all Microsoft Certified Partner for Learning Solutions (CPLS) classrooms in which Official Microsoft Learning Product courseware is taught.

Hardware Level 7

- 64 bit Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V) processor (2.8 GHz dual core or better recommended)
- Dual 500 GB hard disks 7200 RPM SATA or faster (striped)
- 16GB RAM
- DVD drive (dual layer recommended)
- Network adapter
- Sound card
- Dual SVGA 17-inch or larger monitors supporting 1440x900 minimum resolution
- Microsoft Mouse or compatible pointing device

In addition, the instructor computer must be connected to a projection display device that supports SVGA 1024 x 768 pixels, 16 bit colors.

Module 1

SharePoint as a Developer Platform

Contents:

Module Overview	1-1
Lesson 1: Introducing the SharePoint Developer Landscape	1-2
Lesson 2: Choosing Approaches to SharePoint Development	1-13
Lesson 3: Understanding SharePoint 2013 Deployment and Execution Models	1-24
Lab: Comparing Web Parts and App Parts	1-30
Module Review and Takeaways	1-36

Module Overview

Developing solutions for Microsoft SharePoint Server 2013 encompasses a broad range of development activities. When you are presented with a specific business requirement relating to SharePoint, there are often several different ways you can approach the problem from a development perspective. In this module, you will gain a broad understanding of SharePoint 2013 as a developer platform. You will learn about the different approaches to development and deployment that are available to you, together with the types of scenarios in which each approach might be appropriate.

Objectives

After completing this module, you will be able to:

- Describe the opportunities for developers in SharePoint Server 2013.
- Choose appropriate execution models for custom SharePoint components.
- Choose appropriate deployment models for custom SharePoint components.

Lesson 1

Introducing the SharePoint Developer Landscape

SharePoint Server 2013 is the fifth major release from Microsoft of its popular collaboration and content management platform. The feature set provided by SharePoint grows and evolves with every new release, and SharePoint Server 2013 is no exception. However, one of the major strengths of the SharePoint platform is the ease with which developers can customize and extend it to meet specific business needs. In this lesson, you will gain a broad understanding of the opportunities and advantages that SharePoint Server 2013 offers developers.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the core workloads for SharePoint Server 2013.
- Describe the developer tools for SharePoint Server 2013.
- Describe the main new features for developers in SharePoint Server 2013.
- Provide a high-level explanation of the SharePoint technology stack.
- Explain how the SharePoint page rendering process works.
- Describe the different ways in which developers can interact with SharePoint 2013 functionality.
- Describe how developers can troubleshoot and debug SharePoint errors.

SharePoint Server 2013 Workloads

SharePoint 2013 provides functionality across several broad functional areas, commonly known as workloads. Although workload names vary, depending on who you talk to, the division of functionality doesn't matter much because many solutions blur the boundaries and traverse multiple workloads. However, discussing SharePoint workloads does serve to illustrate the breadth and capabilities of the platform. As a developer, you can interact with all of the following functional areas of SharePoint:

- Portals and collaboration
- Search
- Enterprise Content Management
- Web Content Management
- Social and communities
- Business Connectivity Services
- Business Intelligence

- *Portals and collaboration.* SharePoint provides many core capabilities, such as the ability to create sites, workspaces, libraries, and lists, which enable teams to work together effectively regardless of their physical location.
- *Search.* SharePoint provides a comprehensive and highly extensible search platform that enables users to find information from a variety of content sources across an organization.
- *Enterprise Content Management.* SharePoint provides a range of functionality to support business processes, such as workflow, records management, and tools for eDiscovery and compliance.
- *Web Content Management.* SharePoint provides comprehensive Content Management System (CMS) features that can be used to manage and maintain a corporate web presence.
- *Social and communities.* SharePoint provides functionality that enables users to connect with one another, share news and information, and build communities around common interests.

- *Business Connectivity Services*. SharePoint enables you to integrate data from external systems into solutions for SharePoint sites and Microsoft Office 2013 applications.
- *Business Intelligence*. SharePoint provides a range of capabilities for providing insight into business data, from built-in tools such as Excel Services and PerformancePoint Services, to integration with SQL Server 2012 Analysis Services and SQL Server 2012 Reporting Services.

Developer Tools for SharePoint Server 2013

SharePoint development is a broad topic that encompasses a disparate range of activities. As a professional SharePoint developer, you are likely to interact with various functional areas of the platform. You are also likely to use a variety of different development tools and techniques to interact with the system. Commonly used development tools include the following:

- *Microsoft Visual Studio 2012*. Visual Studio is an integrated development environment (IDE) that provides a range of productivity tools, debugging and deployment functionality, and integration with Team Foundation Server.
- *Microsoft Office Tools for Visual Studio 2012*. This plug-in for Visual Studio provides templates for SharePoint Server 2013, including apps for SharePoint and apps for Office, as well as server-side custom components.
- *"Napa" Office 365 Development Tools*. These tools enable you to build apps for Office or SharePoint in a browser window, without installing any other tools. You can also download your "Napa" projects and open them in Visual Studio if required.
- *Microsoft SharePoint Designer 2013*. SharePoint Designer is a free desktop application that offers a range of capabilities for customizing sites and creating non-code-based custom components for SharePoint. For example, you can use SharePoint Designer to create workflows and build declarative Business Data Connectivity (BDC) models. Although SharePoint Designer began as a tool for power users, it is now central to many aspects of the SharePoint development process.
- *Web-design tools*. SharePoint 2013 introduces new processes for designing and branding SharePoint sites and pages. These processes enable web designers to create SharePoint pages by using standard web technologies—HTML, CSS, and JavaScript—in standard web design applications. You can then use built-in SharePoint utilities to add the required SharePoint functionality to the designer's wireframe webpages.

- Microsoft Visual Studio 2012
- Microsoft Office Tools for Visual Studio 2012
- "Napa" Office 365 Development Tools
- Microsoft SharePoint Designer 2013
- Web-design tools

This list is not exhaustive. There are additional tools that you are likely to encounter when you work with SharePoint, such as Windows PowerShell and web traffic debuggers. You will learn more about all of these tools throughout this course.

What's New for Developers in SharePoint 2013

SharePoint Server 2013 includes various new features that are particularly relevant to developers.

The apps for SharePoint model

One of the major changes in SharePoint Server 2013 is the introduction of apps for SharePoint and apps for Office. Apps provide an entirely new model for developing, packaging, and deploying custom functionality for SharePoint. Essentially, apps run outside of the SharePoint server—by running on Windows Azure or a remote server platform, or by executing JavaScript in the client browser, or by a combination of the two—and interact with SharePoint either through the client-side object model (CSOM) or the Representational State Transfer (REST) API. Apps do not deploy or run any server-side code on the SharePoint server.

Apps offer the following advantages to administrators, developers, and end-users over more traditional approaches to SharePoint development:

- *Lower impact on farm performance.* Because apps do not execute any code on SharePoint servers, administrators can worry less about customizations consuming excessive server resources.
- *Better encapsulation of functionality.* When a user installs an app, the app provisions a subsite (the app web) at the installation location. This app web contains any app resources that are stored on the SharePoint site, such as pages, lists, content types, site columns, and so on. This means there is little chance of an app inadvertently interfering with any other content on a SharePoint site. It also helps to ensure that all resources are removed cleanly when the user uninstalls an app.
- *Better management of permissions.* An app must specify all the permissions it requires in order to operate at the point of installation. A user can either choose to grant all permissions or cancel the installation.
- *Consistency across platforms.* Apps work in exactly the same way on both SharePoint Online and on-premises SharePoint installations. There are no differences in capabilities.
- *Greater reach.* In addition to creating apps for use within an organization, developers can target a broader audience by submitting apps to the Microsoft Office Marketplace.



Note: You will learn more about apps for SharePoint throughout this course.

New client-side programming models

SharePoint 2013 includes new and enhanced models for client-side solution developers. You can choose from the following range of client-side APIs, depending on the platform and requirements of your client-side solution:

- *JavaScript object model.* The JavaScript object model, also known as the SharePoint 2013 client-side object model for JavaScript, was introduced in SharePoint 2010. The JavaScript object model includes several enhancements in SharePoint 2013. Most notably, the scope of the API is much larger than in SharePoint 2010. In SharePoint 2013, you can also use the JavaScript object model to access SharePoint data across domain boundaries.
- *.NET Framework object model.* The .NET Framework object model, also known as the SharePoint 2013 object model for managed clients, was also introduced in SharePoint 2010. The .NET Framework

- The apps for SharePoint model
- New client-side programming models
 - JavaScript
 - .NET Framework client
 - Silverlight/Mobile
 - REST/OData endpoints
- New design model
- New workflow model
- Other key enhancements

object model for SharePoint 2013 includes several new capabilities, including the ability to interact with the core SharePoint Server workloads.

- *Silverlight object model and Mobile object model.* The Silverlight object model was introduced in SharePoint 2010 as a tool primarily for developing Silverlight-based Web Parts. In SharePoint 2013, the Silverlight object model is more typically used as a tool for developing apps for Windows Phone devices. The Mobile object model is a version of the Silverlight object model that is specifically designed for Windows Phone devices. It includes additional functionality that is specific to mobile devices, such as the ability to subscribe to push notifications.
- *REST/OData endpoints.* SharePoint 2010 introduced a REST API that enabled developers to perform create, retrieve, update, delete, and query (CRUDQ) operations on SharePoint lists. SharePoint 2013 includes REST/OData endpoints that can be used to perform a much broader set of operations.



Note: The Open Data Protocol (OData) is an open protocol for querying and updating data over the web. REST describes a general architectural pattern for providing and consuming data using HTTP requests. The OData protocol is essentially an implementation of the REST pattern.

For more information about OData, see *What is the Open Data Protocol?* at <http://www.odata.org>.

Although client-side object models were available in SharePoint 2010, the SharePoint 2013 versions support a much broader range of functionality. All client-side programming models now enable you to work with user profiles, the search service, and most of the core server workload features from client-side code, instead of using .asmx services.



Note: Client-side programming models for SharePoint 2013 are covered in greater detail later in this course.

New design model

Creating publishing assets for a SharePoint site, such as master pages and page layouts, has historically been a challenging area. Web designers often lack the SharePoint-specific knowledge required to create these assets, whereas SharePoint developers may not be especially skilled in general web design. SharePoint 2013 introduces a new design model in which pages are entirely based on core web technologies such as HTML, CSS, and JavaScript. For example, you can now create HTML-based master pages for SharePoint using industry-standard design tools. You can then use the Design Manager utility in SharePoint publishing sites to generate HTML snippets for various SharePoint-specific controls, such as navigation menus and site titles.

New workflow model

SharePoint 2013 includes an entirely new workflow model based on Workflow Manager 1.0. Workflow Manager, which is a new server product that is built on .NET Framework 4.5 and Windows Workflow Foundation (WF45), provides a highly-scalable platform for on-premises or Windows Azure-based workflow farms. Workflow development for SharePoint 2013 is entirely declarative. This, in turn, gives you far more flexibility over how you deploy your workflows. Instead of deploying farm-scoped workflows like in SharePoint 2010, you deploy SharePoint 2013 to specific sites using apps for SharePoint or sandboxed solutions.



Note: The SharePoint 2010 workflow platform is still available in SharePoint 2013. This provides backward compatibility for any existing custom workflow components, and it enables

you to develop new SharePoint 2010 workflow components if you require the specific functionality that it provides.

Other key enhancements

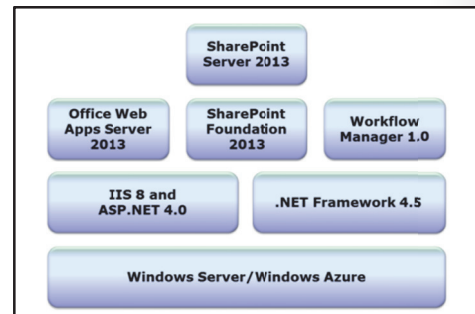
SharePoint 2013 includes new or enhanced functionality in many of the core workloads. The search service has been completely redesigned to incorporate key elements of FAST search functionality, and social computing, web content management, and the mobile user experience were all major areas of investment.



Additional Reading: For more information about new features in SharePoint 2013, see *What's new for developers in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=306774>.

The SharePoint 2013 Technology Stack

SharePoint Server 2013 builds on a variety of products and technologies. Understanding how these products and technologies fit together can be helpful when you design and build solutions for the SharePoint platform.



SharePoint, ASP.NET, and IIS

Like previous versions of SharePoint, SharePoint Server 2013 is an ASP.NET web application.

SharePoint requires Internet Information Services (IIS) and ASP.NET 4 to run in integrated mode, where IIS and ASP.NET are essentially a single platform from the perspective of web applications.

SharePoint adds custom modules and handlers to the IIS/ASP.NET integrated pipeline to implement the SharePoint page rendering process, which is described in the next topic.

.NET Framework 4.5

SharePoint 2013 depends on various features of .NET Framework 4.5. In particular, the SharePoint claims infrastructure relies on Windows Identity Foundation to manage claims-based Windows identities. SharePoint 2013 workflows rely on Windows Workflow Foundation (WF45).

Office Web Apps Server 2013

In SharePoint 2010, Office Web Apps ran as a SharePoint service application. In SharePoint 2013, Office Web Apps Server is a separate server product that must run on separate hardware. This enables architects and administrators to scale an Office Web Apps server farm and a SharePoint server farm independently.

Workflow Manager 1.0

The SharePoint workflow infrastructure is now external to the SharePoint logical architecture. SharePoint workflows now run on an independent Workflow Manager infrastructure. You can install Workflow Manager on the same hardware as SharePoint Server 2013, which is useful for developer workstations and smaller SharePoint deployments. As deployments grow larger, architects and administrators will often scale out Workflow Manager to run on separate hardware.

SharePoint Foundation 2013

SharePoint Foundation 2013 represents the core set of SharePoint functionality. This includes key functional components such as the SharePoint service application architecture, the page rendering

components, and the claims infrastructure, together with the ability to perform core tasks such as creating sites, managing content, and managing users.

SharePoint Server 2013

SharePoint Server 2013 builds on SharePoint Foundation 2013 to provide more advanced functionality in each of the key workload areas.

The SharePoint Page Rendering Process

In SharePoint, the parsing and page rendering process underpins almost all aspects of functionality and user experience. It is essential to gain a solid understanding of this process before you begin developing customizations for the SharePoint platform.

In terms of parsing and rendering, there are two distinct types of pages in SharePoint: application pages and content pages.

- Application pages
 - Physical page on file system
- Content pages
 - Virtual page in content database
- Customized content pages
 - Ghosting and unghosting
 - Safe mode parser

Application pages

An application page is an ASPX page that resides on the file system of SharePoint Web Front End (WFE) servers. These pages are deployed to the SharePoint file structure, which is often referred to as the SharePoint root, the SharePoint hive, or the 15 hive.



Note: The SharePoint root is located at `%COMMONPROGRAMFILES%\Microsoft Shared\Web Server Extensions\15`. The **Web Server Extensions** folder also contains a folder named **14**, which contains resources for rendering sites and components in SharePoint 2010 mode.

Application pages are parsed and rendered by the ASP.NET parser as standard ASPX pages. Application pages are used to encapsulate functionality rather than content. For example, the administration pages and dialog boxes that are used to manage a SharePoint site are all application pages.

SharePoint makes application pages available to every site through virtual directories. For example, the site-relative URL token `_layouts` points to the physical folder at `15\TEMPLATE\LAYOUTS`. Suppose you created site collections at `site1.contoso.com` and `site2.contoso.com`. Requests for `site1.contoso.com/_layouts/MyApplicationPage.aspx` and `site2.contoso.com/_layouts/MyApplicationPage.aspx` would both render the physical page at `15\TEMPLATE\LAYOUTS\MyApplicationPage.aspx`.

Content pages

A content page is an ASPX page that is dynamically constructed when the page is requested. Content pages are formed by combining a page template from the file system with page content from the database. The page template effectively consists of placeholders, such as Web Part zones, into which content and controls are injected when the page is constructed.

Because a SharePoint deployment may include tens of thousands of rapidly changing pages, managing these pages on the server file system and keeping them synchronized across every web server would be an intensive process. For this reason, the vast majority of pages in SharePoint are content pages. When an uncustomized content page has been requested and dynamically constructed, it is parsed and rendered by the ASP.NET parser.

Ghosting and unghosting

Uncustomized content pages are sometimes referred to as ghosted pages. This reflects the fact that the page does not really exist as a single entity—it's a combination of a physical page template and a set of content from the content database.

Administrators and power users can use SharePoint Designer to customize content pages within a site collection. If this customization includes changes to the physical page template, the page becomes unghosted. This means that the page can no longer be created dynamically by merging a physical page template from the site definition with content from the database. Instead, the entire ASPX page is stored in the content database and retrieved when requested.

Customized (or unghosted) content pages are parsed and rendered by the SharePoint safe mode parser, rather than the ASP.NET parser. The safe mode parser includes various restrictions to prevent malicious users from using customized pages to launch attacks. In particular, the safe mode parser blocks the processing of any pages that contain inline script.

Discussion: Page Rendering in SharePoint

The instructor will now lead a brief discussion around the following questions:

- Is a Web Part page a content page or an application page?
- Why might you want to deploy a custom application page to a SharePoint environment?

- Is a Web Part page a content page or an application page?
- Why might you want to deploy a custom application page to a SharePoint environment?

Entry Points for Developers in SharePoint 2013

SharePoint exposes functionality to developers in a variety of ways. This topic provides a high-level overview of the entry points for developers in SharePoint 2013.

Server-side object model

If you have worked with previous versions of SharePoint, you are likely to have worked with the server-side object model, which is primarily contained within the **Microsoft.SharePoint.dll** assembly. The server-side object model enables you to interact with almost every element of SharePoint functionality by writing managed code that runs on the SharePoint server itself. The various ways of deploying and executing server-side SharePoint code are covered in the lessons that follow.

In addition to using the server-side object model in managed code, you can also interact with the server-side object model from Windows PowerShell. This is a useful way of running code on an ad hoc basis, or for developing scripts to perform repetitive tasks.

- Server-side object model
 - Managed code
 - Windows PowerShell
- Client.svc
 - REST/OData clients
 - Client-side object models
- Declarative customizations

Client.svc

SharePoint 2013 exposes a Windows Communication Foundation (WCF) service named `client.svc` on every site collection. This service is central to all client-side programming in SharePoint 2013.

When you use the REST/OData approach to client-side development, your code interacts directly with the `client.svc` service. You construct and submit an HTTP request, using an appropriate verb, and the service responds in accordance with the OData protocol. The client-side object models—the JavaScript object model, the managed client object model, and the Silverlight/mobile object models—essentially provide a client-side proxy for the `client.svc` service. When you use one of the client-side object models to submit a request to SharePoint, the client-side proxy constructs and submits the request and then parses the response.



Note: SharePoint 2013 exposes the `client.svc` service on each site collection at the site-relative URL `_vti_bin/client.svc`. You can also use the shorthand alias `_api` to access the service. For example, if you created a site collection at the URL `team.contoso.com`, you could access the `client.svc` service at `team.contoso.com/_vti_bin/service` or `team.contoso.com/_api`.

Declarative customizations

You can customize many aspects of a SharePoint deployment without writing any executable code. Most custom components can be provisioned and configured by creating and deploying Collaborative Application Markup Language (CAML) files to the server. CAML is an XML-based language that describes many aspects of SharePoint configuration. For example, you can create site columns, lists, content types, and many user interface elements entirely in CAML.



Best Practice: In many scenarios, you will be able to choose whether to configure SharePoint components declaratively (by creating CAML files) or imperatively (by writing code). In most cases, you should use the declarative approach wherever possible. Declarative artifacts are typically easier to read and easier to maintain.

Demonstration: Developer Tools for SharePoint 2013

Demonstration Steps

- Start the 20488B-LON-SP-01 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, type **Visual Studio 2012**, and then press Enter.
- On the Start page, click **New Project**.
- In the **New Project** dialog box, in the left navigation pane, under **Templates**, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
- Notice that there are project templates for both SharePoint 2010 and SharePoint 2013. The SharePoint 2013 templates are provided by Microsoft Office Developer Tools for Visual Studio 2012.
- In the left navigation pane, under **Office/SharePoint**, click **Apps**.
- In the center pane, click **App for SharePoint 2013**.
- In the **Name** box, type **Demo1**.
- In the **Location** box, type **E:\DemoFiles**, and then click **OK**.

- In the **New app for SharePoint** dialog box, under **What is the name of your app for SharePoint**, type **Demo**.
- Under **What SharePoint site do you want to use for debugging your app**, type **http://dev.contoso.com**.
- Under **How do you want to host your app for SharePoint**, review the available options, and then select **SharePoint-hosted**.
- Click **Finish**.
- Briefly review the contents of the project.
- In Solution Explorer, right-click **Demo1**, point to **Add**, and then click **New Item**.
- In the **Add New Item - Demo1** dialog box, review the available project item templates, and then click **Cancel**.
- Close Visual Studio.
- On the Start screen, type **SharePoint Designer**, and then press Enter.
- Click **Open Site**.
- In the **Open Site** dialog box, in the **Site name** box, type **http://team.contoso.com**, and then click **Open**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- When the **Open Site** dialog box displays the contents of the team.contoso.com site, click **Open** again.
- On the ribbon, in the **New** group, briefly explore the different items you can add to the team site.
- On the ribbon, in the **Actions** group, notice that you can perform core administration tasks, such as resetting the site to its site definition and managing users and groups.
- On the ribbon, in the **Manage** group, click the **Preview in Browser** drop-down menu (be sure to click the drop-down menu, not the icon).
- On the **Preview in Browser** menu, notice that you can preview the site at various resolutions and in various browsers.
- On the **Preview in Browser** menu, click **Edit Browser List**, and notice that you can add additional browsers to the preview list.
- Click **Cancel**, and then close SharePoint Designer.

Troubleshooting and Debugging SharePoint Solutions

As a developer, you will sometimes need to troubleshoot issues and debug your code. There are techniques and processes that you can use to simplify your troubleshooting and debugging when developing SharePoint solutions in Visual Studio.

Accessing error information

By default, ASP.NET applications display custom errors. This means that when an error occurs in your SharePoint application, the browser may display a generic error message informing you that an unexpected error has occurred, without providing information about the error or its location. If you want to receive more informative error details, you can modify the default Web.config file.

You will find the Web.config file in the **C:\inetpub\wwwroot\wss\VirtualDirectories\[port number]** folder. The file contains a **customErrors** element that has a **mode** attribute. The default value of this attribute is **On**, but if you change the value to either **RemoteOnly** or **Off**, detailed errors are shown in the browser.

You can also include a full stack trace in the error page by modifying the **SafeMode** element in the Web.config file. This element contains a **CallStack** attribute with a default value of **false**. If you change this value to **true**, and if custom errors are disabled, the browser shows the call stack alongside the error details.

Although changing these settings in the development environment can help you locate and resolve errors, you should not modify them in the production environment because of the overheads which they incur.

Accessing event logs and trace logs

Another useful source of information during the debugging process is log files. SharePoint writes information to the Windows event logs and also to trace files:

- *Event logs.* Like with many applications, SharePoint writes event information to the Windows Application log, which you can access by using the Windows Event Viewer.
- *Trace logs.* SharePoint also writes trace logs to files in the C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\LOGS folder. These files can become large, making it difficult to locate information about your specific error. However, you can copy the correlation ID that the browser displays and then search for this GUID in the trace log file to locate details about the error.

By default, SharePoint logs a lot of information in the event log and trace logs. This can impact both the server performance and the disk usage. To reduce this impact, you can configure the event severity to log either globally or by category by using the SharePoint Central Administration website or a Windows PowerShell script.



Additional Reading: For more information about configuring diagnostic logging, see *Configure diagnostic logging in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=320123>.



Note: The virtual machines that you will use for the labs during this course have logging heavily throttled to improve their performance.

- Modify Web.config to access detailed error information:
 - **customErrors** element
 - **SafeMode** element
- Use logs to access further information:
 - Windows Application Event Log
 - Trace logs
- Attach the Visual Studio debugger to debug code

Using the Visual Studio debugger

Most server-side SharePoint code runs in the IIS worker process, `w3wp.exe`. When you run a SharePoint project in Visual Studio debug mode, the deployment process automatically attaches the Visual Studio debugger to this process. However, you can also debug projects by manually attaching the debugger to the `w3wp.exe` process by using the **Attach to Process** menu command in Visual Studio.

Some SharePoint code runs in other processes, for example, timer jobs run in the `owstimer.exe` process. If you need to debug code in an alternate process, you can select that process when manually attaching the debugger to your code.



Additional Reading: For more information about using the Visual Studio debugger for SharePoint solutions, see *Debugging SharePoint Solutions* at <http://go.microsoft.com/fwlink/?LinkID=318027>.

Lesson 2

Choosing Approaches to SharePoint Development

When you start a SharePoint development project, there are often several different approaches to development that you could take to meet the project's business requirements. In this lesson, you will gain an understanding of the development options that are available in SharePoint 2013, together with the capabilities and limitations of each approach.

Lesson Objectives

After completing this lesson, you will be able to describe the capabilities and limitations of the following approaches to SharePoint development:

- Declarative components
- Client-side SharePoint code
- Web Parts
- Application pages
- Timer jobs
- Event receivers
- Workflows

Declarative Components

With SharePoint, you can define and configure a wide range of custom components using CAML. For example, you can use CAML to create and configure the following elements:

- *Field definitions.* Use these to define new site columns.
- *Content type definitions.* Use these to define new site content types.
- *List templates.* Use these to enable users to create new instances of specified list definitions.
- *List instances.* Use these to provision an instance of a list template.
- *Content type bindings.* Use these to associate an existing content type with an existing list.
- *Modules.* Use these to deploy files, together with any associated metadata, to a SharePoint site.
- *Event registrations.* Use these to register an event receiver assembly as a listener for specific SharePoint events.
- *Custom actions.* Use these to add menu items to the ribbon, the Edit Control Block (ECB), and various other areas of the UI.
- *Workflow definitions.* Use these to define a custom SharePoint workflow.

- Use declarative components to deploy:
 - Site columns
 - Content types and content type bindings
 - List templates and list instances
 - Event registrations and custom actions
 - Workflows, files, and more
- When should you use declarative components?
 - Whenever you can
- Where can you use declarative components?
 - SharePoint Online
 - On-premises deployments

The following code example shows how to use CAML to define a "use by" date site column:

Defining a Site Column Declaratively

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field
    ID="{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"
    Name="UseBy"
    DisplayName="Use By"
    Type="DateTime"
    Format="DateOnly"
    Required="FALSE"
    Group="Contoso Site Columns">
  </Field>
</Elements>
```

Declarative components are always defined within an **Elements** element, and the files containing these components are referred to as element manifests. Element manifests are covered in greater detail in the "SharePoint Features" topic in the next lesson. You will also learn more about defining specific custom components in CAML throughout this course.

When should you use declarative components?

Generally speaking, you should use declarative components if they allow you to meet the requirements of your project. Declarative components are easier to read than their imperative equivalents. Because declarative components do not execute any code, there are fewer limitations on where and how you deploy your customizations.

One of the disadvantages of declarative components has traditionally been that developers often found the unfamiliar CAML syntax hard to master. Fortunately, the CAML developer experience has improved. In many cases, Visual Studio 2012 will provide a basic CAML structure as a starting point when you create a new custom component. You can get IntelliSense support for editing CAML files in Visual Studio by adding references to the SharePoint 2013 schema files, such as WSS.xsd. There are also various third-party and open source tools that provide enhanced support for CAML authoring.



Note: In SharePoint 2013, XML schema files are stored in the 15\TEMPLATE\XML folder. Visual Studio 2012 automatically adds references to common SharePoint XML schemas when you create a declarative component.

Where can you use declarative components?

Declarative components are the most flexible of all approaches to building custom SharePoint functionality. You can deploy declarative components in SharePoint Features within solutions or apps, and you can use declarative components with both SharePoint Online and on-premises SharePoint 2013 deployments. However, declarative components are always deployed to SharePoint, so you cannot use declarative components if you want to create a purely cloud-hosted app.

Client-Side Code

Developing client-side code that interacts with SharePoint 2013 can encompass a wide range of activities. Typically, client-side code is used to develop apps for SharePoint or apps for Office. However, you can use client-side code in a variety of other scenarios, such as Windows Phone apps, custom web services, or desktop applications. You also have a choice of API sets, as described in the previous lesson. For example, you can use the JavaScript object model, the managed client object model, or the Silverlight/Windows Phone object model, or you can interact with the client.svc service directly by using REST and OData. In each case, the client.svc service makes calls to the server-side object model on behalf of the client and then returns the result to the client as a response.

- Use client-side code to:
 - Interact with core SharePoint artifacts and functionality
 - Interact with SharePoint Server workloads
 - Perform almost any operations within the scope of a site collection
- When should you use client-side code?
 - It is the preferred approach when you need to programmatically interact with a SharePoint site collection
- Where can you use client-side code?
 - SharePoint Online
 - On-premises deployments

The capabilities and limitations of client-side SharePoint code are essentially the same, regardless of your platform and your choice of object model. You are restricted only by the capabilities and limitations of the client.svc service. For example, you can use client-side code to:

- Interact with sites, lists, document libraries, content types, and files.
- Manage administrative settings for site collections and sites.
- Interact with user profiles and social features.
- Interact with managed metadata term sets.
- Interact with the search service.
- Interact with Business Data Connectivity components.
- Interact with workflows.

From a functionality perspective, the primary limitation of client-side code for SharePoint is that your code is limited in scope to a site collection. For example, you cannot aggregate or copy data across site collection boundaries, and you cannot interact with components of the SharePoint logical architecture at the web application or farm level.

When should you use client-side code?

Client-side development is now the recommended approach when you need to programmatically interact with a SharePoint 2013 deployment. Client-side development offers several advantages over traditional server-side development approaches. First, client-side development does not require you to deploy and run any code on SharePoint servers, which means that you are less likely to encounter administrative restrictions when you deploy your custom components. Client-side code also works in the same way for SharePoint Online as it does for on-premises SharePoint deployments.



Note: Apps for SharePoint and apps for Office are the most common way (but not the only way) of deploying and executing client-side code. Apps include several additional advantages that are not discussed here, such as encapsulation of functionality and a consistent deployment experience. These advantages are discussed in the next lesson, *Understanding SharePoint 2013 Deployment and Execution Models*.

Generally speaking, you should consider whether you can accomplish your project goals using client-side code before you explore alternative options based on server-side code.

Where can you use client-side code?

You can use client-side code to interact with both SharePoint Online and on-premises deployments of SharePoint in exactly the same way.

Web Parts

You have probably worked with Web Parts if you have developed solutions for previous versions of SharePoint. A Web Part is a type of ASP.NET control, and SharePoint Web Parts build on the ASP.NET Web Part framework. This framework is designed to enable users to modify the content and behavior of webpages by adding custom controls (the Web Parts) to specific regions on the page (Web Part zones). The Web Part framework also enables you to connect Web Parts; for example, you can use the framework to configure one Web Part to consume data from another Web Part in order to present a master/detail view. SharePoint itself relies heavily on Web Parts to provide much of its built-in functionality.

Visual Studio 2012 includes templates for creating three types of Web Parts: regular Web Parts, visual Web Parts, and Silverlight Web Parts. In a regular Web Part, you control how your Web Part renders by overriding base class methods. In a visual Web Part, the template includes an ASP.NET user control (an .ascx file) that can use you to build a user interface declaratively or by using the Visual Studio designer. In a Silverlight Web Part, the Web Part hosts a custom Silverlight control.

In most cases, your Web Parts will inherit from the **System.Web.UI.WebControls.WebParts.WebPart** class. This class provides various life cycle events that integrate with the ASP.NET page life cycle of the page that contains the Web Part. For example, the **WebPart** class exposes **Init**, **Load**, **PreRender**, and **Unload** events that you can handle like you would in an ASPX code-behind file. The **WebPart** class also includes various abstract methods, such as **CreateChildControls** and **RenderContents**, which you can override to implement the functionality of your Web Part.

- Use Web Parts to:
 - Create custom functionality with user interaction
 - Connect to other Web Parts
- When should you create Web Parts?
 - Consider apps first
 - Use Web Parts when you specifically want to leverage the Web Part framework
- Where can you use Web Parts?
 - SharePoint Online (sandboxed solutions only)
 - On-premises deployments (farm and sandboxed solutions)

The following code shows the basic structure of a regular SharePoint Web Part:

Creating a SharePoint Web Part

```
public class ContosoWebPart : WebPart
{
    Label lblGreeting;

    protected override void CreateChildControls()
    {
        //Create your controls within this method.
        lblGreeting = new Label();
        lblGreeting.Text = "Hello, World!"
        this.Controls.Add(lblGreeting);
    }

    protected override void RenderContents(HtmlTextWriter writer)
    {
        // Define your rendering logic within this method.
        writer.Write("<div id='contosowebpart'>");
        lblGreeting.RenderControl(writer);
        writer.Write("</div>");
    }
}
```

In addition to deploying the assembly that contains your Web Part code, you must also deploy a Web Parts control description file to the Web Part Gallery. The Web Parts control description file is a CAML-based XML file with a .webpart file name extension that specifies how the Web Part should appear in the Web Part Gallery when deployed. The file also indicates where the Web Part page can find the control type and assembly. You can also use the Web Parts control description file to define various properties, such as the title of the Web Part, the description, icon URLs, and customization settings.

In terms of capabilities, what you can do with a Web Part depends on which deployment model you choose. If you deploy a Web Part within a farm solution, your code has full access to the server-side object model and, as such, access to the full range of SharePoint 2013 capabilities. You can also deploy a Web Part within a sandboxed solution; in which case, your code is subject to the restrictions of sandboxed solution development.



Note: Farm solutions and sandboxed solutions are described in more detail in the next lesson. Web Part development is described in more detail later in this course.

When should you create Web Parts?

In previous versions of SharePoint, Web Parts were the primary approach to providing custom functionality that required user interaction. In SharePoint 2013, you should use apps instead, at least wherever possible. Apps provide a more consistent installation, deployment, and life cycle management experience.

However, there are still scenarios in which you might want to create Web Parts. This is most likely when you want to take advantage of Web Part connections; for example, to provide data to other Web Parts, consume data from other Web Parts, or provide filters to other Web Parts.

Where can you use Web Parts?

You can deploy a Web Part in a sandboxed solution to both SharePoint Online subscriptions and on-premises SharePoint installations. You cannot deploy farm solutions to SharePoint Online, because the unrestricted code could jeopardize the performance and stability of the platform as a whole.

Application Pages

SharePoint uses application pages extensively to expose site management and administrative capabilities. Application pages are used to expose functionality that is not specific to sites or content.

Creating a custom application page is one of the easier approaches to SharePoint development. You simply create an ASP.NET forms (ASPX) page and deploy it to the 15\TEMPLATE\LAYOUTS folder (or a subfolder within) on each SharePoint WFE server. Your page is then available on every site through the **_layouts** virtual directory. For example, if you created and deployed an application page named `MyApplicationPage.aspx`:

- Users of a Finance site collection could access your page at `http://finance.contoso.com/_layouts/MyApplicationPage.aspx`.
- Users of a Sales site collection could access your page at `http://sales.contoso.com/_layouts/MyApplicationPage.aspx`.
- Users of a regional sales sub-site could access your page at `http://sales.contoso.com/regions/EMEA/_layouts/MyApplicationPage.aspx`.

In terms of capabilities, application pages run with full trust and have full access to the server-side object model. As such, there is little that you cannot do with a custom application page. Typically, application pages are used when you want to expose the same functionality to every site in a SharePoint server farm.

The **Microsoft.SharePoint.WebControls** namespace includes two base classes that you can use as the foundation for your custom application pages:

- *LayoutsPageBase*. Use this class when you want to create an application page that only privileged users can access.
- *UnsecuredLayoutsPageBase*. Use this class when you want to create an application page that anyone (even unauthenticated users) can access.

When should you create custom application pages?

Before you create a custom application page, carefully consider whether it is really necessary. Custom application pages must be deployed in a full-trust farm solution, which limits the environments to which you can deploy your customization. The server-side code runs with full trust, which can jeopardize the performance and stability of the platform as a whole. The solution must be deployed by a farm administrator, or a local server administrator. Administrators are often understandably nervous about solutions that require access to the server-side file system. Developers and solution architects should be cautious about creating code that runs in the same application pool as the SharePoint web application itself.

Typically, developers create custom application pages when they want to expose the same functionality to every SharePoint site. In many cases, you could achieve the same aims by creating an app and making it available across the organization through App Catalog sites. Apps have many advantages over custom application pages—they can be deployed to any environment, they are easier to manage, and they pose less of a performance and stability risk to the host environment.

Generally speaking, the only justification for creating a custom application page would be if you need capabilities that are unavailable in apps. This would typically occur when you want to interact with resources at the web application scope or the farm scope within a SharePoint deployment.

- Use custom application pages to:
 - Expose functionality to every site in a SharePoint farm
- When should you create a custom application page?
 - When there are no other options
 - Consider apps first
- Where can you use custom application pages?
 - On-premises deployments (subject to policy and administrative approval)
 - Not available for SharePoint Online

Where can you use custom application pages?

Custom application pages must be deployed in a farm solution. As a result, you cannot deploy a custom application page to a SharePoint Online subscription. In addition, you can only deploy a custom application page to an on-premises SharePoint installation if your governance policy permits farm solutions, and if a farm administrator approves the deployment.

Timer Jobs

SharePoint timer jobs are used to run background tasks that do not require user interaction on a scheduled basis. SharePoint 2013 uses built-in timer jobs extensively for a wide range of tasks, such as scheduled approval and publishing, processing health and diagnostic data, upgrading site collections, and applying expiration policies.

Custom timer jobs can be useful when you want to perform long-running or resource-intensive tasks without user interaction and without returning any information to a user. For example, you might use a timer job to aggregate and process list or site data across site collection boundaries. Timer jobs run in a separate process—the SharePoint timer service, or `owstimer.exe`—rather than the IIS worker process, `w3wp.exe`. This means that timer jobs should not affect the performance and stability of individual IIS application pools. It also means that you can run long-running processes without the risk of the process being interrupted by an application pool recycle.

SharePoint 2013 supports two types of timer jobs:

- *Regular timer job.* You can use a regular timer job to perform specific tasks on a scheduled basis. Administrators can configure the schedule on which your timer job runs. To create a regular timer job, you create a class that inherits from the **SPJobDefinition** class.
- *Work item timer job.* You can use a work item timer job to queue items to be processed on a scheduled basis. For example, suppose that you enable certain users to request the creation of a specialized site collection by creating a list item containing the required metadata, settings, and so on. You might want to add these requests to a queue and then process the requests to create the sites automatically. To create a work item timer job, you create a class that inherits from the **SPWorkItemJobDefinition** class. You must also create a class that inherits from **SPWorkItem** to represent your individual tasks.

In terms of capabilities, timer jobs run with full trust and have full access to the server-side object model.



Note: Timer job development is covered in more detail later in this course.

When should you create custom timer jobs?

You should consider creating a timer job when:

- You want to execute long-running or resource-intensive processes.
- You want to perform tasks that do not require user input.
- You want to process tasks asynchronously without returning information to the user.

- Use custom timer jobs to:
 - Run background tasks on a scheduled basis
 - Process queues of work items on a scheduled basis
- When should you create a custom timer job?
 - When you do not require user interaction
 - When you want to remove logic from the page load process
- Where can you use custom timer jobs?
 - On-premises deployments (subject to policy and administrative approval)
 - Not available for SharePoint Online

Timer jobs can often present a useful alternative to embedding logic in Web Parts. For example, suppose you create a Web Part that aggregates list data from across a site collection. The aggregation logic will run every time a user loads the page that hosts the Web Part. This is inefficient, because the Web Part ends up processing the same data repeatedly. One alternative would be to create a timer job that aggregates the data to a new list on a regular basis. This reduces the amount of repetitive processing and removes the logic from the page load process.

Where can you use custom timer jobs?

Timer jobs must be deployed in a farm solution; therefore, you cannot deploy a timer job to a SharePoint Online subscription. In addition, you can only deploy a timer job to an on-premises SharePoint installation if your governance policy permits farm solutions, and if a farm administrator approves the deployment.

Event Receivers

You can use event receivers to create code that performs some action in response to an event in SharePoint. SharePoint 2013 exposes a wide variety of events. You can create event receivers for the following categories of events in SharePoint.

- Use event receivers to:
 - Execute code before or after an action occurs
- When should you create an event receiver?
 - When you want to perform custom validation or additional configuration for an action
 - When you want to perform additional processing after an action occurs
- Where can you use event receivers?
 - SharePoint Online
 - On-premises deployments

Event category	Example
Site collection events	A site collection is deleted.
Web events	A site is provisioned, moved, or deleted.
List events	A list is created or deleted.
Field events	A field is added, deleted, or updated.
Item events	An item is created, modified, moved, deleted, checked in, or checked out.
Workflow events	A workflow is started, completed, or postponed.
Feature events	An app is installed, upgraded, or uninstalled.

There are two types of events in SharePoint:

- *Before events.* A before event is fired immediately before an action happens, which allows the event receiver to cancel the action if required.
- *After events.* An after event is fired immediately after an action happens, which allows the event receiver to perform any additional actions required in response to the action.

Before events and after events are distinguished by the form of the verb in the event name. For example, the **SiteDeleting** event occurs immediately before a site collection is deleted, whereas the **SiteDeleted** event occurs immediately after. Event receivers for before events are always processed synchronously, whereas event receivers for after events can be processed either synchronously or asynchronously.

SharePoint provides base classes that you can use to develop event receivers for each category of event. For example, if you want to handle list events, you develop a class that derives from **SPListEventReceiver**. If you want to handle item events, you develop a class that derives from **SPIItemEventReceiver**. You can then use a declarative component to associate your event receiver with particular sites, lists, content types, or other SharePoint artifacts.

The following code shows a simple example of an event receiver class that handles the **ItemDeleting** event:

Creating an Event Receiver

```
public class ContosoEventReceiver : SPIItemEventReceiver
{
    public override void ItemDeleting(SPIItemEventProperties properties)
    {
        properties.ErrorMessage = "You do not want to delete this item";
        properties.Status = SPEventReceiverStatus.CancelWithError;
    }
}
```

You can create remote event receivers when you develop apps for SharePoint. Remote event receivers enable remote-hosted apps to respond to events that occur within the SharePoint site associated with the app.

When should you create event receivers?

Generally speaking, you should create an event receiver when you want to respond to a specific change in a SharePoint environment, as opposed to a specific user action. Event receivers are useful in a wide range of scenarios. For example, you should consider creating event receivers when:

- You want to perform custom validation before an action occurs. For example, you can programmatically verify that a site is not in regular use before you allow a site deletion to proceed.
- You want to provide additional configuration when an action occurs. For example, you can calculate or retrieve and specify additional field values when a user creates a list item.
- You want to perform additional actions after an action occurs. For example, if a user deletes a site, you can programmatically remove references to that site from lists elsewhere in your SharePoint environment.

One common use of event receivers is in list aggregation scenarios, such as lists of lists or lists of sites. For example, suppose you maintain a list of all project sites within a site collection to support quick find or custom filtering. You can create an event receiver for the **WebProvisioned** event, which adds an entry to the list when a site is created. You could also handle the **WebDeleted** event to remove the entry when the site is deleted. By using event receivers in this way, you maintain an up-to-date list of sites with minimal processing overhead.

Where can you use event receivers?

If you use apps to handle events through remote event receivers, you can deploy your event receiver to both SharePoint Online subscriptions and on-premises SharePoint installations. If you use a farm solution to deploy event receivers, you are limited to on-premises SharePoint installations. Like with any farm solution, the solution must also be approved and installed by a farm administrator.

Sandboxed solutions, which you can deploy to both SharePoint Online subscriptions and on-premises SharePoint installations, provide limited support for event receivers. Specifically, you can only deploy event receivers for web events, list events, and list item events.

Workflow

By using workflows, you can model and automate business processes. Generally speaking, you create workflows when you want to capture user input from multiple people and perform actions in response to that user input. For example, suppose you wanted to automate a document publishing process. Your document might go through the following stages on the way to publication:

1. *Authoring.* A user creates a first draft of the document.
2. *Technical review.* Another user reviews the document for accuracy. If the technical reviewer is satisfied, the document proceeds to legal review. If not, the document goes back to the author for changes.
3. *Legal review.* A member of the legal team reviews the document for compliance with regulations and local statute. If the legal reviewer is satisfied, the document proceeds to copy edit. If not, the document goes back to the author for changes.
4. *Copy edit.* An editor checks the document for spelling, grammar, and style. If the editor requires major changes, the document goes back to the author for incorporation. If only minor changes are required, the document proceeds to publishing.
5. *Publishing.* The document is submitted for printing and distribution.

Continuing with this scenario, let us assume that you want to automate this publishing process. In this case, you can create a workflow that assigns a task to a user at each stage. When the user marks his or her task as complete and provides any required information to the workflow, the workflow logic proceeds to another stage. In this example, the workflow would use branching logic to determine who should see the document next.

One of the key aspects of SharePoint workflow—and any other WF45 workflow—is that the workflow is dormant while waiting for user input. In other words, the workflow does not actively wait for user input by polling for changes in field values or task completion statuses. Instead, WF45 will awaken, or rehydrate, the workflow when it has the information it requires to proceed. This enables workflows to become useful tools for managing long-running business processes.

SharePoint 2013 workflows are scoped to a web (an individual sub-site) or a list. SharePoint provides a wide range of built-in workflow actions that enable you to perform almost every conceivable task at the web or list scope. There are also more generic actions available, such as calling a web service and performing a calculation. To perform more specialized tasks, you can create custom workflow actions in Visual Studio 2012.

You can configure workflows to start automatically when something happens, such as when a user adds a new document to a library. You can also create workflows that users can launch manually when required.

When should you create workflows?

You should create a workflow when you want to automate a business process associated with SharePoint content or artifacts, such as documents or webs. Workflows are a good solution when you need to:

- Capture input from more than one person.
- Create multi-stage logic that reacts to changes in documents or sites.
- Create logic that waits for and responds to user input.

- Use workflows to:
 - Automate business processes
 - Manage the flow of documents and information
- When should you create a workflow?
 - When you need to capture input from multiple users
 - When you need to create logic that reacts to changes in documents or sites
- Where can you use event receivers?
 - On-premises deployments
 - SharePoint Online

Where can you use workflows?

SharePoint 2013 workflows are entirely declarative. You can package workflows using either apps or sandboxed solutions, and you can deploy workflows to both SharePoint Online subscriptions and on-premises installations of SharePoint.

Discussion: Choosing a Suitable Development Approach

The instructor will now lead a discussion around the following scenarios. Which approach to development would you use in each scenario, and why?

Scenario 1

Contoso wants to create a system for maintaining core pharmaceutical licensing information and make it available to multiple site collections. This system will consist of several site columns, a new content type, and a list that uses the new content type. Site collection administrators must be able to provision all these components with a single action.

Scenario 2

The research and development team at Contoso uses site collections based on the Project Site template for experimental project work. There are often more than a hundred of these project sites. New project sites are created on a regular basis, and existing project sites are regularly retired. The team wants to provide a central index of active research projects on the Contoso intranet home page. The index should enable users to filter the list of projects by research area, scientists involved, clinical outcomes, and other key criteria.

Scenario 3

Contoso wants to automate the expense approval process. Expense claims for less than \$2,000 USD must be approved by the user's line manager and then processed by the finance team. Expense claims for more than \$2,000 USD must be approved by the user's line manager and then counter-signed by an executive before being processed by the finance team.

- Review the three scenarios in the student workbook
- Which approach to development would you use in each scenario, and why?

Lesson 3

Understanding SharePoint 2013 Deployment and Execution Models

In SharePoint 2013, you can package, deploy, and execute custom functionality in a variety of ways. Each deployment and execution model includes various capabilities and constraints. In this lesson, you will learn about your options for packaging and deploying custom functionality for SharePoint 2013.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain how SharePoint Features work.
- Describe the capabilities of farm solutions.
- Describe the capabilities of sandboxed solutions.
- Describe the capabilities and key development options for apps.

SharePoint Features

A SharePoint Feature is a framework for packaging and deploying declarative components. Features were introduced in SharePoint 2007 and have been a major component of SharePoint development ever since. The SharePoint development tools in Visual Studio 2010 and Visual Studio 2012 automate much of the Feature development process and provide a Feature designer to help you assemble the Feature components. However, it is important for developers to understand what Features are and how they work.



Note: When referencing SharePoint Features, the word Feature is typically initial-capped in product documentation. This helps to distinguish between SharePoint Features and features as a generic term.

Features are contained within a folder. The name of the folder is important, because you will refer to the feature by the name of this folder when you install or uninstall the feature. The root of this folder always contains a file named Feature.xml. This is known as the feature manifest, which describes the contents of the Feature package and indicates what SharePoint should do with the contents of the Feature when it is installed and activated. The feature folder also typically contains subfolders and files that contain the contents and resources that make up the Feature.

- Anatomy of a Feature
 - Feature folder
 - Feature manifest file
 - Element manifests
 - Element files
- Feature deployment
 - Deployment to WFE server file system
 - Deployment as part of SharePoint app or solution

The following code shows a simple example of a feature manifest file:

A Feature Manifest File

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint"
  Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  Title="Contoso Project Resources"
  Description="Contains site columns, content types, and lists for Contoso project
sites."
  Scope="Web">
  <ElementManifests>
    <ElementManifest Location="SiteColumns\Elements.xml" />
    <ElementManifest Location="ContentTypes\Elements.xml" />
    <ElementFile Location="Pages\ContosoProject.aspx" />
    <ElementFile Location="Images\ProjectIcon.png" />
  </ElementManifests>
</Feature>
```

There is far more to feature manifest files than shown in this example, and you will learn more about building Features later in this course. For now, there are three key aspects of note in this feature manifest example:

- *Scope.* Features are always scoped to a particular level in the SharePoint hierarchy. The **Scope** attribute can take the values **Farm**, **WebApplication**, **Site** (a site collection), or **Web** (an individual sub-site).
- *Element manifests.* Features use **ElementManifest** elements to indicate the location of an element manifest within the folder structure of the Feature. Remember from the previous lesson that an element manifest is a CAML-based XML file that contains a declarative definition of a SharePoint component, such as a site column or a content type.
- *Element files.* Features use **ElementFile** elements to indicate the location of files within the folder structure of the feature. Unlike element manifests, which SharePoint must parse to be able to provision the custom components described within, an element file is simply a file—such as an ASPX page or an image—that must be deployed by the Feature.

Traditionally, you deploy a Feature by copying the Feature folder to the 15\TEMPLATE\FEATURES folder on the file system of each SharePoint WFE server. You can then use Windows PowerShell, or the server-side object model, to install the feature and activate it to the desired scope. However, Features are rarely deployed manually in this way. This would be laborious and difficult to manage, because you would need to manually copy files to every SharePoint WFE server and ensure that the files remained synchronized when the Feature was updated, or when new servers were added to the farm. Instead, Features are typically deployed as part of a farm solution, a sandboxed solution, or a SharePoint app, which manages the feature installation and activation process. If you deploy a Feature within a SharePoint app or a sandboxed solution, the Feature is never copied to the server file system; instead, it remains in the content database from where it is accessed when required.



Note: You will learn more about creating Features later in this course.

Farm Solutions

SharePoint solutions were introduced in SharePoint 2007 as a mechanism for deploying custom functionality, including assemblies, to a SharePoint environment. A solution is essentially a cabinet file with a .wsp file name extension. The term *farm solution* indicates that the solution package is deployed at the farm scope by a farm administrator, using either Windows PowerShell or the Central Administration website.

All solutions must contain a solution manifest file named Manifest.xml. This file describes the contents of the package and tells SharePoint how to process each individual item in the package. In addition to the solution manifest file, the solution may contain:

- Anatomy of a farm solution
 - Solution manifest
 - Assemblies
 - Files
 - Features
- Capabilities are unlimited
 - Deploy any server-side components
- Deployment options may be limited
 - Prohibited in SharePoint Online
 - May be prohibited in on-premises deployments

- *Assemblies*. Farm solutions are often used to deploy server-side code to a SharePoint environment. The solution manifest tells SharePoint whether to deploy the assembly to the global assembly cache (GAC) for farm-scoped components, or to the bin directory for the IIS website for web application-scoped components. You can also specify which classes within your assemblies should be registered as safe controls within the Web.config file.
- *Files*. Farm solutions can deploy files both to physical locations within the SharePoint server root folder and to virtual paths within content databases.
- *Features*. Farm solutions are often used to deploy Features. The entire Feature folder is added to the solution package, and the solution manifest indicates the location of the feature manifest file.

The following code shows a simple example of a solution manifest file.

A Solution Manifest File

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint"
  SolutionId="xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx"
  SharePointProductVersion="15.0">
  <Assemblies>
    <Assembly Location="ContosoProjects.dll" DeploymentTarget="GlobalAssemblyCache">
      <SafeControls>
        <SafeControl Assembly="..." Namespace="..." TypeName="*" />
        <SafeControl Assembly="..." Namespace="..." TypeName="*" />
      </SafeControls>
    </Assembly>
  </Assemblies>
  <FeatureManifests>
    <FeatureManifest Location="ContosoProjectResources\Feature.xml" />
  </FeatureManifests>
</Solution>
```



Best Practice: Deploying Features within solution packages offers many advantages over manually installing Features. When you install a farm solution, any Features within the solution are automatically copied to every WFE server in the SharePoint farm. When you uninstall the solution, the Features are automatically removed from every WFE server in the farm.

In some cases, you may never see the solution manifest file for your SharePoint development projects. Visual Studio creates the file for you automatically and creates the solution package when you build your

project. However, a solid understanding of how solutions work will help you to troubleshoot your deployments and anticipate potential problems.

You can use farm solutions to deploy all server-side components. However, it is not always possible to deploy a farm solution. Farm solutions must be deployed by a farm administrator who has permission to deploy components to the server file system. You cannot deploy farm solutions to SharePoint Online subscriptions. In addition, many organizations have strict governance policies that prohibit the deployment of farm solutions, or limit access to servers. As a general rule, you should only package your custom components in a farm solution if there are no alternatives.

Examples of components that are deployed using a farm solution include:

- Timer jobs.
- Application pages.
- Any server-side code that requires access to the server file system.
- Any server-side code that requires elevated permissions.

You should evaluate whether you can use a SharePoint app or a sandboxed solution to deploy your custom components. If you cannot use a SharePoint app or a sandboxed solution, you should consider whether there are alternative ways of meeting business requirements that allow you to use a SharePoint app or a sandboxed solution. If there are no other alternatives, and if your SharePoint environment permits it, creating a farm solution is probably the right choice.

Sandboxed Solutions

Sandboxed solutions were introduced in SharePoint 2010 to address some of the deployment limitations of farm solutions. The introduction of SharePoint Online and multi-tenant SharePoint installations meant that developers were increasingly unable to access the server-side file system or deploy solutions that could affect the performance and stability of the platform as a whole. Sandboxed solutions represented a compromise. Site collection administrators could deploy sandboxed solutions directly to a site collection, without requiring access to the server-side file system and without requiring approval from the IT team. At the same time, sandboxed solutions could run in an isolated, low-privilege process and have only limited access to the server-side object model.

In terms of structure, a sandboxed solution is identical to a farm solution. The solution package is a .wsp file and can contain assemblies, files, and features. However, there are several important differences in how sandboxed solutions are deployed and executed:

- Sandboxed solutions are deployed to a Solutions Gallery within a site collection. This means that all resources are stored within the content database, and nothing is deployed to the server-side file system.
- Any code within sandboxed solutions is executed within an isolated sandbox worker process, rather than within an IIS application pool. Sandbox worker processes run with an extremely limited permission set.

- Structured in the same way as a farm solution
- Deployed to a Solutions Gallery
- Scoped to a site collection
- Functionality is constrained:
 - Isolated worker process
 - No access to server-side file system
 - Limited access to SharePoint object model
- Resource consumption governed by quota system
- Apps for SharePoint are now the preferred approach

- Code running within a sandbox worker process has limited access to the SharePoint object model. Most notably, sandboxed code cannot interact with objects beyond the scope of the site collection, cannot run with elevated privileges, and cannot interact with the server file system in any way.

SharePoint uses a quota system to manage the resources consumed by sandboxed solutions. Each site collection is allocated a daily quota of resource points for use by sandboxed solutions, which is set by the farm administrator. Sandboxed solutions consume resource points according to 14 different measures, such as CPU execution time, number of database queries, and number of unhandled exceptions. If an individual sandboxed solution exceeds the absolute limit for any one of these measures, the sandbox worker process is terminated immediately. If a site collection reaches its daily threshold of resource point consumption, no further sandboxed code is allowed to run for the rest of the day. Site collection administrators can monitor the resource consumption of individual sandboxed solutions to ensure that they stay within their daily limits.

Although the sandboxed solution approach is still supported, apps for SharePoint are now the preferred approach to deploying custom functionality to restricted environments. Apps for SharePoint offer a broader set of functionality than sandboxed solutions, by leveraging the client-side object model rather than a subset of the server-side object model. There is rarely any justification for using a sandboxed solution instead of a SharePoint app for new projects.

Apps for SharePoint

Apps for SharePoint provide a new framework for creating and deploying custom functionality in SharePoint 2013. Apps do not execute any code on SharePoint servers. Any app logic either runs on the client computer (in the form of JavaScript running within the web browser) or in the cloud (either on Windows Azure or an alternative remote server platform). All interaction with SharePoint takes place through client-side programming, either through one of the client-side object models or through the REST/OData API.

- Distribution
 - Publish to App Catalog
 - Publish to Office Marketplace
- Encapsulation
 - No server-side code
 - All SharePoint artifacts hosted within app web
- Development models
 - SharePoint-hosted
 - Remote-hosted
- Interaction
 - Full page
 - App part
 - Command extensions

Apps are packaged and distributed as an .app file.

Within an organization, you can make apps available to site collection administrators and other users by publishing the app to an App Catalog site. You can also make apps available to a wider audience by publishing them to the Microsoft Office Marketplace. SharePoint 2013 includes the infrastructure to manage licensing, subscriptions, and the app life cycle.

When a user installs an app, the app provisions its own subsite within the host web. This subsite is known as the app web. Any SharePoint artifacts contained within the app, such as site columns, content types, and list instances, together with any SharePoint-hosted webpages, are created within the app web. This provides a degree of isolation between the app and the rest of the SharePoint site; any changes that occur within the app web are unlikely to have any unforeseen consequences on content elsewhere on SharePoint. This encapsulation also ensures that installing and uninstalling apps is seamless.

Apps for SharePoint fall into two broad categories: SharePoint-hosted apps and remote-hosted apps. When you create a SharePoint-hosted app, all the resources required by your app are hosted by the app web. Every app has a start page, which is the page that SharePoint displays when a user launches the app. In a SharePoint-hosted app, this start page is an ASPX page contained within the app web. In a remote-hosted app, the start page, together with any other webpages required by the app, are hosted on a remote server. SharePoint-hosted apps always rely on JavaScript to execute any logic and interact with SharePoint, whereas remote-hosted apps can run on any server platform.

You can enable users to interact with apps from a SharePoint site in the following ways:

- *Full-page apps.* A full-page app uses the entire browser window. Users typically launch a full-page app from the Site Content page, or by clicking a button on the ribbon or the Edit Control Block for a list item or document.
- *App parts.* An app part is essentially a Web Part that renders a page from your app within an iFrame. Adding an app part to your app enables users to view app content alongside other SharePoint artifacts in the host web.
- *Command extensions.* A command extension is a declarative component that adds a menu item or button to specific areas of the user interface, such as the ribbon or the Edit Control Block (ECB) for a document. You can deploy command extensions to the host web that launch content from your app.

In terms of capabilities, apps for SharePoint are constrained only by the limitations of the client-side object model. This means that you can perform almost any action within the bounds of a site collection; however, you cannot use the server-side object model, and you cannot interact with the server-side file system in any way.

One of the major benefits of apps for SharePoint, and apps for Office, is that they work in exactly the same way on SharePoint Online subscriptions and on-premises SharePoint installations. Apps offer the reach of sandboxed solutions, but with fewer restrictions on functionality, no resource quotas to manage, and superior life cycle management.



Note: You will learn more about creating and deploying apps for SharePoint throughout this course.

Lab: Comparing Web Parts and App Parts

Scenario

You have been asked to provide the developer team at Contoso with an overview of how to create and deploy custom SharePoint 2013 components. As part of this process, you will create a simple Web Part in Visual Studio 2012. You will explore the structure of the solution in Visual Studio and then deploy it to your SharePoint development environment. You will then create a simple app part in Visual Studio 2012, explore the solution, and deploy it to your SharePoint development environment.

Objectives

After completing this lab, you will be able to:

- Create and deploy a SharePoint Web Part in Visual Studio 2012.
- Create and deploy a SharePoint app in Visual Studio 2012.

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-01
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

Exercise 1: Creating and Deploying a SharePoint Web Part

Scenario

In this exercise, you will create and deploy a Web Part that provides a personalized greeting to the current user. First, you will create a new Visual Web Part project in Visual Studio 2012. You will then explore the components that Visual Studio creates for you, and refactor the solution to use more intuitive names for the solution components. You will then create a simple user interface and add some code to generate a personalized greeting. Finally, you will deploy and test the Web Part.

The main tasks for this exercise are as follows:

1. Create a Visual Web Part Project in Visual Studio 2012
2. Configure the Web Part Solution
3. Create the Web Part User Interface
4. Add Code to Greet the Current User
5. Test the Web Part

► Task 1: Create a Visual Web Part Project in Visual Studio 2012

- Start the 20488B-LON-SP-01 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio 2012 and create a new project with the following settings:
 - a. Use the **SharePoint 2013 - Visual Web Part** project template.
 - b. Name the project **GreetingWebPartProject**.
 - c. Create the project in the **E:\Labfiles\Starter** folder.
 - d. Use the site at **http://team.contoso.com** for debugging.
 - e. Select the farm solution deployment option.



Note: A specific issue prevents the running of sandboxed solution code on Windows Server 2012 servers that are domain controllers, so in this case you will use a farm solution to deploy the Web Part. In a real world scenario, you should configure your development environment to support sandboxed solutions, but that is impractical in the classroom environment.

► Task 2: Configure the Web Part Solution

- In Solution Explorer, change the name of the Feature to **GreetingWebPart**.
- Change the name of the ASCX control to **GreetingWebPart.ascx**.
- Change the name of the Web Parts control description file to **GreetingWebPart.webpart**.
- Explore the contents of the **GreetingWebPart** node. Notice that the node contains:
 - An ASCX file and a corresponding code-behind file.
 - A .webpart file.
 - An Elements.xml file.
- In the Web Parts control description file (.webpart file), change the title of the Web Part to **Greet Users**.
- In the Web Parts control description file, change the description of the Web Part to **Welcomes the current user with a friendly message**.
- In the element manifest file for the Web Part, change the value of the **Group** property to **Contoso Web Parts**.
- In Solution Explorer, change the name of the Feature to **GreetingWebPart.feature**.
- Open the Feature, and notice that the Feature is scoped to the **Site** level.
- Change the title of the Feature to **Greet Users Web Part**.
- Change the description of the feature to **Adds the Greet Users web part to the gallery**.
- Explore the contents of the Feature, and notice that the Feature includes two files: the **Elements.xml** file and the **GreetingWebPart.webpart** file.
- On the **Manifest** tab, notice how the Feature contains an element manifest and an element file.
- In Solution Explorer, double-click **Package**.
- Notice that the solution package will deploy both the Feature and the **GreetingWebPart** project item, which is essentially the assembly for the ASCX control.
- Click **Manifest**, and review the solution manifest file.
- Save your changes and close all open tabs.

► Task 3: Create the Web Part User Interface

- Open the **GreetingWebPart.ascx** page.
- Add a **div** element with an **id** value of **greeting** to the page.
- Within the **div** element, add an ASP.NET **Label** control with an **ID** value of **lblGreeting**.

► Task 4: Add Code to Greet the Current User

- Switch to the code-behind file for the ASCX control.
- Change the name of the class from **VisualWebPart1** to **GreetingWebPart**, and ensure that this change is applied throughout the solution.
- Add the following **using** directive to the file:

```
using Microsoft.SharePoint;
```

- In the **Page_Load** method, retrieve the display name of the current user.



Note: You can get the display name of the current user from the **SPContext.Curent.Web.CurrentUser** property.

- Create a string-based welcome message that includes the name of the current user.
- Display the welcome message in the **lblGreeting** control.
- Build the solution, and verify that your code builds without errors.

► Task 5: Test the Web Part

- On the **DEBUG** menu, click **Start Without Debugging**.



Note: In the classroom environment, the virtual machine performs better if you test the Web Part without attaching the debugger.

- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- When the **Team Site** page loads, add the **Greet User** Web Part to the page.
- Verify that the **Greet User** Web Part greets the current user by name.
- Discard your changes and close Internet Explorer.

Results: After completing this exercise, you should have created and deployed a Web Part that greets the current user.

Exercise 2: Creating and Deploying a SharePoint App Part

Scenario

In this exercise, you will create and deploy an app part that provides a personalized greeting to the current user, in a similar way to the Web Part you created in the previous exercise. First, you will create a new SharePoint app project in Visual Studio 2012. You will explore the artifacts that Visual Studio creates for you, and you will then add and configure a Client Web Part component to the app project. You will add some JavaScript code to generate a personalized greeting, and you will test the app part on a developer site.

The main tasks for this exercise are as follows:

1. Create a SharePoint App Project in Visual Studio 2012
2. Review the Contents of the App Project

3. Add a Client Web Part to the App
4. Add Code to Greet the Current User
5. Test the App Part

► **Task 1: Create a SharePoint App Project in Visual Studio 2012**

- Open Visual Studio 2012 and create a new project with the following settings:
 - a. Use the **App for SharePoint 2013** project template.
 - b. Name the project **GreetingAppProject**.
 - c. Create the project in the **E:\Labfiles\Starter** folder.
 - d. Set the name of the app to **Greet Users**.
 - e. Use the site at **http://dev.contoso.com** for debugging.
 - f. Select the **SharePoint-hosted** deployment model.

► **Task 2: Review the Contents of the App Project**

- Review the contents of the **Content** node. Notice that the node contains a CSS file (App.css), together with an element manifest that deploys the CSS file (Elements.xml).
- Review the contents of the **Images** node. Notice that the node contains an image (AppIcon.png), together with an element manifest that deploys the image (Elements.xml).



Note: The AppIcon.png image is the default icon that represents your app on the Site Contents page in SharePoint. In most cases, you would replace this file with your own icon or logo.

- Review the contents of the **Pages** node. Notice that the node contains an ASPX page (Default.aspx), together with an element manifest that deploys the page (Elements.xml).



Note: The Default.aspx file is the default page for the app. The default page is displayed when a user first launches a full-page app. You can edit this page or make a different page your default page.

- Review the contents of the **Scripts** node. Notice that the node contains several JavaScript files, including the jQuery library, together with an element manifest that deploys the JavaScript files (Elements.xml).
- Change the name of **Feature1** to **GreetingResources**.
- Open the **GreetingResources** feature. Notice that the Feature deploys the files from the **Pages**, **Scripts**, **Content**, and **Images** nodes.



Note: The Feature provisions all of these items to the app web—not the host web—when a user installs the app.

- In Solution Explorer, under **Scripts**, open the App.js file.



Note: App.js is the default code file for the app. By default, the App.js file includes a method that attempts to get the name of the current user. The success callback function uses jQuery to inject a greeting message into the Default.aspx page.

- Open the **AppManifest.xml** file. Notice that the app manifest defines various app properties, including the icon and the start page.
- Close all open tabs.

► Task 3: Add a Client Web Part to the App

- Add a new **Client Web Part (Host Web)** item named **GreetUserPart** to the project.
- On the **GreetUserPart.aspx** page, within the **head** element and after the existing content, add a **script** element that loads the **Greeting.js** file from the **Scripts** folder.



Note: You will add the Greeting.js file to your project shortly.

- Within the **body** element, add a **div** element.
- Within the **div** element, add a paragraph element with an **id** value of **greeting**.
- Set the content of the paragraph element to **Loading your app...**
- Save and close the **GreetUserPart.aspx** page.
- Open the element manifest file for the **GreetUserPart** project item.
- In the element manifest, change the title of the app part to **Greet Users**.
- Change the description of the app part to **Welcomes the current user with a friendly message**.
- Save and close the **Elements.xml** file.

► Task 4: Add Code to Greet the Current User

- Add the JavaScript file from **E:\Labfiles\Starter\Snippets\Greetings.js** to the **Scripts** folder.
- Review the contents of the **Greetings.js** file. The code in the file performs the following actions:
 - a. It retrieves the name of the current user.
 - b. It creates a personalized greeting for the current user, and inserts it into the HTML element with the **id** value of **greeting**.



Note: You do not need to understand how the JavaScript code works in detail at this stage. You will learn more about working with the client-side object model later in this course.

► Task 5: Test the App Part

- Click **Start**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$wOrd**.
- When the **Greet Users** page loads, click **Contoso Development Site**.
- Add the **Greet Users** app part to the page.
- Verify that the **Greet Users** app part welcomes the current user by name.

- Discard your changes and close Internet Explorer.

Results: After completing this exercise, you should have created and deployed an app part that greets the current user.

Module Review and Takeaways

In this module, you gained a broad understanding of SharePoint 2013 as a developer platform. You learned about the different approaches to SharePoint development, together with the scenarios in which each approach might be appropriate. You also learned about the different execution and deployment models provided by SharePoint 2013, together with when you can use each model.

Review Question(s)

Test Your Knowledge

Question	
Which of the following best describes unmodified content pages in SharePoint 2013?	
Select the correct answer.	
<input type="checkbox"/>	The page template and the page content are stored on the server-side file system.
<input type="checkbox"/>	The page template and the page content are stored in the content database.
<input type="checkbox"/>	The page template is stored on the server-side file system, and the page content is stored in the content database.
<input type="checkbox"/>	The page template is stored in the content database, and the page content is stored on the server-side file system.
<input type="checkbox"/>	The page template is stored in the configuration database, and the page content is stored in the content database.

Test Your Knowledge

Question	
You need to automate a business process that collects information from several different users. Which approach to development should you use?	
Select the correct answer.	
<input type="checkbox"/>	Create a Web Part.
<input type="checkbox"/>	Create an application page.
<input type="checkbox"/>	Create a timer job.
<input type="checkbox"/>	Create an event receiver.
<input type="checkbox"/>	Create a workflow.

Test Your Knowledge

Question	
<p>You need to make a list template, together with custom site columns and content types, available to users across Contoso. Which deployment model should you use?</p>	
<p>Select the correct answer.</p>	
<input type="checkbox"/>	Create and install a feature. Manually activate the feature on site collections where the list template is required.
<input type="checkbox"/>	Create a feature within a sandboxed solution. Install the sandboxed solution on site collections where the list template is required.
<input type="checkbox"/>	Create a feature within a farm solution. Install the solution at the farm scope. Activate the feature on site collections where the list template is required.
<input type="checkbox"/>	Create a feature within a SharePoint app. Publish the app to your corporate catalog. Add the app on-site collections where the list template is required.
<input type="checkbox"/>	Create and install a feature. Use feature stapling to associate the feature with the Project Site template.

Module 2

Working with SharePoint Objects

Contents:

Module Overview	2-1
Lesson 1: Understanding the SharePoint Object Hierarchy	2-2
Lesson 2: Working with Sites and Webs	2-13
Lab A: Working with Sites and Webs	2-21
Lesson 3: Working with Execution Contexts	2-25
Lab B: Working with Execution Contexts	2-30
Module Review and Takeaways	2-34

Module Overview

The server-side SharePoint object model provides a core set of classes that represent different items in the logical architecture of a SharePoint deployment. Almost all server-side development tasks for SharePoint involve working with one or more of these objects. In this module, you will learn about these core classes and how they relate to the logical components in your SharePoint environment. You will learn how to work with the classes that represent site collections and sites, which lie at the heart of most SharePoint development activities. You will also learn about execution context and the key considerations for developing code that may be invoked by a variety of different users.

Objectives

After completing this module, you will be able to:

- Explain the purpose of key classes in the server-side SharePoint object model.
- Programmatically interact with SharePoint site collections and sites.
- Adapt solutions for users with different levels of permissions.

Lesson 1

Understanding the SharePoint Object Hierarchy

You can view a SharePoint deployment as a hierarchical set of logical components. At the top level, you have the SharePoint farm itself. Beneath the SharePoint farm, you have services, service applications, and web applications, followed by site collections, sites, and so on. Each of these logical components is represented by a class in the SharePoint object model. Understanding how these classes relate to each other will help you when you develop custom solutions for SharePoint 2013.


Lesson Objectives

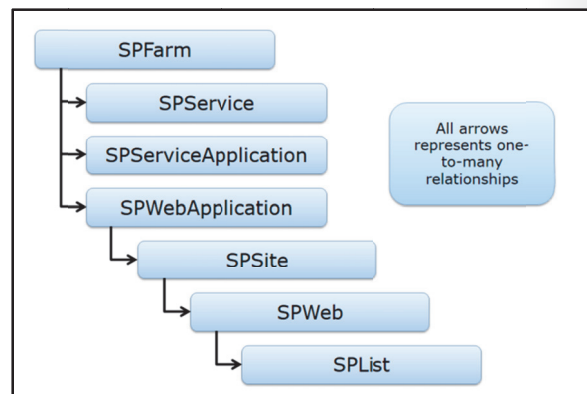
After completing this lesson, you will be able to:

- Describe the key classes in the SharePoint object hierarchy.
- Describe how to work with the **SPFarm** class.
- Describe how to work with services and service instances.
- Explain the service application architecture from a developer's perspective.
- Describe how to work with the **SPWebApplication** class.
- Describe the broad purpose and functionality of the **SPSite** class.
- Describe the broad purpose and functionality of the **SPWeb** class.

The SharePoint Object Hierarchy

All of the SharePoint 2013 object models provide a core set of classes that represent key components of a SharePoint deployment. Just like the logical architecture of a SharePoint deployment, these classes are arranged in a hierarchical manner.

 **Note:** In this module, you will focus on classes in the server-side object model. Later in the course, you will learn more about parallel classes in the client-side object models.



The core classes in the server-side object model are as follows:

- *SPFarm*. The **SPFarm** class represents a SharePoint 2013 farm in the logical architecture sense: a configuration database, a collection of services, a collection of service applications, and a collection of web applications.
- *SPService*. The **SPService** class represents a SharePoint service instance, such as Excel Calculation Services, the Machine Translation Service, or the User Profile Service.
- *SPServiceApplication*. The **SPServiceApplication** class represents a SharePoint service application, such as the Search Service Application, the Managed Metadata Service application, or the User Profile Service Application.
- *SPWebApplication*. The **SPWebApplication** class represents a SharePoint web application. A web application can contain one or more site collections.

- *SPSite*. The **SPSite** class represents a SharePoint site collection. A site collection can contain one or more sites.
- *SPWeb*. The **SPWeb** class represents an individual SharePoint site. A site can contain lists, libraries, and various other artifacts. Each site can also contain one or more subsites, which are also represented by **SPWeb** objects.
- *SPList*. The **SPList** class represents a SharePoint list or library.

Together, these classes represent the hierarchy of a SharePoint deployment in its entirety. Each object exposes an enumerable collection of the objects below it in the hierarchy.



Note: You will learn more about each of these objects in subsequent topics.

The SPFarm Class

SPFarm objects are the highest-level object in the hierarchy and, as their name implies, they represent the SharePoint farm as a logical entity. The **SPFarm** class does not include a constructor. Instead, the class provides a global static object that represents the current farm through the **Local** property.

The following code example shows how to get a reference to the local SharePoint farm by using the server-side object model (C#):

- Highest-level object in the hierarchy
- Represents farm-wide configuration
- Instantiate through the static **Local** property
- Use properties and methods to retrieve configuration settings

```
SPFarm farm = SPFarm.Local;

Guid farmID = SPFarm.Local.Id;
Bool isAdmin =
    SPFarm.Local.CurrentUserIsAdministrator();
```

Getting a Reference to the Local SharePoint Farm

```
SPFarm farm = SPFarm.Local;
```

SPFarm objects represent farm-level configuration settings, and act as an overall container for all other objects in the farm. For example, the **SPFarm** class exposes properties such as **DefaultServiceAccount** and **TimerService**, and contains collection properties such as **Servers**, **Services**, and **Solutions**.



Note: To use the **SPFarm** class in your code, you must add a reference to the **Microsoft.SharePoint.Administration** namespace.

In general, you will only use **SPFarm** objects if you are creating administrative applications for SharePoint. You do not need to use this object to access site collections, sites, lists, or list data. However, the **SPFarm** class is useful when you need to retrieve farm-level configuration settings for use elsewhere in your code. For example, you can use the **Id** property to retrieve the unique identifier for the local farm, which you need when you want to configure trust relationships between SharePoint farms so you can share service applications. In other scenarios, you might want to use the **CurrentUserIsAdministrator** method to determine whether the current user is a member of the farm administrators group before performing some configuration tasks.

The following code example shows how to work with **SPFarm** properties and methods:

Working with SPFarm Objects

```
//Get the unique identifier for the local farm.
Guid farmID = SPFarm.Local.Id;

//Determine whether the current user is a farm administrator.
bool isAdmin = SPFarm.Local.CurrentUserIsAdministrator();
```



Note: You can also work with **SPFarm** objects in Windows PowerShell. The **Get-SPFarm** cmdlet returns the local farm instance.

Because the **SPFarm** class represents farm-level configuration and infrastructure settings, you cannot use the **SPFarm** class from sandboxed solutions or from client-side code. There is no client-side equivalent to the **SPFarm** class. For the same reasons, you cannot use the **SPFarm** class when you work with SharePoint Online subscriptions.

Working with Services

Like previous versions of SharePoint products and technologies, SharePoint 2013 provides functionality through several farm-wide services. Many services, such as Excel Calculation Services or the User Profile Service, underpin specific service applications. Other services, such as the Microsoft SharePoint Foundation Web Application service or the Distributed Cache service, underpin a broader range of SharePoint functionality.

Although services are defined at the farm level, service instances actually run on individual servers. In a scaled-out SharePoint deployment, running service instances are often distributed between servers to even out the workload of each server. In a highly available SharePoint deployment, instances of the same service often run on more than one server. As such, it is important to distinguish between services and service instances. The SharePoint server-side object model uses the following classes to represent services and service instances:

- **Service classes**
 - *SPService* represents a farm-scoped service
 - *SPServiceInstance* represents an instance of an *SPService* on a specific server
- **The SPWebService class**
 - Container for web applications
 - *ContentService*
 - *AdministrationService*
- **Scope**
 - Not available in SharePoint Online
 - Not available in sandboxed solutions or client-side code

- *SPService*. This class represents a farm-scoped service. All farm-wide SharePoint services derive from the **SPService** class.
- *SPServiceInstance*. This class represents a server-scoped service instance. The **SPServiceInstance** class associates a specific service with a specific server.

Working with SPService and SPServiceInstance objects

As a developer, your interactions with services are likely to be minimal. Creating a custom service, although possible, is not a common SharePoint development task. However, in some circumstances you may want to verify programmatically whether particular services are running; for example, if you are automating deployment tasks or attempting to perform procedures that will fail if services are not running.

The following code example shows how to check whether at least one instance of a specified service is running in the local SharePoint farm:

Working with SPService and SPServiceInstance Objects

```
string upssTypeName = "User Profile Synchronization Service";
bool isRunningInstance = false;

foreach (SPService service in SPFarm.Local.Services)
{
    if (String.Equals(service.TypeName, upssTypeName))
    {
        foreach (SPServiceInstance instance in service.Instances)
        {
            if (instance.Status == SPObjctStatus.Online)
            {
                isRunningInstance = true;
            }
        }
    }
}

if (isRunningInstance)
{
    // At least one instance of the User Profile Synchronization Service is running.
}
```



Note: You can also use Windows PowerShell to work with services and service instances. Windows PowerShell is often more convenient than managed code for provisioning and configuring farm-level components such as services.

The SPWebService class

The **SPWebService** class is primarily a management container for SharePoint web applications (**SPWebApplication** objects). Like all SharePoint services, the **SPWebService** class derives from the **SPService** class. However, the **SPWebService** class includes some useful additional functionality. For example, you will work with the **SPWebService** class if you want to:

- Apply configuration changes to multiple web applications.
- Programmatically create or delete web applications.
- Change default settings for new content databases.
- Retrieve farm-scoped features from the SharePoint environment.
- Apply Web.config modifications to one or more web applications.

The **SPWebService** class includes two useful static properties: **ContentService** and **AdministrationService**. The **ContentService** property returns the **SPWebService** instance that contains regular (content) web applications. The **AdministrationService** property returns the **SPWebService** instance that contains the Central Administration web application.

The following code example shows how to enumerate the web applications in **SPWebService** instances:

Working with the SPWebService Class

```
var contentService = SPWebService.ContentService;
foreach (var webApp in contentService.WebApplications)
{
    // Perform actions on each content web application.
}

var adminService = SPWebService.AdministrationService;
foreach (var webApp in adminService.WebApplications)
{
    // Perform actions on the Central Administration web application.
}
```

Services in SharePoint Online

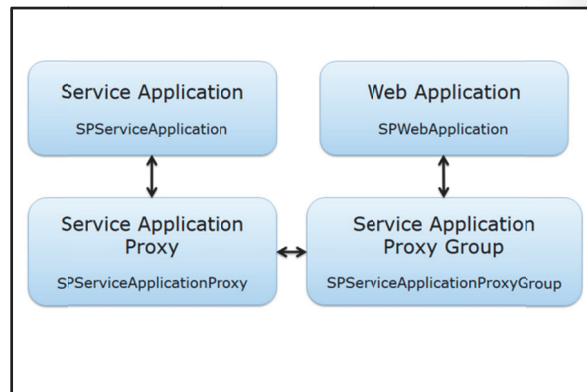
In a SharePoint Online subscription, services and service instances are managed for you. You cannot interact with services programmatically in solutions for SharePoint Online, nor should you need to.

You cannot interact with services from sandboxed solutions or from client-side code, regardless of whether you are deploying your custom functionality to a SharePoint Online subscription or an on-premises deployment of SharePoint 2013. There are no client-side equivalents of the **SPService** or **SPServiceInstance** classes.

Working with Service Applications

SharePoint 2013 uses service applications to make specific functionality available to web applications. For example:

- Search functionality is provided through the Search Service Application.
- Term set functionality and content type publishing is provided by the Managed Metadata Service application.
- User profile management and social features are provided by the User Profile Service Application.



As a SharePoint developer, it is extremely unlikely that you will need, or want, to create a custom service application. You are far more likely to leverage functionality provided by existing service applications. However, it is important to have at least a general understanding of how the service application architecture works.

There are three main components to the service application model in SharePoint 2013:

- *Service applications.* Service applications are provisioned at the farm scope and encapsulate a specific area of functionality, such as managed metadata or user profile management. All service applications derive from the abstract **SPServiceApplication** base class.
- *Service application proxies.* Service applications are consumed through service application proxies. Each service application can have multiple proxies (although one proxy per SharePoint farm is typical). When a web application wants to use some service application functionality, it interacts with

a service application proxy, which in turn communicates with the service application itself. All service applications derive from the abstract **SPServiceApplicationProxy** base class.

- *Service application proxy groups.* Service application proxies are associated with web applications through service application proxy groups. The proxy group is essentially a container for organizing service application proxies. A service application proxy can belong to multiple proxy groups. A web application is associated with a single proxy group. Proxy groups are represented by the **SPServiceApplicationProxyGroup** class.

Discussion: Understanding the Service Application Architecture

The instructor will now lead a brief discussion around the following questions:

- Why have service application proxies? What purpose do they serve?
- Why have service application proxy groups?

- Why have service application proxies?
- Why have service application proxy groups?

Working with Web Applications

SharePoint web applications are containers for site collections and sites. Web applications are also the items in the object hierarchy that map SharePoint sites to Internet Information Services (IIS) websites.

By default, there is a one-to-one mapping between SharePoint web applications and IIS websites. If you use the Central Administration website to create a web application, SharePoint provisions a corresponding IIS website for you, using the URL, port number, and host header you specify. Many settings are configured at the web application level, such as resource throttling settings, authentication providers, and permission policies.

- Containers for site collections
- Map SharePoint content to IIS websites
- Represented by the **SPWebApplication** class

```
var contentService = SPWebService.ContentService;
SPWebApplication webApp = contentService
    .WebApplications["Display Name"];
webApp.MaximumFileSize = 75;
webApp.Update();
```

In the SharePoint object model, web applications are represented by the **SPWebApplication** class. Although you can create and delete web applications programmatically (by working with the parent **SPWebService** object), this is not a common requirement. Web applications are typically created by administrators, using either Windows PowerShell or the Central Administration website. As a developer, you are more likely to work with existing **SPWebApplication** objects to retrieve or modify application-scoped settings.

You can instantiate an **SPWebApplication** object in various ways. For example, you can:

- Enumerate or index the **WebApplications** property of the parent **SPWebService** object.
- Use the **SPWebApplication** constructor and specify the display name of the web application, the parent **SPWebService** object, and the **SPApplicationPool** object that represents the credentials under which the web application process runs.

- Retrieve the **WebApplication** property from an **SPSite** object, to get the parent web application for a specific site collection.

The following code example shows how to retrieve an **SPWebApplication** object and modify settings:

Working with SPWebApplication Objects

```
// Get a reference to the parent SPWebService instance.
var contentService = SPWebService.ContentService;
// Use an indexer to retrieve the web application by display name.
var webApp = contentService.WebApplications["Contoso Content"];

// Change the maximum file size (in MB) that users can upload.
webApp.MaximumFileSize = 75;

// Persist the changes.
webApp.Update();
```

In common with many SharePoint objects, you must call the **Update** method after making changes to an **SPWebApplication** object. The **Update** method serializes the **SPWebApplication** object and updates the configuration database. Any changes you make will not be applied to the web application if you do not call the **Update** method.

Web application zones

In most cases, a SharePoint web application has a one-to-one relationship with an IIS website. However, you can extend a web application to additional IIS websites. In the SharePoint logical architecture, each additional IIS website is represented by a web application zone. A single web application can be extended to up to five zones. Extending a SharePoint web application to additional zones effectively provides users with additional paths to the same content. Each IIS website provides access to the same set of site collections, but each can have a unique URL, port number, host header, protocol, bindings, and Web.config file.



Note: By default, each web application contains a single zone named **Default**.

In earlier versions of SharePoint, web application zones were used fairly frequently to provide access to different sets of users. For example, you might have configured one zone for Windows authentication to support internal users, and configured another zone for Forms-based authentication to support partner extranet accounts. Web application zones are used less frequently since the advent of claims-based authentication. This is because claims-based authentication enables you to associate multiple claims providers with a single IIS website, making additional zones unnecessary in many cases. However, web application zones are still part of the SharePoint logical architecture and are still used under certain circumstances. For example, you might configure your default zone to allow access over HTTP for users within a corporate network, and use an additional zone to allow access over HTTPS for users outside the domain.

Web application zones are not represented by a class in the object model, because each zone is essentially just a mapping between a web application and an IIS website. When you need to specify a web application zone in code, you use the **SPUrlZone** enumeration to indicate which of the five possible zones you want to work with.

Web applications in SharePoint Online

In a SharePoint Online subscription, web applications are provisioned and managed for you. A SharePoint Online tenancy consists of one or more site collections within an already provisioned web application. You cannot interact with web applications programmatically in solutions for SharePoint Online, nor should you need to.

You cannot interact with web applications from sandboxed solutions or from client-side code, regardless of whether you are deploying your custom functionality to a SharePoint Online subscription or an on-premises deployment of SharePoint 2013. There is no client-side equivalent of the **SPWebApplication** class.

Site Collections and the SPSite Class

As the name suggests, a site collection is a collection of individual sites. Site collections also represent the point in the object hierarchy at which control shifts from farm administrators (who work primarily on the server) to end users (who interact with SharePoint through a web browser). In the SharePoint object model, site collections are represented by the **SPSite** class.



Note: The shift in focus from farm administrators to end users is also reflected in the server-side object model namespace. The classes discussed up to this point all reside in the **Microsoft.SharePoint.Administration** namespace. The **SPSite** class, together with the classes beneath it in the object model hierarchy, resides in the **Microsoft.SharePoint** namespace.

As a SharePoint developer, you will work with site collections on a regular basis. Site collections are significant for several reasons. First, they provide a security boundary in SharePoint deployments. SharePoint groups, users, and various security settings are defined and managed at the site collection level. Although authentication is managed at the web application (or web application zone) level, authorization is managed at the site collection level and below. Many artifacts, such as site columns, content types, master pages, apps, Web Parts, and themes, are typically (if not strictly always) deployed to and scoped at the site collection level.

You can instantiate an **SPSite** object in several different ways. For example, you can:

- Pass a URL to the **SPSite** constructor.
- Enumerate or index the **Sites** collection of the parent **SPWebApplication** object.
- Retrieve the current site collection from the execution context, by using static properties of the **SPContext** class.

- Container for individual sites
- Security boundary
- Deployment scope for many artifacts
- Various ways to instantiate:

```
// Supply a URL.
var site1 = new SPSite("http://team.contoso.com");

// From the parent SPWebApplication instance.
var site2 = webApp.Sites["team.contoso.com"];

// From the execution context.
var site3 = SPContext.Current.Site;
```

The following code example shows how to use various approaches to retrieve an **SPSite** instance:

Instantiating SPSite Objects

```
// Pass a URL to the SPSite constructor.
var site1 = new SPSite("http://team.contoso.com");

// Retrieve an SPSite from the parent SPWebApplication.
var contentService = SPWebService.ContentService;
var webApp = contentService.WebApplications["Contoso Content"];
var site2 = webApp.Sites["team.contoso.com"];

// Retrieve an SPSite from the current execution context.
var site3 = SPContext.Current.Site;

// Dispose of SPSite objects where appropriate after use.
site1.Dispose();
site2.Dispose();
```

Creating **SPSite** objects is memory intensive, and after you finish using an **SPSite** object, you must dispose of it. The next lesson describes how to work with various aspects of **SPSite** objects, including disposal considerations, in more detail.

Site collections in client-side code

Each of the client-side object models for SharePoint 2013 provides a way of working with site collections:

- When you use the JavaScript object model, site collections are represented by the **SP.Site** object.
- When you use the managed client object model, site collections are represented by the **Microsoft.SharePoint.Client.Site** class.
- When you use the REST API, you can access a site collection by constructing a URL in the format `http://[site_collection_URL]/_api/site`, for example `http://team.contoso.com/_api/site`.

Site collections in SharePoint Online

Because site collections are within the tenancy of a SharePoint Online subscription, you can work with site collections programmatically in solutions for SharePoint Online. If you are deploying server-side code, you can work with **SPSite** objects to the extent permitted by the sandboxed solution deployment model. If you are deploying an app, you can use any client-side programming approach to working with site collections.

Individual Sites and the SPWeb Class

In the SharePoint server-side object model, an individual site is represented by the **SPWeb** class.


Almost all the information that end users work with on SharePoint sites is stored in lists of some sort. These range from simple lists, such as tasks lists or announcements lists, through calendars, to document libraries and page libraries. Every list in SharePoint belongs to an individual site. As such, instantiating an **SPWeb** object is very often the first thing you do when you want to programmatically interact with SharePoint

- Container for lists and libraries
- Container for child SPWeb objects
- Every site collection contains one root web
- Various ways to instantiate:

```
// From the parent SPSite instance.
var web1 = site.RootWeb;
var web2 = site.AllWebs["finance"];
var web3 = site.OpenWeb("finance");

// From the execution context.
var web4 = SPContext.Current.Web;
```

content.

 **Note:** The names **SPSite** and **SPWeb** refer to site collections, and sites can be confusing. Whereas farm administrators and end users typically talk in terms of site collections and sites, developers often talk in terms of sites (meaning site collections) and webs (meaning individual sites).

Every **SPSite** object has a root web, which is the **SPWeb** object that shares the URL of the **SPSite** object. As such, an **SPSite** object always contains at least one **SPWeb**. End users (or developers) with sufficient privileges can add additional webs to the site collection, up to a supported limit of 250,000 webs per site collection in most scenarios. Unlike other objects in the SharePoint hierarchy, **SPWeb** objects can act as parents to other **SPWeb** objects. From an end user perspective, this allows users to develop an information architecture consisting of sites, subsites, sub-sites, and so on. In other words, you can structure your maximum of 250,000 webs per site collection into as many hierarchical levels as you want.

You can instantiate an **SPWeb** object in several different ways. For example, you can:

- Access the **RootWeb** property of an **SPSite** object.
- Enumerate or index the **AllWebs** collection of an **SPSite** object.
- Retrieve a specific web by calling the **SPSite.OpenWeb** method.
- Retrieve the current web from the execution context, by using static properties of the **SPContext** class.

The following code example shows how to use various approaches to retrieve an **SPWeb** instance:

Instantiating SPWeb Objects

```
var site = new SPSite("http://team.contoso.com");

// Get the root web for the site collection.
var web1 = site.RootWeb;

// Retrieve a specific web from the parent site collection.
var web2 = site.AllWebs["finance"];
var web3 = site.OpenWeb("finance");

// Retrieve a web from the current execution context.
var web4 = SPContext.Current.Web;

// Dispose of objects where appropriate after use.
web1.Dispose();
web2.Dispose();
web3.Dispose();
site.Dispose();
```

Creating **SPWeb** objects is memory-intensive, and after you finish using an **SPWeb** object, you must dispose of it. **SPWeb** objects are similar to **SPSite** objects in this regard. The next lesson describes how to work with various aspects of **SPWeb** objects, including disposal considerations, in more detail.

Webs in client-side code

Each of the client-side object models for SharePoint 2013 provides a way of working with webs:

- When you use the JavaScript object model, site collections are represented by the **SP.Web** object.
- When you use the managed client object model, site collections are represented by the **Microsoft.SharePoint.Client.Web** class.

- When you use the REST API, you can access a web by constructing a URL in the format `http://[web_URL]/_api/web`, for example `http://team.contoso.com/finance/_api/web`.

Webs in SharePoint Online

From a development perspective, there is little difference between working with webs in a SharePoint Online subscription and working with webs in an on-premises deployment of SharePoint 2013. If you want to use server-side code, your code must work within the restrictions of the sandboxed solution deployment model. If you want to use client-side code and deploy an app, you can work with webs in the same way, regardless of your deployment environment.

Lesson 2

Working with Sites and Webs

Most SharePoint development tasks involve some interaction with sites or webs. Although **SPSite** and **SPWeb** objects are easy to use, there are some nuances that you need to be aware of when you work with these objects. For example, you must manage the life cycle of **SPSite** and **SPWeb** objects carefully to avoid causing problems with memory use. This lesson explains how to perform the most common development tasks involving sites and webs.

Lesson Objectives

After completing this lesson, you will be able to:

- Manage the life cycle of **SPSite** and **SPWeb** objects.
- Retrieve and update properties of **SPSite** and **SPWeb** objects.
- Create and delete sites and webs.

Managing Object Life Cycles

Creating in-memory representations of sites and webs is a resource-intensive process. When you create **SPSite** and **SPWeb** objects in server-side code, your code consumes a large amount of server memory. As such, you need to dispose of these objects when you no longer require them. Failure to dispose of **SPSite** and **SPWeb** objects correctly has historically been a major source of poor performance in custom SharePoint code, often jeopardizing the performance and stability of the wider SharePoint environment and causing symptoms associated with memory leaks.

- **SPSite** and **SPWeb** objects are memory-intensive
- Developers must manage the object life cycle
- Disposal guidelines:
 - If you instantiated the object, dispose of it
 - If you referenced an existing object, do not dispose of it
- Disposal patterns:
 - **try-catch-finally** blocks
 - **using** blocks

The **SPSite** and **SPWeb** classes both implement the **IDispose** interface. As a result, both classes expose a **Dispose** method. Calling this method on an **SPSite** or **SPWeb** instance disposes of the object and releases any associated resources.

When should you dispose of SPSite and SPWeb objects?

Use the following guidelines to determine when to dispose of **SPSite** and **SPWeb** objects:

- If you create a new instance of **SPSite** or **SPWeb**, dispose of the object when you no longer need it.
- If you set an object variable to an existing instance of **SPSite** or **SPWeb**, for example by using **SPContext.Current.Site** or **SPContext.Current.Web**, you should not dispose of the object.

It is important to not dispose of **SPSite** and **SPWeb** instances that you have not created, such as objects retrieved through static properties of the **SPContext** class. Disposing of these objects can cause unexpected behaviors in your code.



Additional Reading: Disposal patterns for SharePoint objects form a broad subject area, and there are various nuances beyond those discussed in this module. For more information, see *Best Practices: Using Disposable Windows SharePoint Services Objects* at <http://go.microsoft.com/fwlink/?LinkID=306778>. Although this article was written for SharePoint 2007, the principles it describes apply equally to SharePoint 2010 and SharePoint 2013.

There are two main patterns you can use to dispose of **SPSite** and **SPWeb** objects: disposal within **try-catch-finally** blocks, and disposal within **using** statements.

Disposal within try-catch-finally blocks

At a high level, the life cycle of an **SPSite** or **SPWeb** object should resemble the following:

- Create a new **SPSite** or **SPWeb** object.
- Use the **SPSite** or **SPWeb** object to perform some action.
- Dispose of the **SPSite** or **SPWeb** object.

However, suppose that your actions in step 2 result in an exception. How do you ensure that the **SPSite** or **SPWeb** object is still disposed of correctly? The answer is to dispose of the object in a **finally** block.

The following code example shows how to properly dispose of **SPSite** and **SPWeb** objects within **try-catch-finally** blocks:

Disposal Within Try-Catch-Finally Blocks

```
SPSite site = null;
SPWeb web = null;

try
{
    site = new SPSite("http://team.contoso.com");
    web = site.OpenWeb();

    // Attempt to perform some actions here.
}
catch (Exception ex)
{
    // Handle any exceptions.
}
finally
{
    // This code is called regardless of whether the actions succeeded or failed.
    if (web != null)
        web.Dispose();

    if (site != null)
        site.Dispose();
}
```

Disposal within using blocks

An alternative to the **try-catch-finally** approach is to instantiate **SPSite** and **SPWeb** objects within **using** statements. Any objects you create within a **using** statement are disposed of automatically when execution leaves the associated block of code, either through logical completion of the block of code or because an error has been thrown.

The following code example shows how to use nested using blocks to manage the life cycle of an **SPSite** object and an **SPWeb** object:

Disposal Within Using Blocks

```
using (var site = new SPSite("http://team.contoso.com"))
{
    using (var web = site.OpenWeb())
    {
        // Attempt to perform some actions here.
    }
}
```

Just like you should avoid calling the **Dispose** method on objects that you have not created, you should not reference objects you have not created in **using** statements. For example, you should never enclose statements such as `var site = SPContext.Current.Site` in a using statement.



Note: The disposal considerations described in this topic do not apply to the client-side equivalents of the **SPSite** and **SPWeb** classes.

Retrieving and Updating Properties

Retrieving and updating properties is among the most fundamental programming tasks for many platforms, and SharePoint 2013 is no exception. Many development activities require you to retrieve or update the properties of **SPSite** and **SPWeb** objects.

Retrieving properties from sites and webs

The **SPSite** class and the **SPWeb** class expose a broad range of strongly-typed properties. These include simple, single-value properties and collection-type properties. Retrieving these properties is just like retrieving properties for any other .NET object.

The following code example shows how to retrieve a simple property and a collection-type property from an **SPWeb** instance:

Retrieving Properties from an SPWeb Instance

```
SPWeb web = SPContext.Current.Web;

// Retrieve a simple value-type property.
string title = web.Title;

// Retrieve a collection-type property.
SPListCollection lists = web.Lists;
foreach (SPList list in lists)
{
    // Process each list instance in the web.
}
```

• Retrieving properties

```
SPWeb web = SPContext.Current.Web;

// Retrieve a simple property.
string title = web.Title;

// Retrieve a collection property.
SPListCollection lists = web.Lists;
foreach (SPList list in lists) { ... }
```


• Updating properties

```
// Update various properties.
web.Title = "New Title";
web.Description = "A brand new description.";

// Write the changes to the content database.
web.Update();
```

Because of the hierarchical nature of the SharePoint logical architecture, many SharePoint classes include properties that expose collections of child objects. In the previous example, you can see that the **SPWeb.Lists** property returns an object of type **SPListCollection**, which is a collection of **SPList** objects—or in other words, all the lists within the current web. The *typeCollection* naming convention is used throughout SharePoint. For example:


- The **SPWebService.WebApplications** property returns an object of type **SPWebApplicationCollection**, which is an enumerable collection of **SPWebApplication** objects. This represents all the web applications that run under the **SPWebService** instance.
- The **SPWebApplication.Sites** property returns an object of type **SPSiteCollection**, which is an enumerable collection of **SPSite** objects. This represents all the site collections within the web application (the **SPSiteCollection** class does not represent a site collection!).
- The **SPSite.AllWebs** property returns an object of type **SPWebCollection**, which is an enumerable collection of **SPWeb** objects. This represents all the webs within the current site collection, regardless of whether they are root webs, subwebs, sub-subwebs, and so on.

 **Note:** Although you can use these enumerable collections to programmatically walk through many aspects of a SharePoint deployment, enumerating these collections is often an expensive process and there are often more efficient ways to reach the items you need. Efficient approaches to specific tasks are described in more detail later in this course.

Updating properties of sites and webs

You can set properties on **SPSite** and **SPWeb** objects in the same way that you set properties on any other .NET objects. Most properties of the **SPSite** and **SPWeb** classes are writeable. However, one key difference is that you must call the **Update** method after making any changes to the properties of an **SPWeb** object.

When you set properties on an **SPWeb** object, you are making changes to an in-memory representation of the SharePoint site. The **Update** method writes any changes you have made since instantiating the object to the database.

 **Note:** Many classes in the SharePoint object model have an **Update** method, including the **SPWebApplication** class, the **SPList** class, and the **SPListItem** class. In each case, the **Update** method writes changes to a database. In the case of **SPWebApplication**, the **Update** method writes changes to the farm configuration database. In the case of **SPWeb**, **SPList**, and **SPListItem**, the **Update** method writes changes to the content database. The **SPSite** class does not have an **Update** method. Any changes you make to an **SPSite** object are applied to the underlying site collection immediately.

The following code example shows how to update the properties of an **SPWeb** object:

Updating Properties of an SPWeb Instance

```
SPWeb web = SPContext.Current.Web;

// Update several properties.
web.Title = "New Title";
web.Description = "A brand new description.";

// Write the changes to the content database.
web.Update();
```

Retrieving and updating properties in Windows PowerShell

In some cases, you may find it more convenient to work with site and web properties in Windows PowerShell. You can use the **Get-SPSite** and **Get-SPWeb** cmdlets to retrieve **SPSite** and **SPWeb** instances respectively. You can then use object model methods or properties to make changes to the instances. Alternatively, you can use the **Set-SPSite** and **Set-SPWeb** cmdlets to configure various site and web properties.

Demonstration: Updating Properties

The instructor will now demonstrate briefly how to update the properties of an **SPWeb** object by using the server-side SharePoint object model.

Demonstration Steps

- Start the 20488B-LON-SP-02 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In a File Explorer window, browse to **E:\Democode\UpdatingProperties**, and then double-click **UpdatingProperties.sln**.
- If the **How do you want to open this type of file (.sln)?** dialog box appears, click **Visual Studio 2012**.
- In Visual Studio, in Solution Explorer, expand the **UpdatingProperties** project node, expand **UpdatingPropertiesWebPart**, and then double-click **UpdatingProperties.ascx**.
- At the bottom of the center pane, click **Design**.
- Observe that the user control consists of a simple UI with a text box named **txtTitle**, a text box named **txtDescription**, and a button named **btnUpdate**.
- On the design surface, double-click the **Update** button to generate the click event handler for the button.
- In the **btnUpdate_Click** method, type the following code to retrieve the current **SPWeb** instance, and then press Enter:

```
var web = SPContext.Current.Web;
```

- Type the following code to set the title of the web, and then press Enter:

```
web.Title = txtTitle.Text;
```

- Type the following code to set the description of the web, and then press Enter:

```
web.Description = txtDescription.Text;
```

- Type the following code to write the changes to the database, and then press Enter:

```
web.Update();
```

- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, after the page finishes loading, on the **PAGE** tab, click **Edit**.

- On the **INSERT** tab, click **Web Part**.
- In the **Categories** list, click **Contoso Web Parts**.
- In the **Parts** list, click **Property Updates**, and then click **Add**.
- On the **PAGE** tab, click **Save**.
- In the **Property Updates** Web Part, in the **Site title** box, type a new site title.
- In the **Site description** box, type a new site description, and then click **Update**.
- Verify that the page reloads with the updated site title.
- Close Internet Explorer, and then close Visual Studio 2012.

Creating and Deleting Sites and Webs

Most of the time, your activities as a SharePoint developer will involve working with existing sites and webs rather than creating or deleting them. However, there are scenarios where you may want to create or delete sites or webs programmatically. One relatively common scenario is providing a custom solution that streamlines the creation of sites and webs by capturing any additional information from the site creator, such as site metadata, group members, and so on.

Creating sites and webs

To create a new **SPSite** or **SPWeb** object, you must call the **Add** method on an appropriate collection object:

- To create a new **SPSite**, you must call the **SPSiteCollection.Add** method.
- To create a new **SPWeb**, you must call the **SPWebCollection.Add** method.

Where you retrieve the appropriate collection object from will determine where your new object is created in the SharePoint information architecture. In both cases, the **Add** method returns a new instance of the object you are adding. This enables you to set properties or perform additional actions on the new object.

The following code example shows how to create a new **SPSite**:

Creating an SPSite

```
// Retrieve the web application that will host the new site collection.
var contentService = SPWebService.ContentService;
var webApp = contentService.WebApplications["Contoso Content"];

// Create a new site collection beneath the web application.
SPSite site = webApp.Sites.Add("/sites/finance", @"CONTOSO\Administrator",
"administrator@contoso.com");

// Perform any additional actions on the new SPSite object.
site.OutgoingEmailAddress = "finance@sharepoint.contoso.com";
```

- Creating sites and webs
 - Call the **Add** method on a collection object

```
SPSite site = webApp.Sites.Add("/sites/finance",
@"CONTOSO\Administrator",
"administrator@contoso.com");
SPWeb web = site.AllWebs.Add("project1");
```

- Deleting sites and webs
 - Call the **Delete** or **Recycle** method on the object
 - You must still dispose of the object properly

```
SPWeb web = site.OpenWeb("project1");
web.Delete();
web.Dispose();
```

When you create a new **SPSite** object, you must specify the following information as a minimum:

- The server-relative URL for the new site collection
- The user name of the site collection administrator
- The email address of the site collection administrator

The **SPSiteCollection.Add** method includes 12 overloads that enable you to specify a range of additional information, such as the Locale ID (LCID), the site template, and the content database to which you want to add the site collection.

When you create a new web, the location of the **SPWebCollection** on which you call the **Add** method will determine where the new web appears in the site hierarchy. For example, if you want to add a top-level web, you call the **Add** method on the **AllWebs** property of an **SPSite** object. If you want to add a subweb to an existing web, you call the **Add** method on the **Webs** property of an **SPWeb** object.

The following example shows how to create a new **SPWeb**:

Creating an SPWeb

```
// Add a new top-level web to a site collection.
SPSite site = new SPSite("http://team.contoso.com");
SPWeb web1 = site.AllWebs.Add("project1");

// Call the Update method before you use the new web.
web1.Update();

// Add a new sub-web to an existing web.
SPWeb web2 = web1.Webs.Add("project1/phase1");
web2.Update();
```

When you create a new **SPWeb** object, you must specify the site-relative URL as a minimum. You can also specify various other properties, such as the title, description, LCID, site template, and whether the new web should inherit permissions from its parent object.

Deleting sites and webs

Deleting sites and webs is straightforward. Both the **SPSite** class and the **SPWeb** class include a **Delete** method that you can call to delete the site or web. Be aware that even after deleting a site or web, you must still dispose of the object properly.

The following code example shows how to delete **SPSite** and **SPWeb** objects:

Deleting Sites and Webs

```
// Delete an SPSite object.
using(SPSite site = new SPSite("http://team.contoso.com/sites/finance"))
{
    site.Delete();
}

// Delete an SPWeb object.
using(SPSite site = new SPSite("http://team.contoso.com"))
{
    SPWeb web = site.OpenWeb("project1");
    web.Delete();
    web.Dispose();
}
```

If you delete an **SPSite** object, any webs that belong to the site will also be deleted. However, if you attempt to delete an **SPWeb** object that contains child **SPWeb** objects, SharePoint will throw an exception of type **SPEXception**. You must delete any child webs before you attempt to delete a parent web.

The **SPWeb** object also provides the **Recycle** method, which enables you to send the website to the recycle bin, meaning it can be restored or permanently deleted later.

Creating and deleting sites and webs in Windows PowerShell

As you might expect by now, you can also use Windows PowerShell to create and delete sites and webs. You can use the **New-SPSite** and **New-SPWeb** cmdlets to create sites and webs respectively. You can also use the **Remove-SPSite** and **Remove-SPWeb** cmdlets to delete existing sites and webs.

Lab A: Working with Sites and Webs

Scenario

The management team at Contoso has complained that project sites across the organization use a variety of different naming conventions. They have asked you to find an efficient way to update several site titles. In this lab, you will prototype two different approaches. First, you will use a visual Web Part to enumerate sites and enable users to update site properties. You will then experiment with performing the same procedure in Windows PowerShell.

Objectives

After completing this lab, you will be able to:

- Retrieve and update SharePoint objects in managed code.
- Retrieve and update SharePoint objects in Windows PowerShell.

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-02
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

Exercise 1: Working with Sites and Webs in Managed Code

Scenario

In this exercise, you will develop a Web Part that displays a list of the webs in the current site collection, enables users to select a web from the list, and then enables users to update the title of the selected web. The UI has been created for you. Your task is to add code to make the Web Part functional.

The main tasks for this exercise are as follows:

1. Create Local Variables
2. Display a List of Webs in the Current Site Collection
3. Respond to a User Selecting a Web
4. Update Web Titles When the User Clicks a Button
5. Test the Web Part

► Task 1: Create Local Variables

- Start the 20488B-LON-SP-02 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password Pa\$\$w0rd.
- In Visual Studio 2012, open the solution at **E:\Labfiles\Starter\TitleChecker_LabA\TitleChecker.sln**.
- Review the contents of the **TitleCheckerWebPart.ascx** user control. Notice that the user control contains:
 - A list box control named **lstWebs**.
 - A panel control named **pnlUpdateControls**, which contains a text box named **txtTitle** and a button named **btnUpdate**.
 - A panel control named **pnlResult**, which contains a literal control named **litResult**.
- Review the code-behind file for the user control. Notice that the code-behind file contains:
 - A method that handles the **SelectedIndexChanged** event of the **lstWeb** control.

- A method that handles the **Click** event of the **btnUpdate** control.
- Within the **TitleCheckerWebPart** class, add a variable of type **Guid** named **selectedSiteGuid**. When the user selects a web from the list box, you will use this variable to track the selected item.
- Add a Boolean variable named **siteUpdated**. You will use the variable to track whether a user has changed the title of a site, and adjust the rendering of the control accordingly.

► Task 2: Display a List of Webs in the Current Site Collection

- In the **OnPreRender** method, add code to hide the **pnlUpdateControls** and **pnlResult** controls.
- Add code to clear the contents of the **lstWebs** list box.
- For each web in the current site collection, add a list item to the **lstWebs** list box. The list item text should be the title of the web, and the list item value should be the ID of the web.



Best Practice: Within your **foreach** loop, use a **try-finally** block to dispose of each **SPWeb** object properly after use.

► Task 3: Respond to a User Selecting a Web

- In the **SelectedIndexChanged** handler for the **lstWebs** list box, retrieve the GUID value of the selected web.
- Set the **selectedSiteGuid** variable to the retrieved GUID value.
- Set the **Text** property of the **txtTitle** text box to the title of the selected web.
- In the **OnPreRender** method, after the code that populates the **lstWebs** list box, check whether the **selectedSiteGuid** variable contains a non-empty GUID value.
- If the **selectedSiteGuid** variable contains a non-empty GUID value:
 - a. Ensure that the same item in the **lstWebs** list box is selected when the page is rebuilt.
 - b. Make the **pnlUpdateControls** panel visible.

► Task 4: Update Web Titles When the User Clicks a Button

- In the **Click** handler for the **btnUpdate** button, retrieve the GUID value of the selected web.
- Set the **selectedSiteGuid** variable to the retrieved GUID value.
- Create a local string variable named **newTitle**, and set it to the **Text** property of the **txtTitle** text box.
- If the **newTitle** variable is not a null or empty string, and if the **selectedSiteGuid** variable is not an empty GUID, perform the following actions:
 - a. Change the title of the selected web to the value of the **newTitle** string.
 - b. Display a message confirming the changes you have made in the **litResult** literal control.
 - c. Set the value of the **siteUpdated** variable to **true**.
- In the **OnPreRender** method, after the code that populates the **lstWebs** list box but before the code that checks for a selected item, check whether the **siteUpdated** variable is **true**.
- If **siteUpdated** is **true**, perform the following actions:
 - a. Clear any selection in the **lstWebs** list box.
 - b. Set the **selectedSiteGuid** variable to an empty GUID.
 - c. Make the **pnlResult** panel visible.

► Task 5: Test the Web Part

- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, after the page finishes loading, add the **Title Checker** Web Part to the page.



Note: Unless you have modified the feature manifest, you can find the **Title Checker** Web Part in the **Contoso Web Parts** category.

- In the **Title Checker** Web Part, in the list box, select any web.
- Verify that the Web Part displays a text box containing the title of the selected web, together with an **Update** button.
- In the text box, type a new title, and then click **Update**.
- Verify that:
 - a. The Web Part displays a message confirming the action taken.
 - b. The update controls (the text box and the **Update** button) are no longer visible.
 - c. The new web title is reflected in the list of webs.
- Close all open windows.

Results: After completing this exercise, you should have developed and tested a visual Web Part that enables users to select and update web titles.

Exercise 2: Working with Sites and Webs in Windows PowerShell.

Scenario

In this exercise, you will create a Windows PowerShell script that enables users to update the titles of webs within a site collection. Your script will first prompt users to provide a site collection URL. If the URL is valid, the script will then enumerate the webs in the site collection and give the user the opportunity to make changes.



Note: If you are not familiar with scripting in Windows PowerShell, you can review the sample solution script at **E:\Labfiles\Solution>TitleUpdater.ps1**.

The main tasks for this exercise are as follows:

1. Create a Windows PowerShell Script to Retrieve and Update Web Titles
2. Test the Windows PowerShell Script

► Task 1: Create a Windows PowerShell Script to Retrieve and Update Web Titles

- Open Windows PowerShell ISE and create a new script.
- In your script, add code to clear the command window when the script runs.
- Add code to load the SharePoint Windows PowerShell snap-in.
- Add code to prompt the user for a site collection URL.

- Attempt to retrieve an **SPSite** object using the URL supplied by the user. Verify that the **SPSite** object is not null before continuing.
- If you have successfully created an **SPSite** object, use a **foreach** loop to enumerate the webs in the specified site.
- Within the **foreach** loop:
 - a. Display the title of the current web.
 - b. Ask the user whether they want to edit the current web title (Y/N).
 - c. If the user wants to edit the current web title, prompt them for a new title and update the title of the current web accordingly.
 - d. If the user has updated the current web title, display a message summarizing the change before continuing.



Best Practice: When you ask the user whether they want to edit the current web title, make **No** the default answer. If the user does not want to edit the current web title, they should be able to skip the current web simply by pressing Enter.

- Save your script in the **E:\Labfiles\Starter** folder as **TitleUpdater.ps1**.
- Task 2: Test the Windows PowerShell Script
- Run the **TitleUpdater.ps1** script.
 - In the command window, when you are prompted to provide a site URL, type **http://projects.contoso.com**, and then press Enter.
 - When the command window displays the title of the first web, press Enter repeatedly. Verify that the script moves on to the next web every time you press Enter.
 - Continue to press Enter until the script finishes running, and then click **Run Script** again.
 - When you are prompted to provide a site URL, type **http://projects.contoso.com**, and then press Enter.
 - When the command window displays the title of the first web, type **y**, and then press Enter.
 - Type a new title, and then press Enter. Verify that the command window displays a confirmation message before moving on to the next web in the collection.
 - For each remaining web in the collection, you can either type **y** to edit the web title or press Enter to skip the web.

Results: After completing this exercise, you should have created a Windows PowerShell script that enables users to update the title of each web within a site collection.

Question: Which approach would you recommend to the management team: the visual Web Part or the Windows PowerShell script? Why?

Question: What happens if a user with insufficient permissions loads the Web Part? How would you work around any issues caused by insufficient permissions?

Question: Enumerating sites and webs is computationally expensive. Why is this particularly problematic in a Web Part?

Lesson 3

Working with Execution Contexts

When you create a custom solution for a SharePoint deployment, your code will often be invoked by a variety of different users. For example, if you deploy code in a custom Web Part, your code will be invoked by every user who requests a page that contains your Web Part. These users may represent a wide range of permission levels, from site collection administrators to visitors with read-only permissions. In these scenarios, you need to consider how you want your code to respond to users with varying levels of privileges. To do this, you need to understand the SharePoint execution context, and how you can work with the execution context to adapt your solution to different users.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain what is meant by the term "SharePoint context".
- Describe how to adapt content for different user permissions.
- Describe how to run code using elevated permissions.

Understanding the SharePoint Context

If you create SharePoint code that runs in the context of an HTTP request, such as within a Web Part or an application page, you can access a variety of contextual information about the request. SharePoint makes this information available through an object of type **SPContext**. This is often referred to as the *SharePoint context*. You can use the **SPContext** object to determine the current site collection, the current web, the current user, and various other pieces of information relating to the HTTP request. Where available, you can retrieve an **SPContext** instance that represents the current execution context from the static **SPContext.Current** property.

The following code example shows how to retrieve various pieces of information from the SharePoint context:

Working with the SharePoint Context

```
// Retrieve the current site collection
SPSite site = SPContext.Current.Site;

// Retrieve the current web
SPWeb web = SPContext.Current.Web;

// Retrieve the current user
SPUser user = SPContext.Current.Web.CurrentUser;
```

As you might expect, the information you can access through the SharePoint context depends on what the associated HTTP request relates to. For example, if the user was performing an action on a list item, the current **SPContext** instance would include information about the current list, the current list item, and so on. By contrast, if your code runs within a Web Part on the site home page, any **SPContext** properties relating to lists or list items will return null or empty values.

- **SPContext** object
- Represents context of current HTTP request
- Provides a range of information:


```
SPSite currentSite = SPContext.Current.Site;
SPWeb currentWeb = SPContext.Current.Web;
SPUser currentUser = SPContext.Current.Web.CurrentUser;
```
- Only available when your code is invoked synchronously by an HTTP request



Best Practice: Remember that you should not dispose of an **SPSite** or **SPWeb** object that you have retrieved from **SPContext.Current.Site** or **SPContext.Current.Web** respectively.

When is the SharePoint context available?

You can access the current SharePoint context through the **SPContext.Current** property whenever your code is invoked synchronously by an HTTP request. If your code runs when a user interacts with a Web Part, or clicks a button on the ribbon, or loads an application page, you can retrieve the current SharePoint context.

Some types of SharePoint solution are not associated with an HTTP request. For example, feature receivers and timer jobs are not invoked by HTTP requests. In these types of solutions, a SharePoint context object would be meaningless. As such, the **SPContext.Current** property returns null.

Working with Users and Permissions

If you create code that will run using the identity of the current user—for example, code in a Web Part or in an application page—you must consider how your solution will respond to users with different permission levels. In particular, you must consider how to adapt your code for users who do not have permission to perform particular actions. There are two main ways you can approach this requirement: you can check the permissions of the user programmatically, or you can leverage the SharePoint security-trimming functionality to render content only to users who hold specific privileges.

• Verifying permissions programmatically

```
var web = SPContext.Current.Web;
if (web.DoesUserHavePermissions(
    SPBasePermissions.ManageWeb))
{
    // Perform the update operation.
}
```

• Using security trimming

```
<SharePoint:SPSecurityTrimmedControl
    runat="server"
    PermissionsString="ManageWeb">
    <!-- Add child controls here -->
</SharePoint:SPSecurityTrimmedControl>
```



Note: SharePoint includes a comprehensive authorization framework, and later modules describe authorization and permissions management in more detail. This topic focuses on simple techniques for adapting solutions to different user permissions.

Programmatically verifying permissions

SharePoint users are represented by the **SPUser** class. Regardless of how your users authenticate to the SharePoint platform, SharePoint will always generate an **SPUser** instance to represent the signed-in user (this process is often referred to as *identity normalization*). When your code runs in the SharePoint context, your code is running in the context of a specific **SPUser**.

The **SPUser** class is a key component of the SharePoint identity management and authorization functionality, and later in this course you will see how to work with **SPUser** instances in a variety of ways. However, SharePoint classes also include various methods that implicitly use the current **SPUser** instance to check permissions. For example, many SharePoint classes, such as **SPWeb** and **SPList**, include a method named **DoesUserHavePermissions**. This method provides an easy way to create conditional logic based on the permissions of the current user.

The following code example shows how to use the **DoesUserHavePermissions** method to programmatically verify permissions:

Programmatically Verifying Permissions

```
protected void UpdateWebs()
{
    var web = SPContext.Current.Web;
    // Test whether the current user has the ManageWeb permission.
    if(web.DoesUserHavePermissions(SPBasePermissions.ManageWeb))
    {
        // Perform the update operation.
    }
    else
    {
        // Advise the user that they do not have sufficient permissions.
    }
}
```

SharePoint permissions are defined by the **SPBasePermissions** enumeration. If you need to specify multiple permissions, you can use bitwise combinations of **SPBasePermissions** values. For example, to specify that the user must hold both the **ManageWeb** permission and the **ManageSubWebs** permission, you would combine the permissions in the form (**SPBasePermissions.ManageWeb & SPBasePermissions.ManageSubWebs**).

Using security trimming

Security trimming describes the process of filtering rendered content according to the permissions of the current user. SharePoint uses security trimming extensively; for example, to remove search results that the current user does not have permission to view. You can leverage this security trimming functionality to provide a similar experience in your custom solutions.

The SharePoint object model includes an ASP.NET container control named **SPSecurityTrimmedControl**. This control enables you to specify the permissions that a user must hold in order to view to contents of the control. Any content and child controls that you add within the **SPSecurityTrimmedControl** are only rendered if the user holds the specified permissions.

The following example shows how to use the **SPSecurityTrimmedControl** to filter content according to the permissions of the current user:

Using Security Trimming

```
<SharePoint:SPSecurityTrimmedControl runat="server" PermissionsString="ManageWeb">
  <!-- This content is only rendered for users who hold the ManageWeb permission-->
</SharePoint:SPSecurityTrimmedControl>
```

It is important to understand that the **SPSecurityTrimmedControl** does not simply control the visibility of rendered HTML content. This would not be a secure approach, because a malicious user could simply edit the HTML. If the user does not hold the permissions specified by the **SPSecurityTrimmedControl**, ASP.NET will not generate any output for the content within the control.

Discussion: Adapting Content for Different User Permissions

The instructor will now lead a brief discussion around the following questions:

- When should you use code to adapt content to the permissions of the current user?
- When should you use the **SPSecurityTrimmedControl** to adapt content to the permissions of the current user?

- When should you use code to adapt content to the permissions of the current user?
- When should you use the **SPSecurityTrimmedControl** to adapt content to the permissions of the current user?

Manipulating the Execution Context

There are times when you may want your SharePoint code to run using the identity of someone other than the current user. For example, you may want your code to perform actions that the current user does not have permissions to perform. The server-side SharePoint object model includes utility methods that you can use to manipulate the execution context in these scenarios. You can use the **SPSecurity.RunWithElevatedPrivileges** method to invoke code that requires permissions beyond those held by the current user.

- Use **SPSecurity.RunWithElevatedPrivileges** to run code using the system account

```
var delegateDPO = new
    SPSecurity.CodeToRunElevated(DoPrivilegedOperation);
SPSecurity.RunWithElevatedPrivileges(delegateDPO);

private void DoPrivilegedOperation()
{
    // This method will run with elevated privileges.
}

SPSecurity.RunWithElevatedPrivileges(delegate()
{
    // This code will run with elevated privileges.
});
```



Note: Later modules in this course describe more comprehensive ways of manipulating the execution context, such as running code under the identity of a specific user.

Understanding elevated privileges

If your code runs in a farm solution within the SharePoint context, your code is executed by an IIS worker process (w3wp.exe). This process runs using the application pool identity associated with the SharePoint web application. ASP.NET applications, including SharePoint, use impersonation by default. This means that the application pool identity will impersonate the current user to execute code. As a result, your code can only perform actions that are permitted by the permission set of the current user.

When you use **SPSecurity.RunWithElevatedPrivileges** to invoke code, the worker process reverts to executing code using the application pool identity, rather than the identity of the current user. The application pool identity has full-trust permissions on the SharePoint web application. Consequently, your code is no longer restricted by the permission set of the current user.

Running code with elevated privileges is only applicable to code that runs within the SharePoint context. Code that runs without a SharePoint context—for example, code within a timer job—runs under a process identity rather than a user identity. In these scenarios, elevating privileges would have no effect, because the process is not impersonating the identity of a user.



Note: You can only run code with elevated privileges within a farm solution. Sandboxed solution code is executed within an isolated worker process, and apps do not execute any server-side code.

Invoking code with elevated privileges

You can use the **SPSecurity.RunWithElevatedPrivileges** method to invoke code in two ways:

- You can use an **SPSecurity.CodeToRunElevated** delegate to run a specific method with elevated privileges.
- You can use an anonymous delegate to run a block of code with elevated privileges.

The following code example shows how to use the **SPSecurity.CodeToRunElevated** delegate to invoke code with elevated privileges:

Using the **SPSecurity.CodeToRunElevated** Delegate to Run Code with Elevated Privileges

```
var siteGuid = SPContext.Current.Site.ID;
var delegateDPO = new SPSecurity.CodeToRunElevated(DoPrivilegedOperation);
SPSecurity.RunWithElevatedPrivileges(delegateDPO);

private void DoPrivilegedOperation()
{
    // This method will run with elevated privileges.
    using (var site = new SPSite(siteGuid))
    {
        // Perform the privileged operation here.
    }
}
```

First, you construct the **SPSecurity.CodeToRunElevated** delegate by supplying the name of the method you want to run with elevated privileges. The method must take no parameters and return void. Next, you call the **SPSecurity.RunWithElevatedPrivileges** method, supplying your delegate as an argument.

In the example, notice that the code that runs with elevated permissions does not retrieve anything from the **SPContext** object. Instead, the GUID of the current site is stored in a global variable, and the site GUID is used to reconstruct the **SPSite** object within the elevated code. This is necessary because the SharePoint context is closely associated with the current user. Within the elevated code, if you attempt to use an **SPSite** object or an **SPWeb** object that you have retrieved from the current context, the elevated code will revert to running under the identity of the current user.

If you want to run elevated code inline rather than encapsulated in a method, you can use an anonymous delegate to invoke the elevated code. The same constraints apply: you cannot pass arguments to the elevated code, and you cannot use any objects retrieved from the current SharePoint context within the elevated code.

The following code example shows how to use an anonymous delegate to invoke code with elevated privileges:

Using an Anonymous Delegate to Run Code with Elevated Privileges

```
var siteGuid = SPContext.Current.Site.ID;
SPSecurity.RunWithElevatedPrivileges(delegate()
{
    // This code will run with elevated privileges.
    using (var site = new SPSite(siteGuid))
    {
        // Perform the privileged operation here.
    }
});
```


Lab B: Working with Execution Contexts

Scenario

In this lab, you will modify the visual Web Part you created in the previous exercise to make sure it renders correctly for all users. You want to display the list of webs to all users, so you will adapt the code that populates the list box to run with elevated privileges. However, you only want the update controls to be visible to users who have the permissions required to update web titles, so you will wrap the update controls in an **SPSecurityTrimmedControl** element.

Objectives

After completing this lab, you will be able to:

- Run server-side SharePoint code with elevated privileges.
- Use SharePoint controls to show or hide content based on user permissions.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-02
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

Exercise 1: Running Code with Elevated Privileges

Scenario

Enumerating the webs in a site collection is a computationally expensive process. As such, only highly-privileged users can run code that enumerates webs. To enable all users to see the list of webs, you must use elevated privileges to run the code that populates the list of webs.

The main tasks for this exercise are as follows:

1. Test the Web Part as a Site Member
2. Modify Code to Run with Elevated Privileges
3. Test the Web Part as a Site Member

► Task 1: Test the Web Part as a Site Member

- Open Internet Explorer and browse to **http://projects.contoso.com**.
- When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.
- Notice that you are denied access to the home page, despite being a site member.



Note: Only users who are granted the **ManageWeb** permission can enumerate the webs in a site collection. This permission is granted by the **Full Control** permission level.

- Close Internet Explorer.

► Task 2: Modify Code to Run with Elevated Privileges

- In Visual Studio 2012, open the solution at **E:\Labfiles\Starter\TitleChecker_LabB\TitleChecker.sln**.
- Open the code-behind file for the **TitleCheckerWebPart** user control.
- Within the **TitleCheckerWebPart** class, add a variable of type **Guid** named **siteCollID**.


- In the **OnPreRender** method, locate the following line of code:

```
var site = SPContext.Current.Site;
```

- Replace this line of code with a statement that sets the **siteCollID** variable to the GUID identifier of the current **SPSite** object.
- Extract the code that populates the **IstWebs** list box into a new method named **PopulateWebsList**.
- Add code that calls the **PopulateWebsList** method such that it runs with elevated privileges.
- At the start of the **PopulateWebsList** method, within a **using** statement, instantiate a new **SPSite** object named **site** by using the **siteCollID** identifier. The **using** code block should encapsulate all of the functionality in the **PopulateWebsList** method.
- Rebuild the solution, and verify that your code builds without errors.
- Deploy the solution.
- Close Internet Explorer.

► Task 3: Test the Web Part as a Site Member

- Open Internet Explorer and browse to **http://projects.contoso.com**.
- When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.
- When the home page loads, verify that the **Title Checker** Web Part now displays the list of webs correctly.
- In the **Title Checker** Web Part, in the list box, select any web.
- In the text box, type a new title, and then click **Update**.
- Notice that nothing happens. This is because the call to **SPWeb.Update** in the **btnUpdate_Click** method is throwing an exception of type **UnauthorizedAccessException**.

 **Note:** If you want, you can verify the exception by attaching the Visual Studio debugger to the w3wp.exe process. However, the debugger runs very slowly within the constraints of the classroom environment.

- Close Internet Explorer.

Results: After completing this exercise, you should have configured the Title Checker Web Part to use elevated privileges to populate the list of webs.

Exercise 2: Adapting Content for Different User Permissions

Scenario

Although you want all users to be able to view the list of webs in the Title Checker Web Part, you do not want all users to be able to update web titles. It would be inappropriate to run the update logic with elevated privileges. Instead, you will use a security-trimmed control to prevent ASP.NET from rendering update controls to users with insufficient permissions.

The main tasks for this exercise are as follows:

1. Modify the Web Part to Hide Controls from Unauthorized Users
2. Test the Web Part As a Site Member

► Task 1: Modify the Web Part to Hide Controls from Unauthorized Users

- In Visual Studio, open the **TitleCheckerWebPart** user control in source view.
- Add a **SharePoint:SPSecurityTrimmedControl** element that encapsulates the **pnlUpdateControls** panel.
- Set the **PermissionsString** attribute of the security-trimmed control to **ManageWeb**.



Note: **ManageWeb** is the object model name for the **Manage Web Sites** permission, as defined by the **SPBasePermissions** enumeration.

- Rebuild the solution, and verify that your code builds without errors.
- On the **DEBUG** menu, click **Start Without Debugging**.
- When you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, when the page loads, in the **Title Checker** Web Part, in the list box, select any web.
- In the text box, type a new title, and then click **Update**.
- Verify that the Web Part updates the title.



Note: Because you are testing the Web Part as an administrator, the user experience should be unchanged at this point.

- Close Internet Explorer.

► Task 2: Test the Web Part As a Site Member

- Open **Internet Explorer** and browse to **http://projects.contoso.com**.
- When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.
- In Internet Explorer, when the home page loads, verify that the **Title Checker** Web Part now displays the list of webs correctly.
- In the **Title Checker** Web Part, in the list box, select any web.
- Verify that the Web Part does not display the update controls.



Note: Because you are testing the Web Part as a regular site member, the **SPSecurityTrimmedControl** prevents ASP.NET from rendering the update controls.

Results: After completing this exercise, you should have configured the Title Checker Web Part to use a security-trimmed control to hide the update controls from users with insufficient permissions.

Module Review and Takeaways

In this module, you learned about how to work with core classes in the server-side SharePoint object model. You learned about the hierarchical nature of the object model, and about how object model classes relate to the logical architecture of a SharePoint deployment. You learned about how to manage the life cycle of **SPSite** and **SPWeb** objects, and how to retrieve and update properties. You also gained a broad understanding of the SharePoint execution context and how to create solutions that respond to different user permission levels.

Review Question(s)

Test Your Knowledge

Question	
You want to verify programmatically whether the Managed Metadata Service is running on any servers in the local SharePoint farm. Which of the following classes represents a service running on a server?	
Select the correct answer.	
<input type="checkbox"/>	SPApplicationPool
<input type="checkbox"/>	SPService
<input type="checkbox"/>	SPServiceInstance
<input type="checkbox"/>	SPServiceApplication
<input type="checkbox"/>	SPServiceApplicationProxy

Question: Consider the following code example. Which SPWeb objects require disposal?

```
var site = new SPSite("http://sharepoint.contoso.com");
var web1 = site.OpenWeb();
var web2 = SPContext.Current.Web;
var web3 = SPContext.Current.Site.RootWeb;
var web4 = SPContext.Current.Site.OpenWeb();
var web5 = SPContext.Current.Site.AllWebs["finance"];
```

Test Your Knowledge

Question	
<p>You want to execute a block of code if—and only if—the current user has permission to edit list items and delete list items. Which code sample should you use?</p>	
<p>Select the correct answer.</p>	
	<pre>var web = SPContext.Current.Web; if(web.DoesUserHavePermissions(SPBasePermissions.EditListItems)) { if(web.DoesUserHavePermissions(SPBasePermissions.DeleteListItems)) { // Add code here. } }</pre>
	<pre>var web = SPContext.Current.Web; if(web.DoesUserHavePermissions(SPBasePermissions.EditListItems SPBasePermissions.DeleteListItems)) { // Add code here. }</pre>
	<pre>var web = SPContext.Current.Web; if(web.DoesUserHavePermissions(SPBasePermissions.EditListItems & SPBasePermissions.DeleteListItems)) { // Add code here. }</pre>
	<pre>var web = SPContext.Current.Web; if(web.DoesUserHavePermissions(SPBasePermissions.EditListItems SPBasePermissions.DeleteListItems)) { // Add code here. }</pre>
	<pre>var web = SPContext.Current.Web; if(web.DoesUserHavePermissions(SPBasePermissions.EditListItems && SPBasePermissions.DeleteListItems)) { // Add code here. }</pre>

Module 3

Working with Lists and Libraries

Contents:

Module Overview	3-1
Lesson 1: Using List and Library Objects	3-2
Lesson 2: Querying and Retrieving List Data	3-16
Lab A: Querying and Retrieving List Data	3-27
Lesson 3: Working with Large Lists	3-31
Lab B: Working With Large Lists	3-35
Module Review and Takeaways	3-38

Module Overview

Lists and libraries are central to almost everything that SharePoint does. Every time you upload a document, or create a task, or share a contact, or reply to a discussion, you are interacting with a SharePoint list. All the resources on a SharePoint site, from page layouts to master pages to CSS files, are stored and managed in SharePoint lists. In this module, you will learn about how to work with lists and libraries programmatically using the server-side SharePoint object model. You will learn how to use query classes and LINQ to SharePoint to query and retrieve data from SharePoint lists. You will also learn how to write code that works efficiently with lists that contain large numbers of items.

Objectives


After completing this module, you will be able to:

- Interact with lists and libraries programmatically.
- Query and retrieve list data.
- Perform operations efficiently on large lists.

Lesson 1

Using List and Library Objects

SharePoint lists and libraries are the primary mechanisms for storing data in SharePoint. Almost every type of data on a SharePoint site, from contacts, tasks, and discussions to documents, images, and style resources, are stored in some kind of SharePoint list. As a SharePoint developer, you will work with lists on a regular basis. This lesson describes how to use the server-side object model to work with SharePoint lists and libraries.

 **Note:** Although this lesson focuses on the server-side object model, all of the classes described in the lesson have client-side equivalents. Later in this course, you will see how to apply the concepts you have learned here to client-side development tasks.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the relationships between the object model classes that represent SharePoint lists and libraries.
- Retrieve list and library objects programmatically.
- Create and delete list instances in code.
- Create, retrieve, update, and delete list items.
- Get and set simple and complex field values.
- Retrieve files from and upload files to a SharePoint site.

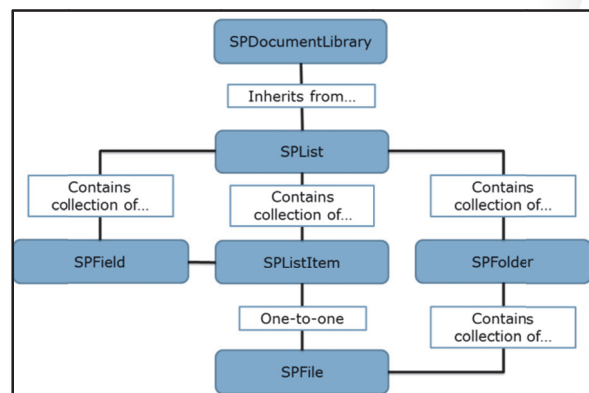
Understanding the List and Library Class Hierarchy

The SharePoint object model provides classes that you can use to work with lists, libraries, and files in your code.

The **SPList** class represents a SharePoint list or library. Every list or library in a SharePoint deployment is represented by an **SPList** instance. Document libraries are essentially a specialized type of list, and consequently the **SPDocumentLibrary** class inherits from the **SPList** class. The **SPList** class includes collections of fields, list items, and folders.

The **SPListItem** class represents an individual item in a SharePoint list. A list item can include a value for each field defined in the parent list. If the list that contains the item is a document library, the **SPListItem** instance will have an associated file, which is represented by the **SPFile** class.

The **SPFolder** class represents a folder. This can be a folder within a list or library, or it can be a virtual directory on the SharePoint site. Both the **SPList** class and the **SPWeb** class expose collections of folders. The **SPFolder** class includes a collection of **SPFile** objects.



Retrieving List and Library Objects

The **SPList** class does not include a constructor. When you need to work with a list or library in code, you retrieve the **SPList** instance from the parent **SPWeb** object. You can do this in various ways:

- Use the *SPWeb.Lists* property. The **Lists** property exposes a collection object of type **SPListCollection**. This is an enumerable collection of all the lists and libraries in the site.
- Use the *SPWeb.GetList* method. The **GetList** method enables you to retrieve a specific list or library by providing the site-relative URL of the list or library.
- Use the *SPWeb.GetListsOfType* method. The **GetListsOfType** method returns an **SPListCollection** object that contains all the lists or libraries of the specified base type. The method requires an argument of type **SPBaseType**. This is an enumeration that defines the fundamental types of SharePoint lists and libraries, such as **DocumentLibrary**, **GenericList**, and **Survey**.
- Use the *SPWeb.GetListFromWebPartPageUrl* method. The **GetListFromWebPartPageUrl** method enables you to retrieve the list or library that is associated with the first Web Part on a specific Web Part page.

The **SPListCollection** class also provides methods for retrieving lists:

- The **SPListCollection.GetList** method enables you to retrieve a list from the collection by name. The method throws an exception if the list does not exist within the collection.
- The **TryGetList** method also enables you to attempt to retrieve a list from a collection by name. However, if the list does not exist, the method returns null rather than throwing an exception.

Regardless of whether you are retrieving a list or a library, you always start by instantiating an **SPList** instance. If you are retrieving a library, and you require the additional functionality offered by the **SPDocumentLibrary** class, you can then cast the **SPList** instance to **SPDocumentLibrary**.

- Retrieve lists and libraries from the parent **SPWeb** instance

```
var web = SPContext.Current.Web;
// Use an indexer.
SPList documents = web.Lists["Documents"];
// Use a method.
SPList images = web.GetList("Lists/Images");
```

- Cast to **SPDocumentLibrary** if required

```
if (documents is SPDocumentLibrary)
{
    var docLibrary = (SPDocumentLibrary)documents;
}
```

The following code example shows how to enumerate and interact with the lists and libraries in a SharePoint site:

Retrieving lists and libraries

```
var web = SPContext.Current.Web;
foreach(SPList list in web.Lists)
{
    // Prevent users from creating folders in the list.
    list.EnableFolderCreation = false;

    // Set the email address for the list, and enable email notifications.
    list.EmailAddress = string.Format("{0}@sharepoint.contoso.com", list.RootFolder.Name);
    list.EnableAssignToEmail = true;

    // Persist list changes to the database.
    list.Update();

    // Check whether the list is a document library.
    if (list is SPDocumentLibrary)
    {
        // Cast the list to use SPDocumentLibrary properties and methods.
        var library = (SPDocumentLibrary)list;

        // If the library is not a gallery...
        if (!library.IsCatalog)
        {
            // Get a list of checked out files.
            IList<SPCheckedOutFile> checkedOutFiles = library.CheckedOutFiles;

            // Get the URL of the document template for the library.
            string docTemplate = library.DocumentTemplateUrl;
        }
    }
}
```

In most cases, the **SPList** class provides all the functionality you need to work with list items and documents. As illustrated by the code example, the **SPDocumentLibrary** class includes some additional properties and methods that you may find useful in certain circumstances. For example, the **CheckedOutFiles** property returns a list of all the files that are currently checked out in the library, and the **DocumentTemplateUrl** property enables you to get or set the template that is applied when a user creates a new document in the library.



Note: You will see more examples of how to work with the **SPList** class throughout this module.

Creating and Deleting List and Library Objects

You can use the SharePoint object model to create and delete lists and libraries programmatically.



Note: Document libraries are a subset of SharePoint lists in which the list template is of type **SPListTemplateType.DocumentLibrary**. Where this topic refers to lists, the concepts apply to all types of SharePoint lists, including document libraries.

List definitions, list templates, and list instances

Before you create your own lists in code, you need to understand some of the terminology used when describing SharePoint list functionality. When you create a new list on a SharePoint site, you are creating a *list instance*. The behavior and capabilities of the list instance are defined by a *list definition* or a *list template*.

The terms list definition and list template are often used interchangeably. The core concepts of list definitions and list templates are as follows:

- A list definition defines an XML-based schema for a list. This schema is deployed to the server file system, either within a site definition or within a Feature. The schema defines every aspect of the list, including the fields it should contain, the views and forms it provides, and the content types that it references. A list definition must specify one of the fundamental list types defined by the **SPBaseType** enumeration.
- A list template file points to a list definition. The list template file specifies how the list appears in the SharePoint user interface. For example, the list template file specifies the display name and description for the list and whether it should be added to the Quick Launch navigation menu by default. The list template file is what makes the list definition available for use, both in the UI and through the object model. List template files are deployed in Features.
- When a user has customized a list instance, either in SharePoint Designer or through the browser, he or she can save the customized list as a custom list template. The user can then create new lists from the saved list template.

List templates can come from three places: standard templates as defined by the **SPListTemplateType** enumeration, custom templates based on a list definition, or a customized and saved list template. In the SharePoint object model, list templates are represented by the **SPListTemplate** class, regardless of whether the list template points to a list definition or a saved template. You can retrieve list template objects in two ways:

- To retrieve all the list templates from a specific web, use the **SPWeb.ListTemplates** property. This returns an enumerable collection of all available list templates, regardless of whether they point to a list definition or a saved template.
- To retrieve all the available custom list templates, use the **SPSite.GetCustomListTemplates** method. This returns an enumerable collection of all the list templates that were saved by users in the site collection.

Terminology:

- List definition
- List template
- List instance

Creating list instances

```
var web = SPContext.Current.Web;
Guid listID = web.Lists.Add("Title", "Description",
    SPListTemplateType.Contacts);
```

Deleting list instances

```
var web = SPContext.Current.Web;
SPList list = web.Lists["Title"];
list.Delete();
```



Note: Creating custom list definitions and list templates is covered later in this course. At this point, you just need to know how to work with existing definitions and templates.

Creating list instances

To create a list instance programmatically, you call the **Add** method on the **SPListCollection** object exposed by the **SPWeb.Lists** property. The **Add** method includes seven overloads that enable you to create lists or libraries in various different ways. All overloads require you to specify a title and a description for the new list. However, you must also indicate the template or definition on which you want to base your new list. You can do this in several ways. For example:

- You can provide an **SPListTemplate** instance that represents a list definition or a custom list template.
- You can specify a member of the **SPListTemplateType** enumeration. This enumeration defines all the built-in list templates (which point to list definitions) provided by SharePoint Foundation.
- You can specify the feature that contains the list definition, together with an integer template type to indicate the associated list template.

Specifying a member of the **SPListTemplateType** enumeration is the most straightforward approach when you want to create a list from a SharePoint Foundation list template. If you want to create a list from a custom list template, or a list template defined by one of the SharePoint Server workloads, you typically retrieve an **SPListTemplate** instance and pass it to the **SPListCollection.Add** method.

The **SPListCollection.Add** method returns a GUID value that uniquely identifies the list instance that you have created. If you need to perform additional actions on the new list, you can use this GUID value to retrieve the **SPList** instance that represents the list.

The following code example shows how to add lists to a SharePoint site:

Creating List Instances

```
var web = SPContext.Current.Web;

// Create a list using a SharePoint Foundation list template.
var contactsListID = web.Lists.Add("Project Contacts", "Use this list to store contact details
for project stakeholders", SPListTemplateType.Contacts);

// Create a list from a custom template.
SPListTemplate template = web.ListTemplates["ContosoInventory"];
var inventoryListID = web.Lists.Add("Project Inventory", "Use this list to monitor inventory
levels for research activities", template);

// Use the GUID to construct the list if you need to perform additional actions.
SPList list = web.Lists[contactsListID];
list.OnQuickLaunch = true;
list.Update();
```

Deleting list instances

You can delete a list instance in two ways:

- You can call the **SPList.Delete** method on the list instance you want to delete.
- You can call the **SPListCollection.Delete** method, and specify the GUID of the list instance you want to delete.

The following code example illustrates how to delete a list:

Deleting List Instances

```
var web = SPContext.Current.Web;
SPList list = web.Lists["Project Inventory"];
list.Delete();
```

In most scenarios, calling the **SPList.Delete** method is the easiest way to delete a list. The **SPListCollection.Delete** method may be preferable if you can provide a GUID identifier for the list you want to delete without first instantiating an **SPList** object.

You can also recycle a list by calling the **SPList.Recycle** method, which returns the GUID for the list. When using this method, the list is placed in the recycle bin of the user account that runs the code.

Working with List Items

Items in a SharePoint list are represented by the **SPListItem** class. This applies regardless of the type of list: for example, an **SPListItem** instance can represent an event in a calendar, a document in a document library, a contact in a contacts list, and so on. The list items in a SharePoint list are stored in an enumerable collection of type **SPListItemCollection**, which you can access through the **SPList.Items** property.

Adding items to a list

The process for adding a new item to a SharePoint list consists of three high-level steps:

1. Call the **SPListItemCollection.Add** method. This creates a new list item and returns an **SPListItem** instance.
2. Specify field values, and other properties as required, on the **SPListItem** instance.
3. Call the **Update** method on the **SPListItem** instance to write the changes to the content database.

The following code example shows how to add a new item to a SharePoint list:

Adding Items to a List

```
var web = SPContext.Current.Web;
var list = web.Lists["Project Contacts"];

// Step 1: Call the SPListItemCollection.Add method.
SPListItem contact = list.Items.Add();

// Step 2: Specify field values.
contact["Last Name"] = "Kretowicz";
contact["First Name"] = "Marcin";
contact["Company"] = "Contoso Pharmaceuticals";
contact["Job Title"] = "Sales Director";
contact["Business Phone"] = "425-555-8910";
contact["Email Address"] = "marcin@contoso.com";

// Step 3: Call the Update method.
contact.Update();
```

- Adding items to a list
 - `SPList.Items.Add()`
- Retrieving and modifying list items
 - `SPList.GetItemById(int ID)`
 - `SPList.GetItemByUniqueId(Guid uniqueID)`
- Deleting list items
 - `SPListItem.Delete()`
 - `SPList.Items.Delete(int index)`
 - `SPList.Items.DeleteItemById(int ID)`

Retrieving and modifying list items

The **SPListItem** class includes two identifier properties:

- The **SPListItem.ID** property returns an integer identifier for the list item. SharePoint lists assign incrementally increasing integer identifiers to new list items.
- The **SPListItem.UniqueId** property returns a GUID value that uniquely identifies the list item.

The **SPList** class provides methods that you can use to retrieve a list item using either the integer ID or the unique ID of the item:

- The **SPList.GetItemById** method returns the list item with the specified integer identifier.
- The **SPList.GetItemByUniqueId** method returns the list item with the specified GUID identifier.

You can also retrieve list items by using an indexer with an **SPListItemCollection** instance (for example, **myList.Items[identifier]**). The indexer allows you to specify a GUID value or an integer value. However, you should use this approach with extreme caution. The integer value you provide to the indexer represents the position of the list item in the collection. This is not necessarily the same as the integer identifier of the list item. As such, best practice is to use one of the **SPList** methods, rather than the **SPListItemCollection** indexer, to retrieve specific list items.

SharePoint list items often include many fields, and these fields can contain large amounts of data. As a result, retrieving a list item is computationally expensive. The **SPList.GetItemByIdSelectedFields** method provides a more efficient way to retrieve a list item, by specifying which field values you want to retrieve. This method returns an **SPListItem** instance in the same way as the **SPList.GetItemById** method, but the returned **SPListItem** instance only contains field values for the fields you specify when you call the method.

The following code example shows how to use the **SPList.GetItemByIdSelectedFields** method:

Retrieving and Modifying List Items

```
var web = SPContext.Current.Web;
var list = web.Lists["Project Contacts"];

// This identifier value is typically retrieved by using a query class or LINQ to SharePoint.
int itemID = 1;

// Retrieve a list item containing only field values for Job Title and Email Address.
SPListItem contact = list.GetItemByIdSelectedFields(itemID, "Job Title", "Email Address");

// Update the selected field values.
contact["Job Title"] = "Vice President, Sales";
contact["Email Address"] = "vpsales@contoso.com";
contact.Update();
```

After you retrieve a list item, you update field values and call the **Update** method in the same way as if you had created a new list item.



Note: You typically retrieve list items by using either a SharePoint query class or LINQ to SharePoint. These approaches are described in the next lesson.

Deleting list items

You can delete a list item in three ways:

- You can call the **SPListItem.Delete** method on the list item you want to delete.
- You can call the **SPListItemCollection.Delete** method, and specify the index of the list item you want to delete.
- You can call the **SPListItemCollection.DeleteItemById** method, and specify the integer identifier of the list item you want to delete.

Demonstration: Creating List Items

The instructor will now demonstrate how to add list items programmatically in a visual Web Part. The Web Part enables users to submit complaints to a list by specifying a title, description, and priority for their complaint.

Demonstration Steps

- Start the 20488B-LON-SP-03 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In a File Explorer window, browse to **E:\Democode\WorkingWithListItems**, and then double-click **WorkingWithListItems.sln**.
- If the **How do you want to open this type of file (.sln)** dialog box appears, click **Visual Studio 2012**.
- In Visual Studio, in Solution Explorer, expand the **WorkingWithListItems** project node, expand the **WorkingWithListItems** project item node, and then double-click **WorkingWithListItems.ascx**.
- At the bottom of the center pane, click **Design**.
- Notice that the user control consists of a simple UI with a text box named **txtTitle**, a text box named **txtDescription**, a radio button list named **rb1Priority**, and a button named **btnSubmit**.
- Right-click the design surface, and then click **View Code**.
- In the **OnPreRender** method, locate the comment that reads **Populate the rb1Priority radio button list**.
- Immediately below the comment, add the following code:

```
var web = SPContext.Current.Web;
var list = web.Lists["Complaints"];
var field = list.Fields["Priority"] as SPFieldChoice;
rb1Priority.DataSource = field.Choices;
rb1Priority.DataBind();
```



Note: This code populates the radio button list with the available choices for the **Priority** field.

- In the **btnSubmit_Click** method, locate the comment that reads **Add a new list item to the complaints list**.

- Immediately below the comment, add the following code:

```
var web = SPContext.Current.Web;
var list = web.Lists["Complaints"];
var item = list.Items.Add();
item["Title"] = txtTitle.Text;
item["Details"] = txtDescription.Text;
item["Priority"] = rblPriority.SelectedValue;
item.Update();
```



Note: This code creates a new list item in the Complaints list, and sets field values using the values provided by the user.

- On the **BUILD** menu, click **Rebuild Solution**.
- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.



Note: If the error page appears, just refresh the page.

- In Internet Explorer, after the page finishes loading, on the ribbon, click **EDIT**.
- On the **INSERT** tab, click **Web Part**.
- In the **Categories** list, click **Contoso Web Parts**.
- In the **Parts** list, click **Working With List Items**, and then click **Add**.
- On the **PAGE** tab, click **Save**.
- On the **Working With List Items** Web Part, in the **Title** box, type a title for your complaint.
- In the **Description** box, type some details for your complaint.
- On the **Priority** list, select a priority, and then click **Submit**.
- On the Quick Launch navigation menu, click **Complaints**.
- Verify that your complaint was added to the list.
- Close Internet Explorer and close Visual Studio.

Working with Fields

Up to this point, you have seen how to set simple string-based field values when you work with list items. However, in practice, SharePoint lists include fields of many different types, such as Text, DateTime, Lookup, Currency, and Geolocation. These field types are defined by the **SPFieldType** enumeration, and you can check the type of a field by retrieving the **SPField.Type** property. There are approximately thirty built-in field types, and you can also define custom field types if your project demands it.

When you set a field value on a list item, you must provide an object of the appropriate type. For simple field types, this is straightforward. For example:

- To set the value of a Text field, provide a **String** value.
- To set the value of an Integer field, provide an **Int32** value.
- To set the value of a Boolean field, provide a **Boolean** value.
- To set the value of a Choice field, provide a **String** value (this must match one of the available choices).
- To set the value of a User field, provide an **SPUser** instance.

For more complex field types, SharePoint provides field value classes. To set the value of a complex field type, you should provide an instance of the appropriate field value class.

Field classes and field value classes

The SharePoint object model includes several classes to represent different field types. These classes all inherit from the **SPField** class, and the class names all begin with **SPField**. For example:

- The **SPFieldBoolean** class represents a Boolean field.
- The **SPFieldCalculated** class represents a calculated field.
- The **SPFieldChoice** class represents a choice field.

For more complex field types, where the value cannot be represented by a simple .NET type, the object model includes a field value class that corresponds to the field class. For example:

- The **SPFieldUrl** class represents a field that contains URL values, and the **SPFieldUrlValue** class represents the associated field value.
- The **SPFieldLookup** class represents a lookup field, and the **SPFieldLookupValue** class represents the associated field value.
- The **SPFieldGeolocation** class represents a geographical location field, and the **SPFieldGeolocationValue** class represents the associated field value.

In many cases, field classes can parse values provided in various formats into an appropriate field value class instance. For example, you can set the value of an **SPFieldGeolocation** field by providing a string value in Well-Known Text (WKT) format. However, your code will typically be more robust if you construct an appropriate field value instance when you need to set the value of a list item field.

- Field classes and field value classes
 - Simple field types accept simple values
 - Complex field types have specific field value classes

- Setting complex field values

```
Double latitude = 51.4198;
Double longitude = -2.6147;
var geoValue = new SPFieldGeolocationValue(latitude,
longitude);
item["location"] = geoVal;
```

- Retrieving complex field values

```
var geoValue = item["location"] as
SPFieldGeolocationValue;
Double latitude = geoValue.Latitude;
Double longitude = geoValue.Longitude;
```

Setting complex field values

The following code example shows how to set values for complex field types:

Setting Complex Field Values

```
// Retrieve an arbitrary list item.
var list = SPContext.Current.Web.Lists["Offices"];
SPListItem item = list.GetItemById(1);

// Set the value of a geolocation field.
Double latitude = 51.4198;
Double longitude = -2.6147;
var geoValue = new SPFieldGeolocationValue(latitude, longitude);
item["location"] = geoVal;

// Set the value of a URL field.
var urlValue = new SPFieldUrlValue();
urlValue.Url = "http://bristol.contoso.com";
urlValue.Description = "Bristol Office";
item["website"] = urlValue;

// Set the value of a multiple choice field.
var multiValue = new SPFieldMultiChoiceValue();
multiValue.Add("Parking");
multiValue.Add("Meeting Rooms");
multiValue.Add("Cafe");
item["facilities"] = multiValue;

// Call the Update method when you have finished making changes.
item.Update();
```

Retrieving complex field values

When you use an indexer to retrieve the value of a list item field, the indexer returns a **System.Object**. You must cast the returned object to an appropriate type before you can get the field values in a meaningful format.

The following code example shows how to retrieve field values for complex field types:

Retrieving Complex Field Values

```
// Retrieve an arbitrary list item.
var list = SPContext.Current.Web.Lists["Offices"];
SPListItem item = list.GetItemById(1);

// Retrieve the value of a geolocation field.
var geoValue = item["location"] as SPFieldGeolocationValue;
Double latitude = geoValue.Latitude;
Double longitude = geoValue.Longitude;

// Retrieve the value of a URL field.
var urlValue = item["website"] as SPFieldUrlValue;
string url = urlValue.Url;
string description = urlValue.Description;

// Retrieve the values in a multiple choice field.
var selectedValues = new List<String>();
var multiValue = item["facilities"] as SPFieldMultiChoiceValue;
for (int i = 0; i < multiValue.Count; i++)
{
    selectedValues.Add(multiValue[i]);
}
```

When you use an indexer with an **SPListItemCollection**, the indexer returns an object with an underlying type of **String**. In some cases, you cannot cast this string to the appropriate field value class. In these cases, you must use the **SPField.GetFieldValue** method to convert the string value into the appropriate field value type. For example, you cannot convert the string representation of a user field value into an **SPFieldUserValue** instance.

The following example shows how to use the **SPField.GetFieldValue** method:

Using the SPField.GetFieldValue Method

```
// Retrieve an arbitrary list item.
var list = SPContext.Current.Web.Lists["Appraisals"];
SPListItem item = list.GetItemById(1);

// Retrieve the value of a user field and get the display name of the user.
var field = item.Fields["Employee"] as SPFieldUser;
var fieldValue = field.GetFieldValue(item["Employee"].ToString()) as SPFieldUserValue;
string displayName = fieldValue.User.Name;
```

Working with Files

In the SharePoint object model, files on a SharePoint site are represented by the **SPFile** class. If the file is a document in a document library, there is a one-to-one relationship between the **SPFile** object and the **SPListItem** object:

- You can retrieve the **SPFile** instance from the **SPListItem.File** property.
- You can retrieve the **SPListItem** instance from the **SPFile.Item** property.

The **SPListItem** class and the **SPFile** class represent different aspects of a document in a document library. You use the **SPListItem** class if you want to work with the metadata of the document, for example to update field values. You use the **SPFile** class if you want to work with the document itself, for example to check the document in or out, view version history, or to access the contents of a file as a stream or a byte array.

Retrieving files

Although you can retrieve **SPFile** instances from individual list items, this is not the best approach unless you specifically want to retrieve the **SPListItem** instance as well as the **SPFile** instance. The primary mechanism for accessing files is through the **SPWeb** class or the **SPFolder** class. Both of these classes include a **Files** property, which exposes an enumerable collection of **SPFile** objects. The **SPWeb** class also includes various methods that you can use to retrieve specific files or folders.

- Retrieving files
 - Get files from SPWeb object or SPFolder object
 - Use an indexer (SPWeb.Files or SPFolder.Files)
 - Use a method (SPWeb.GetFile, SPWeb.GetFileAsString)

• Checking files in and out

```
if (file.CheckOutType == SPFile.SPCheckOutType.None)
{
    file.CheckOut();
    // Update the file as required...
    file.CheckIn("File updated.");
}
```

- Adding and updating files
 - SPFileCollection.Add method

The following code example shows how to retrieve files programmatically:

Retrieving Files

```
var web = SPContext.Current.Web;

// Get the contents of a file as a string.
string contents = web.GetFileAsString("Documents/Text.txt");

// Get the contents of a file as a byte array.
SPFile image = web.GetFile("Documents/Image.bmp");
byte[] imageBytes = image.OpenBinary();

// Get the contents of a file as a stream and save to the local file system.
SPFile video = web.GetFile("Documents/Video.mp4");
int bufferSize = 10 * 1024;
using (Stream stream = video.OpenBinaryStream())
{
    using (FileStream fs = new FileStream(@"C:\LocalPath\Video.mp4", FileMode.Create,
    FileAccess.Write))
    {
        byte[] buffer = new byte[bufferSize];
        int byteCount = 0;
        while ((byteCount = stream.Read(buffer, 0, buffer.Length)) > 0)
        {
            fs.Write(buffer, 0, byteCount);
        }
    }
}
```

Checking files in and out

Check-outs and check-ins are a mechanism that enables users to prevent other users from modifying a document while they make changes. If a user checks out a document in a document library, the document is locked for editing by that user. No other users can make changes to the document while the check-out is in place. When the user checks in the document, the lock is removed and changes made by the user are visible to everyone.

You can use the **SPFile** class to check documents in and out programmatically. For example, you might want to develop a "bulk check-in" solution, so that users who have uploaded multiple documents to a library do not have to check in each file individually.

To check out a file, you call the **SPFile.CheckOut** method. If required, you can also specify a check-out type by providing a member of the **SPFile.SPCheckOutType** enumeration as an argument to the **CheckOut** method. The **SPFile.SPCheckOutType** enumeration defines the following values:

- *None*. This indicates that the document is not checked out.
- *Offline*. This indicates that the document is checked out for editing on a local computer. This is equivalent to selecting the **Use my local drafts folder** option when you check out a file through the browser UI.
- *Online*. This indicates that the document is checked out for editing on the server. This is the default value when you check out a document.

To check in a file, you call the **SPFile.CheckIn** method and provide a comment. If required, you can also use the **SPCheckinType** enumeration to indicate whether you are checking in a major version or a minor version, or whether you want to overwrite the current version.

The following code example shows how to check files in and out:

Checking Files In and Out

```
var web = SPContext.Current.Web;

// Retrieve the file.
var file = web.GetFile("Documents/Invoice.docx");

// Check the file out.
if (file.CheckOutType == SPFile.SPCheckOutType.None)
{
    file.CheckOut();

    // Update the file as required...
    // Check the file back in.
    file.CheckIn("Invoice updated.");
}
```

Adding and updating files

To add a file to a SharePoint site, you call the **Add** method on an **SPFileCollection** object. Both the **SPWeb** class and the **SPFolder** class include a **Files** property that exposes an **SPFileCollection** object.

The **SPFileCollection.Add** method includes 19 overloads that enable you to add files in a multitude of different ways. In each case, you must provide a site-relative or server-relative URL to indicate the location at which you want to add the file. You typically provide the contents of the file as a byte array or a stream. Other overload variations enable you to specify file properties in various different ways.

The following code example shows how to add a file to a SharePoint site:

Adding and Updating Files

```
var web = SPContext.Current.Web;

// Get a byte array from another file, either from the local file system or from elsewhere on
// SharePoint.
SPFile source = web.GetFile("Documents/Image.bmp");
byte[] imageBytes = source.OpenBinary();

// Add the file to a library.
SPFile file = web.Files.Add(@"Images/NewImage.bmp", fileBytes);

// Check in the file after adding it.
file.CheckIn("New image added.");
```

You can also use the **Add** method to update existing files. Several **Add** method overloads include a Boolean parameter that you can use to indicate whether you want to overwrite existing files with the same name. This enables you to add a file as a new version of an existing file.

Lesson 2

Querying and Retrieving List Data

SharePoint enables you to query and retrieve data from lists and libraries in a variety of ways. In addition to built-in approaches, such as customizable list views and the Content Query Web Part, there are several ways to query and retrieve list data programmatically. This lesson describes the main approaches to retrieving list data and discusses when each approach is appropriate.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the main approaches to querying and retrieving list data.
- Build queries using Collaborative Application Markup Language (CAML).
- Use the **SPQuery** class to query data from a single list.
- Use the **SPSiteDataQuery** class to retrieve data from multiple lists in a site collection.
- Use LINQ to SharePoint to query SharePoint lists.
- Use the **SPMetal** utility to build entity classes for LINQ to SharePoint.

Approaches to Querying List Data

In many scenarios, you will want to retrieve one or more list items that match certain criteria. For example, you might want to retrieve all list items that were modified after a particular date, or all list items that were modified by a particular user, or all list items that contain specific field values. You can approach these tasks in various ways.

Why you should not enumerate list items

If you are new to SharePoint development, you might be tempted to retrieve list items by enumerating a list item collection and testing each list item against your selection criteria, as shown by the following example:

The Wrong Approach to Retrieving List Items

```
var web = SPContext.Current.Web;
var list = web.Lists["Inventory"];
foreach (SPListItem item in list.Items)
{
    // Do something with the list item.
}
```

This approach is considered bad practice. When you enumerate a list item collection, the enumerator must instantiate each list item in the collection. Instantiating a list item is a computationally expensive process, particularly if the list item contains large amounts of field data.

Rather than enumerating list item collections, SharePoint provides two optimized approaches that you can use to retrieve list items: query classes, and LINQ to SharePoint.

- Avoid enumerating list item collections
 - Computationally expensive
 - SharePoint provides alternative, optimized approaches
- SharePoint query classes
 - SPQuery
 - SPSiteDataQuery
- LINQ to SharePoint
 - LINQ to SharePoint Provider
 - SPMetal tool

Using query classes

SharePoint provides two query classes for efficient retrieval of list data:

- *SPQuery*. Use the **SPQuery** class to retrieve list items that match specific criteria from a single list.
- *SPSiteDataQuery*. Use the **SPSiteDataQuery** class to retrieve and aggregate list item data from multiple lists in a site collection.

Both of these classes require you to create a query by using Collaborative Application Markup Language (CAML). CAML is a declarative, XML-based syntax that is used widely throughout SharePoint to describe a range of declarative components.

Using LINQ to SharePoint

You can use Language Integrated Query (LINQ) syntax to retrieve data from SharePoint lists. For server-side code, this functionality is provided by the LINQ to SharePoint Provider, which converts LINQ queries into CAML queries.

Before you can use LINQ to SharePoint in your code, you must generate entity classes to represent the lists you want to work with. SharePoint includes a command-line tool, SPMetal.exe, which you can use to generate entity classes for the lists in a specified site collection.

Discussion: Retrieving List Data in Code

The instructor will now lead a brief discussion around the following questions:

- When should you retrieve custom list data in code?
- What built-in alternatives should you consider before you retrieve custom list data in code?

- When should you retrieve custom list data in code?
- What built-in alternatives should you consider before you retrieve custom list data in code?

Building CAML Queries

It is almost impossible for a SharePoint developer to avoid working with CAML. Although it is possible to retrieve list item data without using CAML, you are likely to require a rudimentary knowledge of CAML queries at some point in your SharePoint development career. This topic provides an introduction to creating queries in CAML.



Note: As CAML queries grow longer and more complex, they can become increasingly difficult to create and debug. There are various third-party tools available that provide an interactive interface for building and testing CAML queries.

- The Where clause
- Using comparison operators
- Combining comparison operators

```
<Query>
  <Where>
    <And>
      <Leq>
        <FieldRef Name="Inventory"></FieldRef>
        <Value Type="Integer">300</Value>
      </Leq>
      <Eq>
        <FieldRef Name="OrderPlaced"></FieldRef>
        <Value Type="Boolean">>false</Value>
      </Eq>
    </Where>
  </Query>
```

The Where clause

The top-level element of a CAML query is always a **Where** element. The **Where** clause is functionally equivalent to the **SELECT** statement in SQL. Within the **Where** element, the query can contain logical joins, comparison operators, grouping and ordering operators, and various field references and constants.

Comparison operators

If you want to specify criteria that determine which list items you want your query to return, you use a comparison operator. The CAML query schema includes the following comparison operators:

Operator	Description
BeginsWith	Matches field values that begin with a specified string. Use with Text or Note field types.
Contains	Matches field values that contain a specified string. Use with Text or Note field types.
DateRangesOverlap	Matches recurring events if the dates overlap with a specified DateTime value.
Eq	Matches field values that are equal to a specified value.
Geq	Matches field values that are greater than or equal to a specified value.
Gt	Matches field values that are greater than a specified value.
In	Matches field values that are equal to one of a specified collection of values.
Includes	Matches multiple value lookup field values if the selected values in the list item include a specified value.
IsNotNull	Matches values that are not empty.
IsNull	Matches values that are empty.
Leq	Matches field values that are less than or equal to a specified value.
Lt	Matches field values that are less than a specified value.
Membership	Matches user or group-based field values based on different types of membership.
Neq	Matches field values that are not equal to a specified value.
NotIncludes	Matches multiple value lookup field values if the selected values in the list item do not include a specified value.

The following example shows a CAML query that returns all the items where the **Inventory** field value is less than or equal to **300**:

Using Comparison Operators

```
<Query>
  <Where>
    <Leq>
      <FieldRef Name="Inventory"></FieldRef>
      <Value Type="Integer">300</Value>
    </Leq>
  </Where>
</Query>
```

Combining comparison operators

To create more complex queries, you can combine multiple comparison operators:

- To return items that match all comparison clauses, enclose the comparison clauses in an **And** element.
- To return items that match any comparison clause, enclose the comparison clauses in an **Or** element.

The following example shows a CAML query that returns all the items where the **Inventory** field value is less than or equal to **300** and the **OrderPlaced** field value is equal to **false**:


Combining Comparison Operators

```
<Query>
  <Where>
    <And>
      <Leq>
        <FieldRef Name="Inventory"></FieldRef>
        <Value Type="Integer">300</Value>
      </Leq>
      <Eq>
        <FieldRef Name="OrderPlaced"></FieldRef>
        <Value Type="Boolean">>false</Value>
      </Eq>
    </And>
  </Where>
</Query>
```

Building more complex CAML queries

You can make your CAML queries more sophisticated in a variety of ways. For example, you can add ordering and grouping criteria, and you can work with list relationships by using joins and projected fields. For more information about these techniques, see the following resources:

- *Querying from Server-side Code* at <http://go.microsoft.com/fwlink/?LinkID=306780>.
- *Query Schema* at <http://go.microsoft.com/fwlink/?LinkID=306781>.

 **Additional Reading:** For more general information about working with CAML in SharePoint 2013, see *Introduction to Collaborative Application Markup Language (CAML)* at <http://go.microsoft.com/fwlink/?LinkID=306782>.

Using the SPQuery Class

The **SPQuery** class provides a container for a CAML query. You can use the **SPQuery** class to execute a CAML query against an individual list. To use this approach, you perform the following high-level steps:

1. Construct an **SPQuery** instance.
2. Set properties on the **SPQuery** instance to specify the CAML query text, along with various other query properties as required.
3. Call the **SPList.GetItems** method, supplying the **SPQuery** instance as an argument.

1. Construct an SPQuery instance
2. Set CAML query properties
3. Call SPList.GetItems, passing the SPQuery instance

```
SPQuery query = new SPQuery();
query.Query = @"[CAML query text]";

var web = SPContext.Current.Web;
var list = web.Lists["Company Cars"];

SPListItemCollection items = list.GetItems(query);
```

The following code example shows how to use the **SPQuery** class to retrieve items from a list:

Using the SPQuery Class

```
// Construct an SPQuery instance.
SPQuery query = new SPQuery();
query.Query = @"
  <Where>
    <Eq>
      <FieldRef Name=""Color""></FieldRef>
      <Value Type=""Choice"">Blue</Value>
    </Eq>
  </Where>
";

// Retrieve a list instance.
var web = SPContext.Current.Web;
var list = web.Lists["Company Cars"];

// Execute the query against the list.
SPListItemCollection items = list.GetItems(query);
```

In addition to the **Query** property, you can use various other properties of the **SPQuery** class to control what data is returned by your query. For example:

- **ViewFields.** You can set the **ViewFields** property to specify which field values you want to be populated in the returned list items, in the same way that you can use the **SPList.GetItemByIdSelectedFields** method to return a single list item with selected fields only. Setting the **ViewFields** property can make your queries more efficient, because you are retrieving only the field values that you specifically request.
- **Joins.** You can construct more sophisticated queries by specifying joins that merge data across two or more lists, in the form of a CAML **Joins** element. The lists must be connected through a lookup field.
- **ProjectedFields.** If the list you are querying includes a lookup field, you can use the **ProjectedFields** property to add fields from the foreign list to your query results.



Additional Reading: For more information about these properties, see *SPQuery class* at <http://go.microsoft.com/fwlink/?LinkID=306784>.

Using the SPSiteDataQuery Class

You can use the **SPSiteDataQuery** class to execute a query against multiple lists in a site collection. This is useful in scenarios where you want to aggregate list data from multiple sources. For example, you might want to provide a summary list of outstanding tasks from task lists on various different project sites.

To use the **SPSiteDataQuery** class, you perform the following high-level steps:

1. Construct an **SPSiteDataQuery** instance.
2. Set properties on the **SPSiteDataQuery** instance to specify query parameters.
3. Call the **SPWeb.GetSiteData** method, supplying the **SPSiteDataQuery** instance as an argument.

```

1. Construct an SPSiteDataQuery instance
2. Set CAML query properties
3. Call SPWeb.GetSiteData, passing the SPSiteDataQuery instance

SPSiteDataQuery query = new SPSiteDataQuery();
query.Query = @"[CAML query text]";

query.Webs = @"<Webs Scope=""SiteCollection"" />";
query.Lists = @"<Lists ServerTemplate=""107"" />";

var web = SPContext.Current.Web;
DataTable results = web.GetSiteData(query);
    
```

To specify the scope of an **SPSiteDataQuery** instance, you use the **Webs** and **Lists** properties. There are three options for the **Webs** property:

- *Not set.* By default, the query will only return data from the current web (in other words, the web on which you call the **GetSiteData** method).
- `<Webs Scope="Recursive" />`. In this case, the query will return data from the current web and any children of the current web.
- `<Webs Scope="SiteCollection" />`. In this case, the query will return data from all webs in the current site collection.

To specify which lists the query should consider, you set the **Lists** property to a **Lists** element. You can configure the **Lists** element in various ways. The following table shows some examples of **Lists** elements.

Lists element	Description
<code><Lists ServerTemplate="116" /></code>	Includes lists based on the template with ID 116. This is the ID of the Master Page Gallery template.
<code><Lists BaseType="4" /></code>	Includes lists with a base type value of 4. This base type value corresponds to survey-type lists.
<code><Lists></code> <code><List ID="[GUID]" /></code> <code><List ID="[GUID]" /></code> <code><List ID="[GUID]" /></code> <code></Lists></code>	Includes lists with the specified unique identifiers.
<code><Lists ServerTemplate="101"</code> <code>Hidden="TRUE" /></code>	Includes lists based on the template with ID 101 (document libraries), including hidden lists. Hidden lists are omitted from query results by default.

The **SPWeb.GetSiteData** method returns results as a **DataTable**. As such, you cannot use the **SPSiteDataQuery** approach to retrieve multiple list items for editing. As with the **SPQuery** class, you can use the **ViewFields** property to restrict the field values returned by the query to only those fields in which you are interested.

The following code example shows how to use the **SPSiteDataQuery** class to retrieve data from multiple lists:

Using the SPSiteDataQuery Class

```
// Construct an SPSiteDataQuery instance.
SPSiteDataQuery query = new SPSiteDataQuery();
query.Query = "
  <Where>
    <Eq>
      <FieldRef Name=\"Task Status\"></FieldRef>
      <Value Type=\"Choice\">Not Started</Value>
    </Eq>
  </Where>
";

// Set the query scope.
query.Webs = @"<Webs Scope=\"SiteCollection\" />";
query.Lists = @"<Lists ServerTemplate=\"107\" />";

// Specify the fields you want to return.
query.ViewFields = @"
  <FieldRef Name=\"Task Name\" />
  <FieldRef Name=\"Description\" />
  <FieldRef Name=\"Due Date\" />
  <FieldRef Name=\"Assigned To\" />
  <FieldRef Name=\"Task Status\" />
";

// Execute the query against an SPWeb.
DataTable results = SPContext.Current.Web.GetSiteData(query);
```

Using LINQ to SharePoint

Many developers do not like writing CAML queries. The syntax can be unintuitive, particularly as queries grow more complex, and troubleshooting CAML can be challenging (a "bad" CAML query typically fails quietly and returns no results).

LINQ to SharePoint provides an alternative server-side approach to querying SharePoint lists. When you use the LINQ to SharePoint provider, you can use LINQ expressions to retrieve data from SharePoint lists. Behind the scenes, the LINQ to SharePoint provider converts your LINQ expressions into CAML queries.

- Use LINQ syntax to query SharePoint lists
- LINQ to SharePoint provider converts LINQ statements into CAML queries
- Requires generation of entity classes

```
TeamDataContext teamWeb = new
    TeamDataContext("http://team.contoso.com");

var blueCars = from car in teamWeb.CompanyCars
    where car.Color.Contains("Blue")
    select car;
```



Note: Before you can use LINQ to SharePoint, you must generate entity classes to represent the lists and content types that you want to work with. The next topic describes how to use the SPMetal tool to generate entity classes.

Understanding entity classes

When you use LINQ to SharePoint, you work with a hierarchical set of entity classes. At the top of the hierarchy is the **Microsoft.SharePoint.Linq.DataContext** class, which represents a SharePoint web. When you use the SPMetal tool to generate entity classes for a specific SharePoint web, SPMetal generates a top-level class that derives from **DataContext**. This derived class includes a strongly-typed property for each list in the web.


The following code example shows how to instantiate a **DataContext** object and use it to query a list. In this example, the **TeamDataContext** class derives from the **DataContext** class:

Using the DataContext Object

```
// Instantiate the DataContext object by passing in the web URL.
TeamDataContext teamWeb = new TeamDataContext("http://team.contoso.com");

// Use strongly-typed properties to get lists from the web.
// In this case, we want to query the CompanyCars list.
var blueCars = from car in teamWeb.CompanyCars
               where car.Color.Contains("Blue")
               select car;
```

Individual lists are represented by the **EntityList<TEntity>** class. **TEntity** is a class that represents the content type of the list items, which is typically generated by the SPMetal tool. For example, a contacts list could be represented by an **EntityList<Contact>** instance. The content type class includes strongly-typed properties for every field in the content type.

 **Additional Reading:** For more information about entity classes, see *Entity Classes* at <http://go.microsoft.com/fwlink/?LinkID=306788>.

Using SPMetal to Generate Entity Classes

SPMetal is a command-line tool that you can use to generate entity classes for LINQ to SharePoint. SPMetal is included in SharePoint Foundation, and you can find the tool in the following folder:

```
%COMMONPROGRAMFILES%\Microsoft
Shared\Web Server Extensions\15\BIN
```

To use SPMetal, you must specify two command-line options as a minimum:

- Use the **web** option to specify the SharePoint web for which you want to generate entity classes.
- Use the **code** option to specify the name and path of the code file to be created.

For example, to generate a code file named `TeamSite.cs` for the SharePoint web at `http://team.contoso.com`, you would use the following command:

```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs
```

If you specify a code file with a file name extension of `.cs` or `.vb`, SPMetal will infer a code language of Visual C# or Visual Basic, respectively. Alternatively, you can use the **language** option to specify a value of **csharp** or **vb**.

- Use the **web** option to indicate the target site
- Use the **code** option to specify the output code file


```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs
```
- Add **user** and **password** options to run in a different user context


```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs
/user:administrator@contoso.com /password:Pa$$w0rd
```
- Use the **parameter** option to specify a parameter file for customized output


```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs
/parameters:TeamSite.xml
```

In some cases, you may want to run the command using alternative credentials to your current logged-on identity. To generate entity classes, the identity you use must as a minimum have full read permissions on the target web. You can specify alternative credentials by including **user** and **password** options:

```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs /user:administrator@contoso.com /password:Pa$$w0rd
```

You can customize the output of the SPMetal tool by creating a parameters file. For example, you can use a parameters file to:

- Change the class names that are assigned to SharePoint entities.
- Exclude items you do not need from the code file.
- Generate classes for items that would otherwise be excluded from the code file, such as hidden content types.



Additional Reading: For detailed information about how to create a parameters file, see *Overriding SPMetal Defaults by Using a Parameters XML File* at <http://go.microsoft.com/fwlink/?LinkID=311888>.

To use a parameters file to customize the output from SPMetal, you must use the **parameters** option to specify the location of the parameters XML file:

```
SPMetal /web:http://team.contoso.com /code:TeamSite.cs /parameters:TeamSite.xml
```



Additional Reading: For more information about SPMetal, including command line options and how to use a parameters file, see *SPMetal* at <http://go.microsoft.com/fwlink/?LinkID=306787>.

Demonstration: Generating Entity Classes in Visual Studio 2012

The instructor will now demonstrate how to run the SPMetal command-line tool as a pre-build command in a Visual Studio 2012 solution.



Additional Reading: For more information about using SPMetal, see *SPMetal* at <http://go.microsoft.com/fwlink/?LinkID=306787>.

Demonstration Steps

- Connect to the 20488B-LON-SP-03 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, type **Computer**, right-click **Computer**, and then click **Properties**.
- In the **System** window, click **Advanced system settings**.
- In the **System Properties** dialog box, click **Environment Variables**.
- In the **Environment Variables** dialog box, under **System variables**, select the **Path** row, and then click **Edit**.

- In the **Edit System Variable** dialog box, in the **Variable value** box, press End, type the following, and then click **OK**:

```
;C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\BIN
```



Note: Take care not to edit or delete any of the existing values in the **Path** variable.

- In the **Environment Variables** dialog box, click **OK**.
- In the **System Properties** dialog box, click **OK**.
- In a File Explorer window, browse to **E:\Democode\UsingSPMetal**, and then double-click **UsingSPMetal.sln**.
- If the **How do you want to open this type of file (.sln)** dialog box appears, click **Visual Studio 2012**.
- In Solution Explorer, right-click the project node (the **UsingSPMetal** node immediately below the Solution node), and then click **Properties**.
- Click **Build Events**.
- In the **Pre-build event command line** box, type the following:

```
cd $(ProjectDir)
SPMetal /web:http://team.contoso.com /code:TeamSite.cs
```

- On the **FILE** menu, click **Save All**.
- On the **BUILD** menu, click **Rebuild Solution**.
- In Solution Explorer, right-click **References**, and then click **Add Reference**.
- In the search box, type **Microsoft.SharePoint.Linq**.
- In the center pane, select **Microsoft.SharePoint.Linq** (make sure the check box is selected), and then click **OK**.
- In Solution Explorer, right-click the project node, point to **Add**, and then click **Existing Item**.
- In the **Add Existing Item** dialog box, click **TeamSite.cs**, and then click **Add**.
- Briefly explore the **TeamSite.cs** file. Point out that the file contains a top-level class named **TeamSiteDataContext**, together with classes for all the list content types in the site.
- In Solution Explorer, expand **UsingSPMetalWebPart**, expand **UsingSPMetalWebPart.ascx**, and then double-click **UsingSPMetalWebPart.ascx.cs**.
- At the top of the page, immediately below the existing **using** statements, add the following **using** statements:

```
using Microsoft.SharePoint.Linq;
using System.Linq;
```

- In the **Page_Load** method, type the following code to instantiate a **DataContext** object, and then press Enter:

```
TeamSiteDataContext context = new TeamSiteDataContext("http://team.contoso.com");
```

- Type the following code, and then press Enter:

```
var salesDocs = from doc in context.Documents
```

- Type the following code, and then press Enter:

```
where doc.Title.Contains("Sales")
```

- Type the following code, and then press Enter:

```
select doc;
```

- Note that the Visual Studio IDE is using LINQ to SharePoint to provide IntelliSense and strongly typed objects in your code.
- Save your changes and close all open windows.

Lab A: Querying and Retrieving List Data

Scenario

The finance department at Contoso uses a SharePoint list to track capital expenditure requests and approvals. The team wants a quicker way to approve low-value requests in bulk. Your task is to create a custom Web Part that enables finance team members to rapidly locate all requests for expenditure of less than \$500 and to approve these requests in bulk.

Objectives

After completing this lab, you will be able to:

- Use the **SPQuery** class to retrieve list items that match specified criteria.
- Update list items from custom server-side code.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-03
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

Exercise 1: Querying List Items

Scenario

In this exercise, you will configure a Web Part to retrieve data by using the **SPQuery** class. First, you will construct a query object and configure various CAML-based properties. Next, you will use this query object to retrieve items from the Expenditure Requests list. Finally, you will bind the results to an ASP.NET **ListView** control.

The main tasks for this exercise are as follows:

1. Review the Starter Code
2. Create a Query Object
3. Query the Expenditure Requests List
4. Bind List Item Fields to the ListView Control
5. Test the Web Part

► Task 1: Review the Starter Code

- Start the 20488B-LON-SP-03 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password Pa\$\$w0rd.
- In Visual Studio 2012, open the solution at **E:\Labfiles\Starter\ExpenseChecker_LabA\ExpenseChecker.sln**.
- Open the **ExpenseCheckerWebPart.ascx** file, and review the contents of the page. Notice that the user control contains an ASP.NET **Listview** control with the following row headers:
 - a. Unnamed row
 - b. Requestor
 - c. Category
 - d. Description

- e. Amount
- Notice that the page also includes buttons named **btnApprove** and **btnReject**.
- Review the code-behind file for the user control. Notice that the **btnApprove_Click** and the **btnReject_Click** both invoke a method named **UpdateItems**, which retrieves the collection of selected items from the **ListView** control.
- ▶ **Task 2: Create a Query Object**
- In the code-behind file, in the **OnPreRender** method, create and instantiate a new **SPQuery** object named **query**.
- Configure the query object to retrieve list items that match the following criteria:
 - a. The **CapExAmount** field has a value of less than \$500.
 - b. The **CapExStatus** field has a value of **Pending**.
- Configure the query object to populate the following fields in the returned list items:
 - a. The **Uniqueld** field
 - b. The **CapExRequestor** field
 - c. The **CapExCategory** field
 - d. The **CapExDescription** field
 - e. The **CapExAmount** field



Note: Storing the **Uniqueld** value makes it easier to retrieve the list item again should you need to amend it.

- ▶ **Task 3: Query the Expenditure Requests List**
- Retrieve the **Expenditure Requests** list from the current SPWeb instance.
- Use the query object that you created in the previous task to get a collection of list items from the **Expenditure Requests** list.
- Convert the collection of list items into a **DataTable** object, and then set the data source of the **lstExpenses** control to the **DataTable** object.
- Call the **DataBind** method on the **lstExpenses** control.
- ▶ **Task 4: Bind List Item Fields to the ListView Control**
- Switch to the **ExpenseCheckerWebPart.ascx** file.
- Within the **asp:ListView** element, within the **ItemTemplate** element, locate the **asp:HiddenField** element with an **ID** value of **hiddenID**.
- Use an **Eval** statement to set the value of the hidden field to the **Uniqueld** property of the current item.
- Locate the **asp:Label** element with an **ID** value of **lblRequestor**.
- Use an **Eval** statement to set the text value of the label to the **CapExRequestor** property of the current item.
- Locate the **asp:Label** element with an **ID** value of **lblCategory**.

- Use an **Eval** statement to set the text value of the label to the **CapExCategory** property of the current item.
- Locate the **asp:Label** element with an **ID** value of **lblDescription**.
- Use an **Eval** statement to set the text value of the label to the **CapExDescription** property of the
- Locate the **asp:Label** element with an **ID** value of **lblAmount**.
- Use an **Eval** statement to set the text value of the label to the **CapExAmount** property of the current item. (Hint: use the format string **{0:C2}** to make the list view format the value as currency.)

► Task 5: Test the Web Part

- On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Add the **Expense Checker** Web Part to the page, and then save the page.



Note: If you haven't modified the element manifest, you can find the **Expense Checker** Web Part in the **Contoso Web Parts** group.

- Verify that the **Expense Checker** Web Part displays several list items with the following information:
 - a. The requestor name
 - b. The expense category
 - c. The expense description
 - d. The amount
- Close Internet Explorer.

Results: After completing this exercise, you should have created a visual Web Part that retrieves and displays data from a SharePoint list.

Exercise 2: Updating List Items

Scenario

In this exercise, you will modify the Expense Checker Web Part to enable users to approve or reject expense items.

The main tasks for this exercise are as follows:

1. Update the Status of Selected List Items
2. Test the Web Part

► Task 1: Update the Status of Selected List Items

- In the code-behind file for the Expense Checker Web Part, locate the **UpdateItems** method.
- After the existing code in the method, add code to retrieve the **Expenditure Requests** list from the current **SPWeb** instance.

- Add code to enumerate the **selectedItems** collection.
- For each item in the collection, perform the following actions:
 - a. Retrieve the unique ID of the item. (Hint: use the **FindControl** method to return the **HiddenField** control that contains the unique identifier value.)
 - b. Use the unique ID to retrieve the corresponding list item from the **Expenditure Requests** list.
 - c. Update the **Request Status** field of the list item to the value of the **status** variable.
 - d. Call the **Update** method on the list item.

► **Task 2: Test the Web Part**

- On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, when the page loads, on the **Expense Checker** Web Part, select three or four items, and then click **Approve**. Verify that the items are no longer displayed in the Web Part.
- On the **Expense Checker** Web Part, select another three or four items, and then click **Reject**. Verify that the items are no longer displayed in the Web Part.
- On the Quick Launch navigation menu, click **Expenditure Requests**.
- Verify that the **Expenditure Requests** list includes the items with statuses of **Approved** and **Rejected** that you amended in the Web Part.
- Close Internet Explorer.
- Close Visual Studio.

Results: After completing this exercise, you should have configured the Expense Checker Web Part to enable users to approve or reject multiple expense items simultaneously.

Lab Discussion: LINQ to SharePoint versus SPQuery

In this lab, you used the **SPQuery** class to retrieve list data. In the last lesson, you saw how to use LINQ to SharePoint to retrieve list data. The instructor will now lead a brief discussion around the advantages and disadvantages of each approach.

- In what scenarios do you think the LINQ to SharePoint approach might be the best choice?
- When do you think the SPQuery approach might be more appropriate?

Lesson 3

Working with Large Lists

SharePoint list operations often require large amounts of server resources. Performing operations involving large numbers of list items can result in poorly-performing code. It can also affect the performance of the SharePoint farm as a whole. This lesson describes how SharePoint manages list performance and how you can manage the performance of large list queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how SharePoint manages list performance.
- Explain how to use object model overrides to bypass list view thresholds.
- Use the **ContentIterator** class to perform queries involving large numbers of items.

Understanding List Performance Management

Performing operations on SharePoint lists consumes server resources, such as memory and processor time. As the number of items involved in the operation increases, the resources required to service the operation also increase rapidly. List operations that involve large numbers of list items can affect the performance of the server farm as a whole. The performance impact increases sharply if the number of list items increases beyond 5,000.

Query throttling

To protect the farm from performance degradation, SharePoint implements list view thresholds. This restricts the number of list items that you can access in a single operation to a value configured by the farm administrator (at the web application level). By default, SharePoint lists include the following list view thresholds:

- Users cannot perform operations that access more than 5,000 list items.
- Auditors and administrators cannot perform operations that access more than 20,000 list items.

List view thresholds apply to all types of operations that require access to multiple list items. Most commonly, list view thresholds affect data retrieval. However, other operations can also be affected. For example, if you try to delete a list or folder that contains more than 5,000 items, the delete operation will be throttled.

If you are viewing a list through a built-in list view and you encounter a list view threshold, SharePoint limits the number of items returned to the threshold value and displays a warning indicating that not all results were returned. By contrast, if you are retrieving list items programmatically and you encounter a threshold, SharePoint will throw an exception of type **SPQueryThrottledException** and no results will be returned. If there is a chance that your code could encounter list view thresholds, you should check for this exception and inform the user or log the error as appropriate.

- Query throttling
 - 5,000 items for users
 - 20,000 items for auditors and administrators
- Happy hour
 - Permit larger queries during daily time windows
 - Allow larger queries at periods of low demand
- Indexing strategies
 - Careful indexing makes queries more efficient

Happy hour

Farm administrators can specify a daily time window in which large queries are permitted. This is commonly known as "happy hour." During this time window, list view thresholds are not applied. This feature is often used to allow background processing tasks, such as timer jobs, to perform large list queries during periods of low demand.

Indexing strategies

Careful use of indexing can make list queries more efficient and reduce the likelihood that queries will be blocked by list view thresholds. List indexes are typically configured through the user interface by site administrators. You can define up to 20 indices on a list. These can be simple indices, which index the value of a single column, or compound indices, which index the combined value of a primary and secondary column.

To illustrate the effect of indexing, suppose you have a list that contains 10,000 items. You want to submit a query that returns all items with a **Status** field value of **Outstanding**, of which there are roughly 100. If the **Status** column is not indexed, the server would need to load all 10,000 list items to evaluate the **Outstanding** field value of each item. By contrast, if the **Status** column is indexed, the server would load only those items with a field value of **Outstanding**.



Note: List indexing is also a performance-intensive activity, and overuse of list indices will adversely affect the performance of the SharePoint farm.

Overriding List View Thresholds

Farm administrators can choose whether to allow developers to override list view thresholds programmatically. If object model overrides are enabled in the web application settings, you can choose to bypass list view threshold in your code. In most cases, you override the list view threshold by setting the **QueryThrottleMode** property on an **SPQuery** instance.



Note: You can also change throttling settings programmatically on **SPList** and **SPWebApplication** objects. However, in both cases, your code will need to run with farm administrator permissions. These settings are typically more useful for administrators who use Windows PowerShell to automate SharePoint management and administration tasks.

- Use the **SPQuery.QueryThrottleMode** property to override list view thresholds
- Object model overrides must be enabled at the web application level

```
SPQuery query = new SPQuery0();
query.Query = @"[CAML query text]";

query.QueryThrottleMode =
  SPQueryThrottleOption.Override;
```

The following code shows how to bypass list view thresholds programmatically:


Overriding List View Thresholds

```
// Construct an SPQuery instance.
SPQuery query = new SPQuery();
query.Query = @"[Query text]";

// Set the QueryThrottleMode property.
query.QueryThrottleMode = SPQueryThrottleOption.Override;

// Retrieve a list instance.
var web = SPContext.Current.Web;
var list = web.Lists["Large List"];

// Execute the query against the list.
SPListItemCollection items = list.GetItems(query);
```

 **Additional Reading:** Various nuances and limitations apply when you use an object model override. For more information, see *SPQueryThrottleOption enumeration* at <http://go.microsoft.com/fwlink/?LinkID=306789>.

Be aware that list view thresholds exist both to protect the performance of your code from slow-running queries and to protect the performance of the farm as a whole. You should use object model overrides judiciously and only where other options are unacceptable.

Using the ContentIterator Class

The **ContentIterator** class provides methods that you can use to perform operations that would otherwise be blocked by list view thresholds. The class works by breaking the query down into components, such as individual sites, lists, or list items, and then using callback methods to process one component at a time.

The following example shows how to use the **ContentIterator** class to submit a list query and process the results:

- Breaks queries down into manageable chunks
- Uses callback methods to process individual items

```
var iterator = new ContentIterator();
iterator.ProcessListItems(list, query, ProcessItem,
    ProcessError);

private void ProcessItem(SPListItem item)
{
    // Process the individual list item.
}

private void ProcessError(SPListItem item, Exception ex)
{
    // Log the error.
}
```

Using the ContentIterator Class

```
private void UseIterator()
{
    var web = SPContext.Current.Web;
    SPList list = web.Lists["Large List"];

    var query = new SPQuery();
    query.Query = @"[CAML query text]";

    var iterator = new ContentIterator();
    iterator.ProcessListItems(list, query, ProcessItem, ProcessError);
}

private void ProcessItem(SPListItem item)
{
    // Process the individual list item.
}

private void ProcessError(SPListItem item, Exception ex)
{
    // Log the error.
}
```

In this example, the **ProcessItem** method is called for each list item returned by the query. Execution does not proceed past the **ProcessListItems** method until all items have been processed.

Lab B: Working With Large Lists

Scenario

The finance team at Contoso has warned that the capital expenditure approval list may grow beyond 5,000 items. To avoid encountering problems with list query throttling limits, and the adverse performance impacts of submitting large list queries, you must modify the Web Part you created in the first exercise to use the **ContentIterator** class.

Objectives

After completing this lab, you will be able to use the **ContentIterator** class to submit queries to SharePoint lists.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-03
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

Exercise 1: Using the ContentIterator Class

Scenario

In this exercise, you will modify the Expense Checker Web Part to support large lists. You will edit the code that queries the capital expenditure approval list so that it retrieves and processes one list item at a time, thereby preventing any issues arising from list throttling limits.

The main tasks for this exercise are as follows:

1. Add Assembly References and Using Statements
2. Create a Data Table to Store Query Results
3. Create the ProcessItem Method
4. Create the ProcessError Method
5. Configure a ContentIterator Instance
6. Test the Web Part

► Task 1: Add Assembly References and Using Statements

- Connect to the 20488B-LON-SP-03 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Visual Studio 2012, open the solution at **E:\Labfiles\Starter\ExpenseChecker_LabB\ExpenseChecker.sln**.
- In the **ExpenseChecker** project, add a reference to the **Microsoft.Office.Server** assembly.
- In the code-behind file for the Expense Checker Web Part, add a **using** statement for the **Microsoft.Office.Server.Utilities** namespace, and the **System.Data** namespace.

► Task 2: Create a Data Table to Store Query Results

- Within the **ExpenseCheckerWebPart** class, declare a variable of type **DataTable** named **dtable**.
- In the **OnInit** method, after the existing code, instantiate **dtable** as a new **DataTable**.

- Add the following columns to the data table:
 - a. A column of type **String** named **UniqueId**
 - b. A column of type **String** named **CapExRequestor**
 - c. A column of type **String** named **CapExCategory**
 - d. A column of type **String** named **CapExDescription**
 - e. A column of type **Decimal** named **CapExAmount**
- ▶ **Task 3: Create the ProcessItem Method**
- Within the **ExpenseCheckerWebPart** class, add a method named **ProcessItem**. The method should:
 - a. Accept a single argument of type **SPListItem**.
 - b. Return void.
- Within the **ProcessItem** method, retrieve the following information from the passed-in list item:
 - a. The unique identifier, as a string
 - b. The display name of the requestor, as a string
 - c. The expense category, as a string
 - d. The expense description, as a string
 - e. The amount, as a decimal
- Add a new row to the **dtable** data table that contains the five values you retrieved in the previous step.
- ▶ **Task 4: Create the ProcessError Method**
- Within the **ExpenseCheckerWebPart** class, add a method named **ProcessError**. The method should:
 - a. Accept two arguments, of types **SPListItem** and **Exception**, respectively.
 - b. Return bool.
- Within the **ProcessError** method, throw a new exception that includes the exception parameter as an inner exception.
- ▶ **Task 5: Configure a ContentIterator Instance**
- In the **OnPreRender** method, locate and delete the following code:

```
var items = list.GetItems(query);
lstExpenses.DataSource = items.GetDataTable();
lstExpenses.DataBind();
```
- Create a new **ContentIterator** instance.
- Configure the **ContentIterator** instance to run the existing query object on the **Expenditure Requests** list. The operation should:
 - a. Use the **ProcessItem** method to process each list item.
 - b. Use the **ProcessError** method to process any errors.

► **Task 6: Test the Web Part**

- On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, when the page loads, verify that the Expense Checker Web Part behaves as before.
- Close Internet Explorer and close Visual Studio.

Results: After completing this exercise, you should have modified the Expense Checker Web Part to use the **ContentIterator** class.

Module Review and Takeaways

In this module, you learned about how to work with SharePoint lists and libraries in the server-side SharePoint object model. First, you learned about how to use the core classes that represent SharePoint lists and libraries. You learned about how to build CAML queries, and how to retrieve list data using query classes and LINQ to SharePoint. You also gained an understanding of how SharePoint manages the performance of large lists, and how you can work with this functionality in your code.

Review Question(s)

Test Your Knowledge

Question	
Which class represents a column in a SharePoint list?	
Select the correct answer.	
<input type="checkbox"/>	SPField
<input type="checkbox"/>	SPFieldValue
<input type="checkbox"/>	SPListItem
<input type="checkbox"/>	SPFile
<input type="checkbox"/>	SPFolder

Test Your Knowledge

Question	
You have created and configured an SPQuery object named query. You have also retrieved an SPList instance named list. You want to use the query object to retrieve data from the list. Which code sample should you use?	
Select the correct answer.	
<input type="checkbox"/>	<code>var items = query.Execute(list);</code>
<input type="checkbox"/>	<code>var items = list.GetSiteData(query);</code>
<input type="checkbox"/>	<code>var items = query.GetItems(list);</code>
<input type="checkbox"/>	<code>var items = list.GetItems(query);</code>
<input type="checkbox"/>	<code>var items = list.ExecuteQuery(query);</code>

Test Your Knowledge

Question	
What is the default list view threshold for regular users?	
Select the correct answer.	
<input type="checkbox"/>	200
<input type="checkbox"/>	500
<input type="checkbox"/>	2,000
<input type="checkbox"/>	5,000
<input type="checkbox"/>	20,000

Module 4

Designing and Managing Features and Solutions

Contents:

Module Overview	4-1
Lesson 1: Understanding Features and Solutions	4-2
Lesson 2: Configuring Features and Solutions	4-12
Lesson 3: Working with Sandboxed Solutions	4-23
Lab: Working With Features and Solutions	4-31
Module Review and Takeaways	4-38

Module Overview

Developing a SharePoint solution can involve creating several different types of custom components. There are many different ways in which you can customize SharePoint, and you will often need to make several customizations to meet a specific business requirement. As you develop custom functionality, it is essential that you have a strategy for deploying your solution that ensures both the integrity of the solution and the security and stability of the SharePoint farm. In this module, you will learn about how you can create Features and solutions to package and deploy your customizations, how you can minimize your impact on the SharePoint farm, and how you can ensure that upgrades are performed smoothly.

Objectives

After completing this module, you will be able to:

- Explain the purpose and key functionality of Features and solutions.
- Configure and manage Features and solutions.
- Create and manage sandboxed solutions.

Lesson 1

Understanding Features and Solutions

Any extensible platform, such as SharePoint, must provide a way of enabling administrators, developers, and end users to manage their customizations. Administrators and end users must be able to enable and disable custom components at various levels of the SharePoint hierarchy. Administrators and developers must be able to manage the life cycle of custom components, from initial deployment, through upgrades, to removal.

SharePoint provides this functionality through Features and solutions. In this lesson, you will learn about Features and the solution framework in SharePoint 2013.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the core concepts of Features.
- Describe the structure of a Feature.
- Explain the purpose of Feature scopes.
- Explain the core concepts of solutions.
- Describe the structure of a solution.

Understanding Features

A Feature is a set of files that provisions custom SharePoint functionality or components. Features are targeted to a specific scope in the SharePoint logical architecture—the farm level, the web application level, the site collection level, or the site level—according to the type of functionality that the Feature provides. Administrators at the specified scope can make the Feature functionality available to their users by activating the Feature, and they can remove the functionality by deactivating the Feature. In this way, Features support a modular approach to customizations by enabling administrators to turn on and turn off customizations.

- What is a Feature?
 - A way of defining and scoping declarative components
 - A mechanism for turning components on and off
- What can you include in a Feature?
 - Any declarative component
- How are Features deployed?
 - Manually
 - In a SharePoint solution (farm or sandboxed)
 - In an app for SharePoint

What is a Feature?

In SharePoint, a Feature is a set of one or more XML files. These XML files, which are known as *elements files* or *element manifests*, use Collaborative Application Markup Language (CAML) to define various SharePoint components, such as site columns or content types.

Every Feature includes a file named Feature.xml. This file is often referred to as the *feature manifest*. This file performs two key functions:

- It identifies the individual element manifests that belong to the Feature.
- It identifies the scope of the Feature.

For example, suppose you create and install a Feature at the site collection scope. That Feature is then made available on every site collection in the SharePoint farm. Site collection administrators can provision

the functionality defined by the Feature by *activating* the Feature. They can subsequently remove the functionality defined by the Feature by *deactivating* the Feature.

Features also provide a range of more sophisticated functionality. For example, you can define dependencies between Features, manage Feature versioning, and create code that responds to Feature events. You will learn more about these capabilities in the next lesson.

What can you include in a Feature?

You can use a Feature to deploy any SharePoint component that can be defined declaratively using CAML. The following table describes the different types of components that you can include in a Feature.

Component	Description
Field Definition	Defines a new site column.
Content Type Definition	Defines a new content type. Specifies the columns that the content type should include, along with various other content type properties.
Content Type Binding	Associates an existing content type with an existing list instance.
List Instance	Creates an instance of a list based on an existing list template.
List Template	Defines a new list template (also known as a list definition).
Site Definition	Defines a new site definition.
Workflow Definition	Defines a new workflow. In SharePoint 2013, workflows are entirely declarative and scoped to the site collection, which means that you can deploy workflows within features.
Event Registration	Associates an event receiver assembly with a specific site collection or site.
Feature/Site Template Association	Associates an existing Feature with an existing site definition, so that the Feature is provisioned when new instances of the site definition are created. This process is known as <i>Feature stapling</i> .
Module	Deploys files to a SharePoint site or document library. You can use a module to deploy documents, images, JavaScript files, Web Part definition files, and more.
Delegate Control	Registers a custom control (.ascx file) for use in an area where delegate controls are supported. For example, you could create a delegate control to provide an alternative search box for SharePoint pages.
Document Converter	Registers an executable and an associated webpage that provide functionality to convert one file type to another.

How are Features deployed?

You can deploy a Feature in three different ways:

- *Manually copy the Feature files to the SharePoint file system.* You can copy the Feature folder to the 15\TEMPLATE\FEATURES folder in the SharePoint server file system and then use Windows PowerShell to install the Feature. If you have more than one SharePoint server, you must copy the Feature files to the file system on each server in the SharePoint farm. For this reason, manual deployment of Features is no longer a recommended approach.

- *Include the Feature in a farm solution or a sandboxed solution.* You can use SharePoint solutions to deploy and retract Features. The Features are installed when the solution is added and uninstalled when the feature is removed. Features deployed within solutions are automatically synchronized across every server in the farm.
- *Include the Feature in an app for SharePoint.* Features are used within app packages to provision declarative components on the app web and the host web.

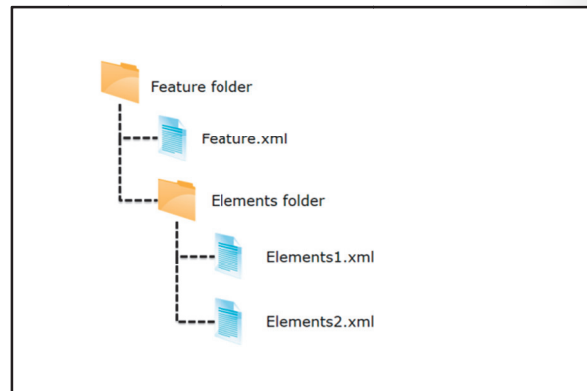
In earlier versions of SharePoint, Features were always deployed to the server file system. Built-in Features and Features provisioned by farm solutions are still deployed in this way—you can explore the built-in Features by browsing to the 15\TEMPLATE\FEATURES folder on any SharePoint server. By contrast, Features provisioned by sandboxed solutions or apps for SharePoint are deployed to the content database.

Anatomy of a Feature

When you create a Feature, every component of the Feature is contained within a top-level Feature folder. The name of this folder is important, because this is the name by which you will refer to the Feature if you install and uninstall it by using Windows PowerShell.



Note: Although you can manually create a Feature structure, it is more common to use the Visual Studio 2012 Feature Designer. You will see how to use the Feature designer in the next lesson.



The Feature manifest file

The Feature manifest file is created at the root of the Feature folder. The Feature manifest file must always be placed in the root Feature folder, rather than in any subfolders, and the Feature manifest file is always named Feature.xml. The Feature manifest file performs several important roles:

- It defines a unique identifier, a title, and a description for the Feature.
- It defines the scope of the Feature.
- It identifies the element manifest files that belong to the Feature, and specifies their relative location within the Feature folder.

The following code example shows a feature manifest file. This is taken from the built-in DocumentLibrary feature, which defines the Document Library list template:

Example Feature Manifest File

```
<Feature Id="00BFEA71-E717-4E80-AA17-D0C71B360101"
  Title="$Resources:core,documentlibraryFeatureTitle;"
  Description="$Resources:core,documentlibraryFeatureDesc;"
  Version="1.0.0.0"
  Scope="Web"
  Hidden="TRUE"
  DefaultResourceFile="core"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="ListTemplates\DocumentLibrary.xml" />
  </ElementManifests>
</Feature>
```

The key points in this example are as follows:

- *The Feature is scoped at the web level.* This means that the Feature can be activated and deactivated on individual sites.
- *The Feature is hidden.* This means that the Feature cannot be managed through the user interface. In this example, the DocumentLibrary feature is provisioned and activated by various site definitions.
- *The Feature includes a single element manifest.* The Feature manifest references a single element manifest file named DocumentLibrary.xml, which is in a subfolder named ListTemplates within the top-level Feature folder.

Element manifest files


Features can contain one or more element manifest files. Element manifest files are often organized in subfolders within the top-level Feature folder. However, this is not mandatory—beneath the top-level Feature folder and the Feature manifest file, you can structure your Feature any way you want. You can also choose any name you want for subfolders and element manifest files.

The following code example shows an element manifest file. This is taken from the built-in DocumentLibrary feature:

Example Element Manifest File

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ListTemplate
    Name="doclib"
    Type="101"
    BaseType="1"
    OnQuickLaunch="TRUE"
    SecurityBits="11"
    Sequence="110"
    DisplayName="$Resources:core,doclibList;"
    Description="$Resources:core,doclibList_Desc_15;"
    Image="/_layouts/15/images/itdl.png?rev=23"
    DocumentTemplate="121"
  </ListTemplate>
</Elements>
```

The top-level XML element in an element manifest file is always an **Elements** element. The Elements element supports various child elements that you can use to define different types of functionality.

 **Additional Reading:** For more information about the SharePoint Feature schemas, see *SharePoint Features schemas* at <http://go.microsoft.com/fwlink/?LinkID=306795>.

Understanding Feature Scopes

Every SharePoint Feature must be scoped to a level in the SharePoint logical architecture. The scope defines the level at which the Feature must be activated and where the functionality defined by the Feature is available.

The scope of a Feature is defined by the **Scope** attribute of the **Feature** element in the Feature manifest file. This attribute can take four possible values:

- Farm
- WebApplication
- Site
- Web

```
<Feature Id="..."
  Title="..."
  Description="..."
  Version="..."
  Scope="Web">
  ...
</Feature>
```

- *Farm*. The Farm scope specifies that the functionality contained within the Feature is available to all web applications, site collections, and sites within the farm. Farm-scoped Features must be activated by a farm administrator. Often, the Farm scope is used for Features that are not directly applicable to a particular site. For example, a Farm-scoped Feature might be used to install a timer job that interacts with all site collections, or directly with the content databases, or to add a new site definition that should be available to all site collections in all web applications within the farm.
- *WebApplication*. The WebApplication scope applies to all of the sites collections and sites contained within a specific web application. WebApplication-scoped Features must be activated by a farm administrator. Examples of Features with the WebApplication scope include Features that set security permissions at the web application level and Features that register event receivers for site collection events.
- *Site*. The Site scope scopes the Feature to an individual site collection. Site-scoped Features must be activated by the site collection administrator. The components installed by the Feature are then available to all sites within that site collection. Site owners of subsites cannot activate a Feature with the Site scope; they must be activated by the site collection administrator at the site collection level.
- *Web*. The Web scope is the most restrictive of scopes because it applies to a single site; however, it can be activated by a site owner. An example of a Web-scoped Feature (and a Site-scoped Feature) includes a Feature that deploys a content type or custom list definition.

Choosing the most appropriate scope is an important design consideration when you create a Feature, and you will learn more about this in the next lesson. It is important to understand that you do not scope a Feature to a specific web application, site collection, or site. Instead, if you deploy a WebApplication-scoped Feature, the Feature is available for activation on every web application in the SharePoint farm. The administrator can choose to activate the Feature if the functionality is relevant to that specific web application.

Demonstration: Exploring a Feature

The instructor will now explore some of the built-in Features in SharePoint 2013. This demonstration will help you to understand:

- How Features are structured.
- How Feature manifest files and element manifest files are configured.
- How activation dependencies and Feature receivers are configured.

Demonstration Steps

- Open a File Explorer window and browse to the **C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15** folder.
- Explain that the **15** folder is the root of the SharePoint file system, commonly known as the *SharePoint root* or the *SharePoint hive*.
- Within the **15** folder, browse to the **TEMPLATE\FEATURES** folder.



Note: Features within the FEATURES folder are either built-in Features, Features that were deployed manually, or Features that were deployed within a farm solution.

- Browse to the **ctypes** folder. This is a built-in Feature that provisions a range of built-in content types.
- Right-click **feature.xml**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- Notice the following key aspects of the Feature.xml file:
 - a. The feature is scoped to the **Site** (site collection) level.
 - b. The feature is hidden from the user interface.
 - c. The feature references three element manifest files, named **ctypeswss.xml**, **ctypeswss2.xml**, and **ctypeswss3.xml**.
- Briefly review the upgrade actions and versioning sections. You will learn more about these capabilities in the next lesson.
- Switch back to File Explorer.
- Double-click **ctypeswss.xml**.
- Briefly review the contents of the file. Notice that the file defines several built-in content types, such as **Item**, **Document**, **Event**, and **Announcement**.
- Notice that one element manifest file can define multiple elements.
- Close Visual Studio 2012.
- In File Explorer, browse back to the **FEATURES** folder, and then browse to the **BaseSite** folder.
- Notice that this Feature does not include any element manifest files.
- Double-click **feature.xml**.
- Notice that instead of specifying element manifest files, the Feature specifies a series of activation dependencies. When this Feature is activated, the dependency Features are automatically activated. Feature dependencies are covered in more detail in the next lesson.
- Close Visual Studio 2012.
- In File Explorer, browse back to the **FEATURES** folder, and then browse to the **SiteAssets** folder.
- Notice that this Feature does not include any element manifest files.
- Double-click **feature.xml**.
- Notice that instead of specifying element manifest files or activation dependencies, this Feature just registers a Feature receiver assembly that responds to Feature life cycle events. Feature receivers are covered in more detail in the next lesson.
- Close all open windows.

Understanding Solutions

In reality, you will rarely, if ever, manually deploy a Feature. There are two main reasons why manually deploying Features is not a good idea:

- You must manually add the Feature folder to the SharePoint file system, and remove it when it is no longer required.
- You must manually copy the Feature folder to every server in the SharePoint farm, and make sure the Feature contents remain synchronized on every server.

- What is a solution?
 - A cabinet file with a .wsp solution
 - Farm solutions
 - Sandboxed solutions
 - Solutions in app packages
- What can you include in a solution?
 - Features
 - Assemblies (sometimes) and files
 - Configuration settings
- How are solutions deployed?
 - Windows PowerShell (farm solutions)
 - Windows PowerShell or site collection UI (farm and sandboxed solutions)
 - Within an app package

Fortunately, SharePoint provides a more satisfactory way of managing the life cycle of custom components in the form of the SharePoint solution package.

What is a solution?

In SharePoint terminology, a solution is essentially a cabinet file with a .wsp file name extension. The solution framework enables you to package, deploy, and manage the life cycle of various custom components for SharePoint.

There are three types of solutions in SharePoint 2013:


- *Farm solutions.* Farm solutions are deployed at the farm scope by a farm administrator. You can use a farm solution to package and deploy any custom functionality to SharePoint, including fully-trusted server-side code.
- *Sandboxed solutions.* Sandboxed solutions, also known as user solutions, are deployed at the site collection scope by a site collection administrator. Any custom code within a sandboxed solution runs under an isolated process with limited permissions, and sandboxed solutions are subject to various monitoring and throttling constraints. You will learn more about sandboxed solutions in the final lesson in this module.
- *Solutions within apps for SharePoint.* When you create an app for SharePoint, any components that you want to deploy to the app web are packaged in a solution. This solution package cannot contain any server-side code. You will learn more about app packages later in this course.

The big advantage of using solutions is that they provide clean deployment, activation, and removal of custom components. For example, if you deploy a Feature within a farm solution, SharePoint will ensure that the Feature files are added to every web server in the server farm. When you retract the solution, SharePoint will remove the Feature files from every web server in the server farm. More broadly, any changes that are made when you activate a solution will be reversed when you retract the solution.

What can you include in a solution?

You can use a solution package to deploy two types of component to a SharePoint environment:

- *Features.* You can use solution packages to deploy Features by including the entire contents of the Feature, including the Feature manifest file, within the solution package.
- *Assemblies.* If your custom components include server-side code, you can use the solution package to deploy the assembly to the global assembly cache or the bin folder for a specific IIS web application. In sandboxed solutions, assemblies are stored in the content database and loaded by an isolated worker process.

 **Note:** Assemblies that are deployed to the global assembly cache are automatically granted full trust. Deploying assemblies to the bin folder for a specific web application was often used in previous versions of SharePoint, because it enabled administrators to apply code access security (CAS) restrictions to the assembly. The bin deployment approach is no longer commonly used because there are alternative approaches to providing a restricted permission set.

Every solution package includes a solution manifest file named Manifest.xml that specifies what the package contains. You can also use the solution manifest file to perform various configuration tasks, such as adding safe control entries to the Web.config file when you deploy an assembly. You will learn more about solution manifest files in the next topic.

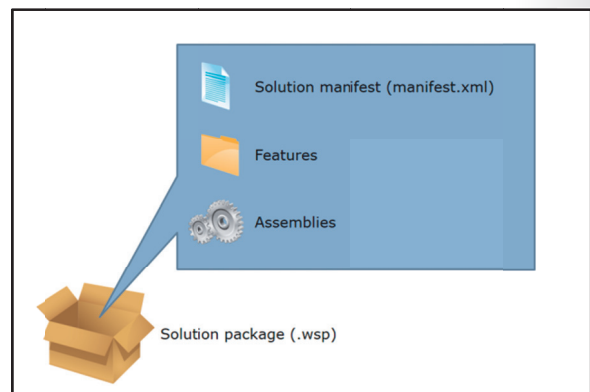
How are solutions deployed?

To deploy a farm solution, a farm administrator must add the solution to the farm using Windows PowerShell, and then install it using either Windows PowerShell or the Central Administration website. To deploy a sandboxed solution, a site collection administrator must add the solution to the Solutions gallery on a site collection. Farm administrators can also deploy sandboxed solutions to specific site collections using Windows PowerShell.

Solutions contained within app packages are deployed automatically as part of the app installation process.

Anatomy of a Solution

A SharePoint solution package is a cabinet file with a .wsp file name extension. The contents and behavior of the solution are specified by the solution manifest file, which is an XML file named Manifest.xml.



The following code example shows a typical solution manifest file:


The Solution Manifest File

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionID="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  SharePointProductVersion="15.0">
  <FeatureManifests>
    <FeatureManifest Location="ContosoProject\feature.xml" />
  </FeatureManifests>
  <RootFiles>
    <RootFile Location="Resources\contoso.resx" />
    <RootFile Location="Resources\contoso.en-US.resx" />
    <RootFile Location="Resources\contoso.en-ES.resx" />
  </RootFiles>
  <TemplateFiles>
    <TemplateFile Location="Controls\ContosoLogIn.ascx" />
  </TemplateFiles>
  <Assemblies>
    <Assembly Location="ContosoSolution.dll" DeploymentTarget="GlobalAssemblyCache">
      <SafeControls>
        <SafeControl Assembly="[Assembly strong name]"
          Namespace="[Namespace]"
          TypeName="*" />
      </SafeControls>
    </Assembly>
  </Assemblies>
</Solution>
```

This example illustrates the deployment of various different types of custom component. For example:

- *Features.* To include a Feature in a solution, add a **FeatureManifest** element for each Feature. The **Location** attribute indicates the relative path to the Feature manifest file within the solution package.
- *Resources files.* To include localization resources in a solution, add a **RootFile** element for each resource file. The **Location** attribute indicates the relative path to the resource file within the solution package.
- *Template files.* To include template files in a solution, add a **TemplateFile** element for each template file. Template files are typically items that you want to add to the server file system, such as ASCX control files or default webpages. The **Location** attribute indicates the relative path to the template file within the solution package.
- *Assemblies.* To include assemblies in a solution, add an **Assembly** element for each assembly. The **Location** attribute indicates the relative path to the assembly within the solution package, and the **DeploymentTarget** attribute specifies whether you want to deploy the assembly to the global assembly cache or the web application bin folder. You can use a **SafeControl** child element to register classes within the assembly, such as Web Part classes, as safe controls in the Web.config file.

You can also use the solution manifest file to configure activation dependencies between solutions. You will learn more about activation dependencies in the next lesson.

 **Additional Reading:** For more information about solution manifest files, see *Solution schema* at <http://go.microsoft.com/fwlink/?LinkID=323477>.

Demonstration: Exploring a Solution

The instructor will now explore the contents of a solution package. In this demonstration, you will see:

- How to explore the contents of a .wsp file by changing the file extension.
- How to review the contents of a solution manifest file.

Demonstration Steps

- Open a File Explorer window and browse to **E:\Democode\ExpenseChecker**.
- Right-click **ExpenseChecker.sln**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.



Note: This is the Expense Checker Web Part solution from the lab in the previous module.

- On the **BUILD** menu, click **Rebuild Solution**.
- On the **BUILD** menu, click **Publish**.
- In the **Publish** dialog box, under **Target Location**, click the ellipsis button.
- In the **Select Target Location** dialog box, browse to the desktop, and then click **Select Folder**.
- In the **Publish** dialog box, click **Publish**.
- Close Visual Studio.
- On the desktop, right-click **ExpenseChecker.wsp**, and then click **Rename**.
- Change the file name to **ExpenseChecker.wsp.cab**, and then press Enter.
- In the **Rename** dialog box, click **Yes**.
- Double-click **ExpenseChecker.wsp.cab**.
- In the File Explorer window, copy all the files, paste them onto the desktop, and then close the File Explorer window.
- On the desktop, right-click **manifest.xml**, point to **Open with**, and then click **Microsoft Visual Studio 2012**. The Manifest.xml file is the solution manifest file.
- Notice that the manifest includes an **Assembly** element. This deploys the Web Part assembly to the global assembly cache.
- Notice that the manifest includes a **SafeControl** element. This registers the Web Part as a safe control in the Web.config file.
- Notice that the manifest includes a **FeatureManifest** element. **FeatureManifest** elements identify Features that are included in the solution.
- Close Visual Studio.
- On the desktop, double-click **elements.xml**. The elements.xml file is the element manifest file for the Feature that is included in the solution.
- Notice the element manifest file uses a **Module** element to deploy the Web Part control description file (ExpenseChecker.webpart) to the Web Parts gallery on the site collection when the Feature is activated.
- Close Visual Studio.
- Delete all the files that you added to the desktop.

Lesson 2

Configuring Features and Solutions

Both Features and the solution framework provide a range of functionality for administering and managing the life cycle of custom components. In this lesson, you will learn how you can leverage this functionality to manage your own Features and solutions.

Lesson Objectives

After completing this lesson, you will be able to:

- Plan and configure Feature dependencies.
- Configure solution activation dependencies.
- Configure and manage Feature upgrades.
- Configure and manage solution upgrades.
- Manage Feature and solution life cycles.
- Create and register Feature receiver classes.

Planning and Configuring Feature Dependencies

The built-in Features in SharePoint 2013 use Feature activation dependencies extensively. Feature activation dependencies enable you to require the activation of an existing Feature before an administrator can activate a new Feature. Dependencies are a useful tool when you design your own Feature sets.

Dependency scenarios

Feature activation dependencies are used in two main scenarios:


- *Grouping*. You can use dependencies to group multiple granular Features together in a single Feature. Suppose you create a set of hidden Features that each deploy a site column. You could then create one or more visible Features that depend on a subset of these hidden Features to deploy commonly-used sets of site columns, such as site columns for finance teams or project management tasks. When you activate the visible Feature, SharePoint will automatically activate the hidden Features that are included as dependencies. This approach enables you to define functionality at a very granular level but expose functionality at a business requirements level.
- *Resource guarantees*. You can use dependencies to make sure particular components are available before your Feature is activated. For example, suppose you create a Feature that binds a content type to a list instance. You could use Feature dependencies to make sure both the Feature that provisions the content type and the Feature that provisions the list instance are activated before your content type binding Feature is activated.

- Dependency scenarios
 - Grouping (automatic activation)
 - Resource guarantees (manual activation)
- Dependency rules
 - Same-scope dependencies
 - Cross-scope dependencies
 - Hidden Features
- Configuring dependencies
 - ActivationDependency element

Dependency rules

Feature dependencies are governed by a series of rules that are designed to prevent circular dependencies and performance issues. The most notable rules are as follows:

- *Dependency chains.* Generally speaking, dependency chains are not supported. If Feature A depends on Feature B, Feature B cannot include dependencies to any other visible Features. Feature B can include dependencies on hidden Features.
- *Hidden Features.* Typically, hidden Features are intended to be used as dependencies by other Features. Hidden Features cannot themselves include dependencies. SharePoint will automatically deactivate a hidden Feature when the last visible Feature that depends on it is deactivated.
- *Same-scope dependencies.* If Feature A depends on Feature B, and both Features are at the same scope, SharePoint will automatically activate Feature B when you activate Feature A.
- *Cross-scope dependencies.* Features can only depend on other Features at a less restrictive scope. For example, a Web-scoped Feature can include a dependency on a Site-scoped Feature, but a Site-scoped Feature cannot include a dependency on a Web-scoped Feature. SharePoint will never automatically activate Features referenced in cross-scope dependencies—instead, the activation will fail and the administrator will be prompted to install the dependency Feature first. You cannot create cross-scope dependencies on hidden Features.

 **Additional Reading:** For more information about activation dependency rules, see *Activation Dependencies and Scope* at <http://go.microsoft.com/fwlink/?LinkID=323478>. This article was written for SharePoint 2010, but the concepts are equally applicable to SharePoint 2013.


Configuring dependencies

When you define a Feature, you can specify dependencies on other Features by adding **ActivationDependency** elements to your Feature manifest file. Visual Studio 2012 includes a designer that can aid you in this process.

The following code example shows a Feature manifest file with two activation dependencies:

Defining Feature Activation Dependencies

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Title="FinanceAssets"
  Id="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
  Scope="Web">
  <ActivationDependencies>
    <ActivationDependency FeatureId="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
      FeatureTitle="FinanceSiteColumns" />
    <ActivationDependency FeatureId="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
      FeatureTitle="FinanceContentTypes" />
  </ActivationDependencies>
</Feature>
```

 **Note:** When you add an **ActivationDependency** element, you only need to specify a **FeatureId** attribute. The **FeatureTitle** attribute is optional and simply serves to make your Feature manifest file more comprehensible for other humans.

Creating Solution Activation Dependencies

In addition to creating Features that require the activation of one or more existing Features, you can create solutions that require the activation of one or more existing solutions. Solution activation dependencies are used far less commonly than Feature activation dependencies. This is because a solution, as the name suggests, typically represents a complete and self-contained set of components. By contrast, Features often represent more granular functionality.

To define a solution activation dependency, you must add an **ActivationDependency** element to the solution manifest file. You must provide a **SolutionId** attribute, and you can also specify an optional **SolutionName** attribute to make your manifest more readable. In contrast to Feature activation dependencies, the Visual Studio designer does not provide tooling for adding solution activation dependencies and you must manually edit the solution manifest.

The following code example shows a solution manifest file that includes an activation dependency:

- Less common than Feature activation dependencies
 - Solutions typically represent a self-contained solution
 - Features often represent granular functionality
- Add an **ActivationDependency** element to the solution manifest file

```
<Solution xmlns="...">
  <ActivationDependencies>
    <ActivationDependency SolutionId="..."
                          SolutionName="..." />
  </ActivationDependencies>
</Solution>
```

Defining Solution Activation Dependencies

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx"
  SharePointProductVersion="15.0">
  <ActivationDependencies>
    <ActivationDependency SolutionId="xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx"
                          SolutionName="ContosoFinanceTools" />
  </ActivationDependencies>
  <!-- Other solution manifest content -->
</Solution>
```

Demonstration: Using the Visual Studio Designer

The instructor will now demonstrate how to use the Visual Studio 2012 Feature Designer to add Feature activation dependencies.

Demonstration Steps

- Open a File Explorer window and browse to **E:\Democode\FeatureDependencies**.
- Right-click **FeatureDependencies.sln**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- In Solution Explorer, briefly review the contents of the solution. Point out that the solution contains:
 - Site columns named **ClientName**, **ContosoDepartment**, **InvoiceDueDate**, **InvoiceAmount**, and **InvoiceStatus**.
 - A content type named **ContosoInvoice**.
 - A list template named **Invoices** and an associated list instance named **InvoicesInstance**.



Note: The content type uses the site columns, and the list uses the content type.

- Notice that the solution does not currently contain any Features.
- In Solution Explorer, right-click **Features**, and then click **Add Feature**.
- In the **Title** box, type **Invoicing Resources**.
- In the **Scope** drop-down list, click **Site**.
- In the **Items in the Solution** list box, hold down Ctrl and click **ClientName**, **ContosoDepartment**, **ContosoInvoice**, **InvoiceDueDate**, **InvoiceAmount**, and **InvoiceStatus**.
- Click the single right arrow button, and then click **Save**.
- Close the **Feature1.feature** tab.
- In Solution Explorer, right-click **Feature1**, and then click **Rename**.
- Type **InvoicingResources**, and then press Enter.
- In Solution Explorer, right-click **Features**, and then click **Add Feature**.
- In the **Title** box, type **Invoices List**.
- In the **Scope** drop-down list, leave **Web** selected.
- In the **Items in the Solution** list box, hold down Ctrl and click **InvoicesInstance** and **Invoices**.
- Click the single right arrow button, and then click **Save**.
- At the bottom of the page, expand **Feature Activation Dependencies**, and then click **Add**.
- In the **Add Feature Activation Dependencies** dialog box, under **Add a dependency on features in the solution**, click **Invoicing Resources**, and then click **Add**.
- On the **Manifest** tab, notice that an **ActivationDependency** element has been added to the Feature manifest file.
- Click **Save**, and then close the **Feature1.feature** tab.
- In Solution Explorer, right-click **Feature1**, and then click **Rename**.
- Type **InvoicesList**, and then press Enter.
- In Solution Explorer, expand **Package**, and then click **Package.package**.
- Notice that the solution package includes both Features.
- On the **PROJECT** menu, click **FeatureDependencies Properties**.
- On the **FeatureDependencies** tab, in the list on the left of the page, click **SharePoint**.
- On the **Active Deployment Configuration** drop-down list, click **No Activation**, and then click **Save**.



Note: The **No Activation** deployment configuration means that Visual Studio will deploy the solution but will not attempt to automatically activate the Features.

- On the **BUILD** menu, click **Build Solution**.
- On the **BUILD** menu, click **Deploy Solution**.
- On the Start screen, click **Internet Explorer**.
- In the address bar, type **team.contoso.com**, and then press Enter.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.

- After the page finishes loading, on the **Settings** menu, click **Site settings**.
- On the Site Settings page, under **Site Actions**, click **Manage site features**.
- On the **Site Features** page, in the **Invoices List** row, click **Activate**.
- Notice the error message advising that the **Invoicing Resources** feature must be activated first.
- On the **Settings** menu, click **Site settings**.
- Under **Site Collection Administration**, click **Site collection features**.
- In the **Invoicing Resources** row, click **Activate**.
- On the site breadcrumb trail, click **Site Settings**.
- Under **Site Actions**, click **Manage site features**.
- On the **Site Features** page, in the **Invoices List** row, click **Activate**.
- Notice that the Feature now activates successfully because you activated the dependency Feature.
- On the Quick Launch navigation menu, under **Recent**, click **Invoices**.
- On the **Invoices** page, on the ribbon, on the **ITEMS** tab, notice that the **Contoso Invoice** content type is available on the **New Item** drop-down menu.
- Close all open windows.

Managing Feature Upgrades

In any organization, custom software components are likely to evolve over time. It is very rare that an organization will release version 1.0 of a component and then make no additional changes. SharePoint 2013 provides a mechanism for managing component evolution and version in the form of Feature upgrades. An upgraded Feature must be able to do two things:

- Where a previous version of the Feature is not activated, it must provision every component within the Feature.
- Where a previous version of the Feature is activated, it must modify the existing components and add new components as required.

SharePoint meets these requirements by enabling you to target upgrade actions to specific Feature versions.

Upgrading a Feature

You can use the following high-level steps to upgrade a Feature:

1. Add any new or modified files to the Feature folder.
2. Add new element manifest files for any new items. Do not modify the existing element manifest files.
3. In the Feature manifest file, set the **Version** attribute to a new version number.
4. In the Feature manifest file, add an **UpgradeActions** element.


- Upgrading a Feature
 1. Add new element manifests
 2. Add an **UpgradeActions** element to the Feature manifest
 3. Add one or more **VersionRange** child elements
 4. Add actions within each **VersionRange** element
- Feature upgrade actions
 - ApplyElementManifests
 - MapFile
 - AddContentTypeField
 - CustomUpgradeAction

5. Within the **UpgradeActions** element, add one or more **VersionRange** elements to target changes to specific Feature versions or version ranges.

The following code example shows a Feature manifest file that includes upgrade actions:

Upgrade Actions in a Feature Manifest File

```
<Feature Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  Title="Invoicing Resources"
  Scope="Site"
  Version="3.0.0.0">
  <UpgradeActions>
    <VersionRange BeginVersion="1.0.0.0" EndVersion="2.9.9.9">
      <ApplyElementManifests>
        <ElementManifest Location="Columns\InvoiceOwner.xml" />
        <ElementManifest Location="Columns\ClientContact.xml" />
      </ApplyElementManifests>
      <AddContentTypeField ContentTypeId="0x01..."
        FieldId="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        PushDown="True" />
      <AddContentTypeField ContentTypeId="..."
        FieldId="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        PushDown="True" />
    </VersionRange>
  </UpgradeActions>
  <ElementManifests>
    <!--Register all element manifests here as before. -->
  </ElementManifests>
</Feature>
```

 **Note:** In many cases, Features are deployed without specifying a **Version** attribute. In these instances, SharePoint treats the version number as **0.0.0.0**.

Feature upgrade capabilities

Within a **VersionRange** element, you can include the following child elements to perform various upgrade tasks:

- *ApplyElementManifests*. Use this to apply additional element manifests to a range of Feature versions. The element manifests specified here should be those that are missing from the specified range of Feature versions.
- *MapFile*. Use this element to replace an existing file with a new file in the specified range of Feature versions. Use the **FromPath** and **ToPath** to identify the locations of the existing file and the new file, respectively, within the Feature folder.
- *AddContentTypeField*. Use this element to add a new field to a site content type. The site content type you specify must be defined in the previous version of the Feature. Use the **PushDown** attribute to specify whether you want to cascade changes to list content types and inherited content types.
- *CustomUpgradeAction*. Use this element to specify Feature upgrade event receiver classes that you want to execute when the Feature is upgraded.

You do not have to include a **VersionRange** attribute in your Feature manifest file; all the preceding elements can be added directly to the **UpgradeActions** element if you want your actions to apply to all versions of the Feature. However, it is good practice to always specify a version range for readability and backward compatibility.

Managing Solution Upgrades

Like any software component, custom SharePoint solutions are likely to evolve over time. Just like Features, the SharePoint solution framework provides mechanisms to help ensure that solution updates are applied smoothly and reliably.

To create a new version of a solution, you simply build a new solution package (.wsp) with any required changes. You can then update deployed instances of the solution in one of two ways:

- Remove the existing solution and then deploy the updated version.
- Perform an in-place update by using the **Update-SPSolution** Windows PowerShell cmdlet.

Determining the most appropriate approach depends on various aspects of the SharePoint environment and the solution contents.

Uninstalling and then reinstalling the solution

Uninstalling the previous version of the solution and then installing the new version of the solution provides the most flexible approach to updates. When you update a solution in this way, you can choose to uninstall the solution and remove it from the solution store, or just uninstall the solution. If you choose to only uninstall the solution, but not to remove the solution from the solution store, you must use a new name and solution ID when you deploy the updated solution. If you remove the solution from the solution store, you can reinstall by using the same name and solution ID.

The major drawback of this approach is that it requires you to be able to uninstall the existing solution. In practice, this is often not possible. For example, if your solution includes a content type that has been deployed and used in a list, you will not be able to uninstall the solution while the content type remains in use. This is not possible without losing data. In these circumstances, you will usually need to perform an in-place update by using the **Update-SPSolution** cmdlet.

Performing an in-place update

To perform an in-place update, you use the **Update-SPSolution** cmdlet to update the solution. When you call the **Update-SPSolution** cmdlet, you should use the **Identity** parameter to specify the GUID of the solution and you should use the **LiteralPath** parameter to specify the location of the revised solution package.

Performing an in-place update enables you to update your solution without first uninstalling it. This prevents data loss, which would be necessary with an uninstall/reinstall solution update; however, in-place updates do not support any of the following changes:

- Adding or removing a Feature
- Changing the solution ID
- Changing the scope of a Feature
- Updating a Feature receiver
- Adding, removing, or editing an element manifest file
- Adding, removing or editing a **Property** element in the Feature.xml file

- Approaches to solution upgrades
 - Retract, uninstall, and then reinstall
 - Use the **Update-SPSolution** cmdlet
- Retracting a solution is not always possible
 - For example, when a content type in use
- Updating a solution has some restrictions
 - For example, you cannot add or remove Features
- Assembly binding redirection

If you need to make any of these changes, you must either perform an uninstall/reinstall update, or consider developing a new solution with your new changes. You can make the new solution dependent on the original solution if appropriate.

Assembly binding redirection

In many cases, when you update a solution, you will need to replace assemblies within the solution with new versions. When you update an assembly, you should increment the version number. However, this may break other components if they are dependent on the old version of the assembly.

In ASP.NET, you typically solve this problem by adding **bindingRedirect** elements to your Web.config files. In SharePoint, you can add **BindingRedirect** child elements to your solution manifest. For each **BindingRedirect** element that you specify in your solution manifest file, when you run the solution update, SharePoint adds a **bindingRedirect** element to the Web.config file of each web application on every server in the SharePoint farm.

The following code example shows a solution manifest file that includes assembly binding redirection:

Using Assembly Binding Redirection

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
  SharePointProductVersion="15">
  <Assemblies>
    <Assembly Location="ContosoAssembly.dll" DeploymentTarget="GlobalAssemblyCache">
      <BindingRedirects>
        <BindingRedirect OldVersion="1.0.0.0" />
      </BindingRedirects>
    </Assembly>
  </Assemblies>
  ...
</Solution>
```

Administering Features and Solutions

As a developer, you will not often have to administer Features and solutions. In a production environment, Features and solutions are managed by farm administrators or site collection administrators. In your development environment, Visual Studio 2012 largely automates the process for you. However, as a well-rounded SharePoint developer, you should understand the basic techniques for deploying and managing Features and solutions.

Administering farm solutions

To administer farm solutions, you need to understand the solution life cycle. To activate the functionality within a solution, a farm administrator must:

1. *Add the solution to the solution store.* You can use the **Add-SPSolution** cmdlet in Windows PowerShell to add the solution.
2. *Install the solution.* You can use the **Install-SPSolution** cmdlet in Windows PowerShell to install the solution. Alternatively, you can install the solution from the Central Administration website.

- Administering farm solutions
 - Add the solution to the store
 - Install the solution
 - Retract the solution
 - Remove the solution from the store
- Administering Features
 - Install the Feature
 - Activate the Feature
 - Deactivate the Feature
 - Uninstall the Feature

When you install a solution, SharePoint will automatically install any Features contained within the solution.

To remove a solution from the SharePoint farm, the farm administrator must:

1. *Retract the solution.* You can use the **Uninstall-SPSolution** cmdlet to retract a solution. Alternatively, you can retract the solution from the Central Administration website. SharePoint will not allow you to retract a solution if items within the solution, such as content types, are currently in use.
2. *Remove the solution from the solution store.* You can use the **Remove-SPSolution** cmdlet to remove a solution from the solution store. You can also use the Central Administration website to remove solutions.



Note: The administration process for sandboxed solutions differs from the administration process for farm solutions. You will learn more about sandboxed solutions in the next lesson.

Administering Features

Like solutions, making the functionality defined by Features available to users involves two steps:

1. *Install the Feature.* You can use the **Install-SPFeature** cmdlet in Windows PowerShell to do this.
2. *Activate the Feature.* You can use the **Enable-SPFeature** cmdlet in Windows PowerShell to do this. You can also activate Features from the Central Administration website (for farm or web application-scoped Features) or the SharePoint site user interface (for site collection or site-scoped Features).

To remove a Feature, the administrator must also perform two steps:

1. *Deactivate the Feature.* You can use the **Disable-SPFeature** cmdlet to do this. You can also deactivate Features through the Central Administration website or the SharePoint site user interface, depending on the scope of the Feature.
2. *Uninstall the Feature.* You can use the **Uninstall-SPFeature** cmdlet to do this.

In most cases, Features are deployed within solutions or app packages. In these cases, you do not need to install or uninstall the Feature. Features are installed when the parent solution or app package is installed, and Features are uninstalled when the parent solution or app package is uninstalled.

In contrast to the default experience when you deploy a solution from Visual Studio, installing a solution does not automatically activate any Features within the solution. Installing the solution makes the Features available at the specified scope. Administrators at the specified scope can then choose whether they want to activate the Features provided by the solution.

Creating Feature Receivers

When you deploy a Feature within a solution, you can add code that runs in response to Feature life cycle events. This is useful in a variety of scenarios, such as when you need to register custom components such as timer jobs or to programmatically modify configuration files.

Creating a Feature receiver class

To create a Feature receiver, add a class that inherits from **SPFeatureReceiver** to your solution. The abstract **SPFeatureReceiver** class includes methods that respond to various events in the Feature life cycle. You add functionality to your Feature receiver by overriding the appropriate base class method:

- Creating a Feature receiver
 - Inherit from the **SPFeatureReceiver** base class
 - Override appropriate method for life cycle event
- Registering a Feature receiver
 - Edit the Feature manifest


- To run code after a Feature is installed, override the **FeatureInstalled** method.
- To run code after a Feature is activated, override the **FeatureActivated** method.
- To run code when a Feature is upgraded, override the **FeatureUpgrading** method.
- To run code when a Feature is deactivated, override the **FeatureDeactivating** method.
- To run code when a Feature is uninstalled, override the **FeatureUninstalling** method.

The following code example shows a Feature receiver class. This example adds an announcement to a list when the Feature is activated:

Creating a Feature Receiver Class

```
public class InvoiceListEventReceiver : SPFeatureReceiver
{
    public override void FeatureActivated(SPFeatureReceiverProperties properties)
    {
        var web = properties.Feature.Parent as SPWeb;
        var listAnnouncements = web.Lists.TryGetList("Announcements");
        if (listAnnouncements != null)
        {
            var announcement = listAnnouncements.Items.Add();
            announcement["Title"] = "Invoice List Feature Activated!";
            announcement["Body"] = String.Format("{0} was activated at: {1}",
                properties.Definition.DisplayName,
                DateTime.Now.ToString());

            announcement.Update();
        }
    }
}
```

 **Best Practice:** If you use the **FeatureInstalled** method to make changes when a Feature is installed, you should also use the **FeatureUninstalling** method to reverse your changes if the Feature is uninstalled. Similarly, if you use the **FeatureActivated** method to make changes when a Feature is activated, you should use the **FeatureDeactivating** method to reverse the changes when the Feature is deactivated.

Registering a Feature receiver

After you create a Feature receiver class, you need to associate it with your Feature. You can do this by specifying **ReceiverAssembly** and **ReceiverClass** attributes in the Feature manifest file.

The following code example shows a Feature manifest file that registers an event receiver class:

Registering an Event Receiver

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
  Title="Invoice List"
  Scope="Site"
  ReceiverAssembly="[Assembly strong name]"
  ReceiverClass="[Fully-qualified class name]">
</Feature>
```

Discussion: Using Feature Receivers

The instructor will now lead a brief discussion around the following questions:

- In what scenarios might you create a Feature receiver?
- In what scenarios might you add a Feature receiver to an empty Feature?

- In what scenarios might you create a Feature receiver?
- In what scenarios might you add a Feature receiver to an empty Feature?

Lesson 3

Working with Sandboxed Solutions

Sandboxed solutions support a broader range of deployment scenarios than farm solutions, but they support a more limited range of functionality. In this lesson, you will learn about how sandboxed solutions work and the capabilities and constraints of the sandboxed solution framework. You will also gain an understanding of when sandboxed solutions are an appropriate deployment choice for your custom SharePoint components.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the purpose and key features of the sandboxed solution framework.
- Identify the main capabilities and constraints of sandboxed solutions.
- Manage installation, versioning, and upgrades for sandboxed solutions.
- Create and install a solution validator.

Understanding Sandboxed Solutions

Sandboxed solutions were introduced in SharePoint 2010. The aim of sandboxed solutions was to enable site collection administrators to deploy certain types of customization without requiring access to the servers or the intervention of a farm administrator. In particular, this enabled users of hosted environments, who are unable to deploy farm solutions, to enjoy some level of customization within their tenancies.

To mitigate the inherent risks in allowing users to deploy customizations without administrative approval, the sandboxed solution model provides various restrictions on functionality. Sandboxed solutions have no access to the server-side file system and changes beyond the scope of the site collection are not permitted. In addition, sandboxed solution code runs within an isolated worker process (the "sandbox") with a minimal set of permissions.

To summarize, sandboxed solutions support a broader range of deployment options than farm solutions, but they offer a more limited set of capabilities.

To summarize, sandboxed solutions support a broader range of deployment options than farm solutions, but they offer a more limited set of capabilities.

Sandboxed solution fundamentals

In terms of the structure and contents of the solution package, a sandboxed solution is identical to a farm solution. The differences between farm solutions and sandboxed solutions lie in how the solution is deployed and executed.

Deployment

Farm solutions are deployed to a server-side solution store by a farm administrator. By contrast, sandboxed solutions are deployed to a site collection-scoped Solutions gallery by a site collection administrator. The Solutions gallery is simply a specialized SharePoint list. As a result, sandboxed solutions can also be deployed to SharePoint Online.

- Sandboxed solution fundamentals
 - Deployment
 - Scope
 - Code execution
- Monitoring and throttling
 - Resource points
 - Resource measures
 - Daily limits
 - Per request limits

Like any SharePoint list, the contents of each Solutions gallery are stored in the content database associated with the site collection. Whereas the components of a farm solution are typically deployed to the SharePoint file system on the web servers in the server farm, the components of a sandboxed solution are stored in the content database. This means that the sandboxed solution requires no interaction with the server-side SharePoint file system.

Scope

Farm solutions can access resources and make changes at any level of the SharePoint hierarchy: farm level, web application level, site collection level, or site level. By contrast, sandboxed solutions can only access resources and make changes within the scope of the current site collection. One impact of this constraint is that a sandboxed solution can only contain Site-scoped or Web-scoped Features.

Code execution

Assemblies within farm solutions are deployed either to the global assembly cache or to the bin folder associated with the IIS web application. When code within the assembly is called, the assembly is loaded by the calling process. For example, if the code is invoked as part of a web request, the assembly is loaded by the W3WP.exe process.


By contrast, assemblies within sandboxed solutions remain in the solution package within the content database. When code within the assembly is called, the assembly is loaded into an isolated worker process named SPUCWorkerProcess.exe. This process runs with a highly restricted set of permissions, which reduces the chances of sandboxed code inadvertently or maliciously compromising the performance of the farm as a whole.

Monitoring and throttling

To prevent sandboxed solutions from consuming excessive server resources, SharePoint monitors, and where necessary, throttles, user code processes through a system of *resource points*. Sandbox solutions consume resource points based on 14 different *resource measures*, such as memory consumption, processor time, and the number of database queries. For example, 20 database queries correspond to one resource point. Farm administrators can set limits on how many resource points the sandboxed solutions within a specified site collection can consume within 24 hours (this is set to 300 points by default). If the site collection reaches this threshold, SharePoint does not permit that site collection to run any further sandboxed code for the remainder of the 24-hour period.

In addition to this cumulative resource limit, SharePoint applies limits to the resources that a single sandboxed code request can consume. For example, if a single request takes more than 30 seconds to execute, SharePoint will terminate the request. Each resource measure also includes an absolute limit. If a single request reaches one of these limits, the entire user code process is terminated. This terminates not only the current request, but also any other requests that are currently running within the same user code process. For example, if a request reaches 100 percent of processor time, SharePoint will immediately terminate the user code process.

Administrators can configure every aspect of resource point calculation, monitoring, and throttling through Windows PowerShell.

 **Additional Reading:** For comprehensive information about sandboxed solutions, see *Sandboxed Solutions* at <http://go.microsoft.com/fwlink/?LinkID=306982>. This article was written for SharePoint 2010, but the principles are equally applicable to SharePoint 2013.

Capabilities and Constraints

Before you can choose an appropriate deployment model for your custom components, you need a solid understanding of what you can and cannot do with each approach. This topic provides an overview of the capabilities and constraints of the sandboxed solution framework.



Note: When you create a SharePoint solution in Visual Studio 2012, you specify whether you want to create a farm solution or a sandboxed solution. When you select the sandboxed solution option, the Visual Studio compiler will warn you when you attempt to include code or components that are not supported in the sandbox environment.

- Supported operations
 - Limited subset of Microsoft.SharePoint.dll
 - No file system access
 - No network access
 - No elevated permissions
- Supported components
 - Any declarative component within a Site-scoped or Web-scoped Feature
 - Some programmatic components
- Full-trust proxies

Supported operations

Assemblies within sandboxed solutions are loaded and executed by the SPUCWorkerProcess.exe process. This process is granted a low-privileged security token and is subject to a restrictive code access security (CAS) policy. Within sandboxed code, you can use a limited subset of the namespaces within the Microsoft.SharePoint.dll assembly, together with various common .NET types. Code within sandboxed solutions cannot:

- Read or write to the file system.
- Call web services or access the network.
- Call assemblies that are not deployed to the global assembly cache or do not allow partially-trusted callers.
- Use types within the Microsoft.SharePoint.Administration namespace.
- Run code with elevated privileges.



Note: The restrictions described here apply only to server-side code. Where applicable, you can use the JavaScript object model or the REST API within sandboxed solutions—for example, in the output of a Web Part—without restriction.

Supported components

You can deploy a variety of components within a sandboxed solution. You can deploy any declarative component that can be provisioned within a Site-scoped or Web-scoped Feature. You can also deploy a range of code-based components, provided that the code contained within the components does not violate the constraints of the sandbox environment. For example, you can use a sandbox solution to deploy:

- Web Parts and visual Web Parts.
- Custom SharePoint 2013 workflow activities.
- Declarative components, such as list definitions and list templates, content types, and site columns.
- Web event receivers.
- List event receivers.

- List item event receivers.
- Customizations to the ribbon and the Edit Control Block (ECB).

You cannot deploy components that are executed beyond the scope of the site collection. For example, you cannot deploy timer jobs, user controls, or application pages. You also cannot deploy Feature receivers or site collection event receivers.

Hybrid solutions

SharePoint enables you to extend the functionality available to sandboxed solutions by creating and deploying a *full-trust proxy*. Full-trust proxy assemblies are deployed as farm solutions, but they expose functionality that can be consumed by sandboxed solutions.



Reference Links: The patterns & practices team at Microsoft created a full-trust proxy to enable sandboxed code to write to event logs and trace logs. For more information, see *The SharePoint Logger* at <http://msdn.microsoft.com/en-us/library/ff798395.aspx>.

To create a full-trust proxy, you create a class that inherits from the **SPPProxyOperation** abstract base class. The resulting assembly must be deployed to the global assembly cache. You must also register your class as a trusted assembly. You can do this programmatically, for example from code within a Feature receiver, or by using Windows PowerShell.

To call a full-trust proxy from sandboxed code, you must first use the **SPPProxyOperationsArgs** class to structure the arguments that you want to pass to the proxy. You can then invoke the proxy operation by calling the **SPUtility.ExecuteRegisteredProxyOperation** method.



Additional Reading: For more information about full-trust proxies, see *Hybrid Approaches* at <http://go.microsoft.com/fwlink/?LinkID=323479>.

Managing the Sandboxed Solution Life Cycle

Sandboxed solutions are typically managed by site collection administrators. Although the deployment and management process for sandboxed solutions is similar to the process for farm solutions, there are some key differences that you need to be aware of.

Deployment and activation

To make the functionality contained within a sandboxed solution available to users, an administrator must:

1. *Upload the solution to the Solutions gallery for the site collection.* Site collection administrators can add a solution to the gallery through the SharePoint site user interface in the browser. Farm administrators can add solutions by using the **Add-SPUserSolution** cmdlet in Windows PowerShell.
2. *Activate the solution.* Site collection administrators can activate sandboxed solutions from the Solutions gallery in the site user interface. Farm administrators can activate sandboxed solutions by using the **Install-SPUserSolution** cmdlet in Windows PowerShell.

- Deployment and activation
 - Farm administrators: Windows PowerShell
 - Site collection administrators: SharePoint site UI
- Versioning and upgrading
 - Solution gallery stores multiple versions
 - Features are upgraded automatically

To remove a sandboxed solution, an administrator must:

1. *Deactivate the solution.* Site collection administrators can deactivate sandboxed solutions from the Solutions gallery in the site user interface. Farm administrators can deactivate sandboxed solutions by using the **Uninstall-SPUserSolution** cmdlet in Windows PowerShell.
2. *Remove the solution from the Solutions gallery.* Site collection administrators can remove a solution from the gallery through the site user interface. Farm administrators can remove a sandboxed solution by using the **Remove-SPUserSolution** cmdlet in Windows PowerShell.

You can access the Solutions gallery from the Site Settings page on any SharePoint site. In addition to providing functionality for managing the solution life cycle, the Solutions gallery also provides information about resource quotas and the number of resource points consumed by sandboxed solutions within the site collection.

Versioning and upgrading

In many ways, upgrading a sandboxed solution is a similar process to upgrading a farm solution. You must increment the version number in the solution manifest, and you must ensure that the solution ID remains unchanged. The differences between upgrading sandboxed solutions and farm solution lie in how the upgrade is handled by SharePoint.

When you upgrade a farm solution, the existing version of the solution in the solution store is replaced by the new version. The solution store does not retain old versions of farm solutions. By contrast, the Solutions gallery does retain old versions of sandboxed solutions. When you create an updated sandboxed solution, you must rename the published file to differ from the original solution name. You can choose any name for the update solution; the solution ID is used to determine if the new solution is an upgrade to an existing solution. A simple approach to managing solution versions is to include the solution number in the file name. For example, if you want to upgrade a sandboxed solution named *ContosoInvoicing.wsp*, you might name the upgraded version *ContosoInvoicing_2.0.0.0.wsp*. Using this kind of naming convention makes it easier for administrators to identify upgrades.

Another significant deviation from farm solutions is the Feature upgrade process. With Farm solutions, if an upgrade solution contains upgraded versions of Features, those Features are not automatically upgraded. You can use Windows PowerShell or custom code to perform the update on a per instance basis. This enables you to stage the update of the Feature across sites, site collections, web applications, and the farm. With sandboxed solutions, this is not possible. When you install an updated solution that contains updated Features, all deployed instances of that Feature within the site collection are automatically upgraded.

To update the version of the solution, site collection administrators upload the new version of the solution package to the solution gallery in their site collection, and then make sure the solution is activated.

Creating and Installing Solution Validators

In addition to providing monitoring and throttling functionality for sandboxed solutions, SharePoint provides a solution validation framework that you can leverage to provide additional control over what sandboxed solutions can be deployed. To use this validation framework, you must deploy a custom *solution validator* to the SharePoint farm. Solution validators run whenever a site collection administrator attempts to install a sandboxed solution. Within the solution validator, you can allow or block the installation of the solution based on:

- Creating solution validators
 - Inherit from **SPSolutionValidator**
 - Override **ValidateSolution**
 - Override **ValidateAssembly**
- Installing solution validators
 - Deploy validator assembly to the global assembly cache
 - Add the validator to the **SolutionValidators** collection of the user code service
 - Typically implemented in a Feature receiver class

- The properties and contents of the solution package.
- The properties of the assembly or assemblies within the solution package.

You can deploy multiple solution validators to a SharePoint farm. Every solution validator is executed whenever a sandboxed solution is installed.

Creating solution validators

To create a solution validator, you must create a class that inherits from the **SPSolutionValidator** abstract base class. You must then override one or both of the following methods:

- *ValidateSolution*. This method is called once for each solution package.
- *ValidateAssembly*. This method is called once for every assembly in the solution package.

The following code example shows a simple solution validator class:

Creating a Solution Validator

```
[Guid("A8CDBF14-5914-40D0-A213-2320EE08B386")]
public class ContosoSolutionValidator : SPSolutionValidator
{
    // Create a unique string value to identify the solution validator.
    const string validatorId = "ContosoValidator1";

    // Include a public default constructor to support serialization.
    public ContosoSolutionValidator() { }

    // Include a constructor that accepts an SPUserCodeService instance.
    // Within this constructor, set the Signature property to an integer value.
    // The Signature property identifies the version of your validator.
    public ContosoSolutionValidator(SPUserCodeService service)
        : base(validatorId, service)
    {
        Signature = 1;
    }

    public override void ValidateSolution(SPSolutionValidationProperties properties)
    {
        // Block the solution if the solution filename includes the text "Northwind".
        if (properties.Name.ToLower().Contains("northwind"))
        {
            properties.Valid = false;
            properties.ValidationErrorMessage = "Solutions from Northwind Traders are not
permitted!";
        }
    }
}
```

```
}

public override void ValidateAssembly(SPSolutionValidationProperties properties,
    SPSolutionFile assembly)
{
    // Block the solution if any assembly names include the text "Northwind".
    if (assembly.Location.ToLower().Contains("northwind"))
    {
        properties.Valid = false;
        properties.ValidationErrorMessage = "Assemblies from Northwind Traders are not
permitted!";
    }
}
}
```

The preceding code is a fairly contrived example of a solution validator. In practice, the capabilities of solution validators are unlimited. If you want, you can retrieve and examine the contents of every file within the solution package before validating or blocking the solution installation.

Installing solution validators

Solution validator assemblies must be deployed to the global assembly cache. To make your solution validator active, you must register it with the SharePoint farm. To do this, you add your validator to the **SolutionValidators** collection of the user code service. The best way to do this is to use a Feature receiver class. You can add your solution validator in the **FeatureActivated** method and remove it in the **FeatureDeactivating** method.

The following code example shows a Feature receiver class that installs a solution validator:

Installing a Solution Validator

```
public class SolutionValidatorFeatureReceiver : SPFeatureReceiver
{
    public override void FeatureActivated(SPFeatureReceiverProperties properties)
    {
        SPUserCodeService service = SPUserCodeService.Local;
        ContosoSolutionValidator validator = new ContosoSolutionValidator(service);
        service.SolutionValidators.Add(validator);
    }

    public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
    {
        SPUserCodeService service = SPUserCodeService.Local;
        Guid validatorID = service.SolutionValidators["ContosoValidator1"].Id;
        service.SolutionValidators.Remove(validatorID);
    }
}
```

Discussion: When Are Sandboxed Solutions Appropriate?

The instructor will now lead a brief discussion around the following questions:

- When would you create a sandboxed solution instead of a farm solution?
- When would you create a sandboxed solution instead of an app for SharePoint?

- When would you create a sandboxed solution instead of a farm solution?
- When would you create a sandboxed solution instead of an app for SharePoint?

Lab: Working With Features and Solutions

Scenario

Contoso increasingly relies on the assistance of contractors in various business divisions. Each contractor must sign a contract that specifies the terms of engagement and includes a non-disclosure agreement. The management team at Contoso wants a way to centrally manage these agreements.

Your requirements analysis shows that you require some custom site columns, a content type, and a custom list template. You will use a Site-scoped Feature to deploy the site columns and the content type. You will create the site columns declaratively, and you will use a Feature receiver class to create the content type programmatically. Creating content types programmatically provides more flexibility if you need to update the content type at a later date. You will then create a Web-scoped Feature to provision the list. Because the list relies on the site columns and the content type, the Web-scoped Feature must include a dependency on the Site-scoped Feature.

Objectives

After completing this lab, you will be able to:

- Configure SharePoint Features.
- Create Feature receiver classes and create content types programmatically.
- Configure dependencies between Features.

Estimated Time: 60 minutes

Virtual machine: 20488B-LON-SP-04

Exercise 1: Configuring SharePoint Features

Scenario

In this exercise, you will create a new SharePoint project in Visual Studio and add several site columns. You will then configure a Site-scoped feature to deploy and provision the custom site columns.

The main tasks for this exercise are as follows:

1. Create a new SharePoint project in Visual Studio
2. Add site columns to the solution
3. Create and configure a Feature

► Task 1: Create a new SharePoint project in Visual Studio

- Start the 20488B-LON-SP-04 virtual machine.
- Log on to the **LONDON** machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
- Open Visual Studio 2012 and create a new empty SharePoint 2013 project.
- Set the name of the project to **ContractorAgreements**.
- Create the project in the **E:\Labfiles\Starter** folder.
- Use the site at **http://team.contoso.com** for debugging.
- Configure the project to deploy the solution as a farm solution.



Note: In a production environment, you would deploy these components in a sandboxed solution. However, SharePoint 2013 will not allow you to execute sandboxed code on a domain controller. As such, you must use a farm solution in the classroom environment.

► **Task 2: Add site columns to the solution**

- Add a new site column named **ContosoManager** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Contoso Manager**.
 - b. Set the Type of the column to **User**.
 - c. Configure the column to accept only users, not users and groups.
 - d. Make the column compulsory.
 - e. Set the Group of the column to **Contoso Site Columns**.
- Add a new site column named **ContosoTeam** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Contoso Team**.
 - b. Set the Type of the column to **Choice**.
 - c. Make the column compulsory.
 - d. Set the Group of the column to **Contoso Site Columns**.
 - e. Add the following choices to the column:
 - i. Research
 - ii. Production
 - iii. Audit
 - iv. IT
- Add a new site column named **DailyRate** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Daily Rate**.
 - b. Set the Type of the column to **Currency**.
 - c. Set the LCID of the column to **1033**.
 - d. Set the number of decimal places the column displays to **0**.
 - e. Make the column compulsory.
 - f. Set the Group of the column to **Contoso Site Columns**.
- Add a new site column named **AgreementStartDate** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Agreement Start Date**.
 - b. Set the Type of the column to **DateTime**.
 - c. Set the Format of the column to **DateOnly**.
 - d. Make the column compulsory.
 - e. Set the Group of the column to **Contoso Site Columns**.
- Add a new site column named **AgreementEndate** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Agreement End Date**.
 - b. Set the Type of the column to **DateTime**.

- c. Set the Format of the column to **DateOnly**.
- d. Make the column compulsory.
- e. Set the Group of the column to **Contoso Site Columns**.
- Add a new site column named **SecurityCleared** to the project. Configure the column properties as follows:
 - a. Set the Display Name of the column to **Security Cleared**.
 - b. Set the Type of the column to **Boolean**.
 - c. Make the column compulsory.
 - d. Set the Group of the column to **Contoso Site Columns**.

► Task 3: Create and configure a Feature

- Delete the existing Feature named **Feature1** from the project.



Note: Visual Studio creates a Feature automatically when you add declarative items to a SharePoint project. However, in this exercise you will create a new Feature to help you to gain an understanding of the complete process.

- Add a new Feature named **ContractingResources** to the project.
- Set the title of the Feature to **Contractor Management Resources**.
- Set the Feature description to **Provisions site columns and content types for managing contractor agreements**.
- Set the scope of the Feature to **Site** (site collection).
- Save your changes, and then review the Feature manifest file that Visual Studio has generated for you.

Results: After completing this exercise, you should have configured a Site-scoped Feature to deploy several site columns.

Exercise 2: Creating Feature Receiver Classes

Scenario

In this exercise, you will add a Feature receiver class to the Feature you created in the previous exercise. In the **FeatureActivated** method, you will create a new content type and add it to the root web of the site collection. In the **FeatureDeactivating** method, you will reverse your changes and remove the content type.

The main tasks for this exercise are as follows:

1. Create a Feature receiver class
2. Create a content type programmatically
3. Remove a content type programmatically
4. Test the Feature receiver

► Task 1: Create a Feature receiver class

- Add an event receiver to the **ContractingResources** Feature.
- Within the event receiver class, declare a static read-only field of type **SPContentTypeId** named **ctid**.
- Initialize the **ctid** variable as a new **SPContentTypeId** instance by passing the following string value as an argument to the constructor:

```
0x010100C3316E15A95F420F8187FBBE1B9636F9
```



Note: You are creating a static content type ID to make it easy to identify and remove your content type at a later date. You will learn more about content type IDs later in this course.

► Task 2: Create a content type programmatically


- In the event receiver class, uncomment the **FeatureActivated** method.
- Retrieve the root web of the site collection in which the Feature is being activated.
- Create a new **SPContentType** instance named **contractingCT**.
- Check whether the content type already exists. To do this, attempt to retrieve the content type from the **ContentTypes** collection of the root web. Use the **ctid** field to specify the content type ID of the content type you want to retrieve. Set the **contractingCT** variable to the content type you retrieve from the root web.
- If the **contractingCT** variable is equal to null (in other words, if the content type does not already exist in the root web), set the **contractingCT** variable to a new instance of **SPContentType** by passing the following values to the constructor:
 - a. Use the **ctid** variable to specify the content type ID.
 - b. Specify the **ContentTypes** collection of the root web.
 - c. Set the name of the content type to **Contractor Agreement**.
- Set the description of the content type to **A contractual agreement between Contoso Pharmaceuticals and an independent contractor**.
- Set the content type group to **Contoso Content Types**.
- Add the following fields to the content type:
 - a. Full Name
 - b. Contoso Manager
 - c. Contoso Team
 - d. Daily Rate
 - e. Agreement Start Date
 - f. Agreement End Date
 - g. Security Cleared
- Call the **Update** method on the content type to persist your changes to the content database. Make sure the changes are persisted to inherited content types and list content types.

► **Task 3: Remove a content type programmatically**


- In the event receiver class, uncomment the **FeatureDeactivating** method.
- Retrieve the root web of the site collection in which the Feature is being deactivated.
- Create a new **SPContentType** instance named **contractingCT**.
- Check whether the content exists in the **ContentTypes** collection of the root web. To do this, attempt to retrieve the content type from the **ContentTypes** collection of the root web. Use the **ctid** field to specify the content type ID of the content type you want to retrieve. Set the **contractingCT** variable to the content type you retrieve from the root web.
- If the **contractingCT** variable is not null (in other words, if the content type exists on the root web), call the **Delete** method on the content type.

► **Task 4: Test the Feature receiver**

- Build the solution and start without debugging.
- Verify that the SharePoint site includes a site content type named **Contractor Agreement**.
- Verify that the Contractor Agreement content type includes the following columns:
 - a. Full Name
 - b. Contoso Manager
 - c. Contoso Team
 - d. Daily Rate
 - e. Agreement Start Date
 - f. Agreement End Date
 - g. Security Cleared
- In the site collection settings, deactivate the **Contractor Management Resources** feature.
- Verify that the Content Types gallery no longer includes the Contractor Agreement content type.
- Reactivate the **Contractor Management Resources** site collection feature.

 **Note:** You will want the Feature to be active so that Visual Studio can retrieve the custom site columns from the site in the next exercise.

- Close Internet Explorer and close Visual Studio.

 **Note:** You will reopen Visual Studio in the next task. Closing and reopening Visual Studio forces Visual Studio to retrieve the current list of content types from the SharePoint site.

Results: After completing this exercise, you should have created and tested a Feature receiver class that creates a new content type.

Exercise 3: Creating Features with Dependencies

Scenario

In this exercise, you will create a Web-scoped Feature that provisions a list template and a list instance. You will configure the Feature to be dependent on the Site-scoped Feature that you created in the first exercise.

The main tasks for this exercise are as follows:

1. Add a list template and a list instance to the solution
2. Configure a Feature with dependencies
3. Test the Feature dependency

► Task 1: Add a list template and a list instance to the solution

- Open Visual Studio and reopen the **ContractorAgreements** project.
- Add a new list named **ContractorAgreementsList** to the project.
- Set the display name for the list to **Contractor Agreements**.
- Use the customizable list template option and use **Document Library** as the base type for the list.
- Remove all the existing content types from the list, and add the **Contractor Agreement** content type to the list.
- Set the list description to **Use this list to manage contracts between Contoso Pharmaceuticals and external contractors**.

► Task 2: Configure a Feature with dependencies

- Add a new Feature named **ContractorAgreementsList** to the project.
- Set the title of the Feature to **Contractor Agreements List**.
- Set the description of the **Feature to Provisions a Contractor Agreements list at the Web scope**.
- Set the scope of the Feature to **Web**.
- Add the **ContractorAgreementsListInstance** and **ContractorAgreementsList** items to the Feature.
- Add a dependency on the **Contractor Management Resources** Feature.

► Task 3: Test the Feature dependency

- Change the active deployment configuration of the project to **No Activation**.
- Rebuild the solution, and then start without debugging.
- On the Site Settings page for the SharePoint site, attempt to activate the **Contractor Agreements List** Feature.
- Verify that SharePoint displays an error message stating that you must activate the Contractor Management Resources feature before trying again.
- In the site collection settings, activate the **Contractor Management Resources** Feature.
- In the site settings, activate the **Contractor Agreements List** Feature.
- Browse to the **Contractor Agreements** list.
- Add the file at **E:\Labfiles\Starter**, click **SJAgreement.docx** to the list.

- Set the properties of the document as follows:

Property	Value
Title	SJ Work for Hire Agreement
Full Name	Sanjay Jacob
Contoso Manager	Carol Troup
Contoso Team	Research
Daily Rate	\$800
Agreement Start Date	10/1/2013
Agreement End Date	10/1/2014
Security Cleared	Yes

- Save your changes, and then close all open windows.

Results: After completing this exercise, you should have configured a Web-scoped Feature that includes a dependency on a Site-scoped Feature.

Module Review and Takeaways

In this module, you learned about how you can use Features and solutions to manage the life cycle of your custom SharePoint components. You learned about the core concepts of Features and solutions, and you learned how to configure and manage Features and solutions for your own custom components. You also learned about the sandboxed solution framework, including key capabilities and constraints and when sandboxed solutions are the best deployment choice.

Review Question(s)

Test Your Knowledge

Question	
<p>You have created a Feature that provisions a content type. You want the content type to be available on every site collection on the farm, if the site collection administrator wants to use it. What value should you use for the Feature scope?</p>	
<p>Select the correct answer.</p>	
<input type="checkbox"/>	Web
<input type="checkbox"/>	Site
<input type="checkbox"/>	WebApplication
<input type="checkbox"/>	Farm

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
<p>True or false: A Site-scoped Feature can include a dependency on a Web-scoped Feature.</p>	

Test Your Knowledge

Question	
<p>Which of the following items cannot be deployed within a sandboxed solution?</p>	
<p>Select the correct answer.</p>	
<input type="checkbox"/>	A Web Part.
<input type="checkbox"/>	A custom list.
<input type="checkbox"/>	A list item event receiver.
<input type="checkbox"/>	A timer job.
<input type="checkbox"/>	A site column.

Module 5

Working with Server-Side Code

Contents:

Module Overview	5-1
Lesson 1: Developing Web Parts	5-2
Lesson 2: Using Event Receivers	5-7
Lesson 3: Using Timer Jobs	5-13
Lesson 4: Storing Configuration Data	5-21
Lab: Working with Server-Side Code	5-28
Module Review and Takeaways	5-35

Module Overview

When you develop SharePoint solutions, you will often need to develop code that runs on the server. This can range from code that runs in the IIS worker process, for example a Web Part code file, or code that runs in a separate process, such as a timer job.

With server-side code, you can develop solutions that run during a page request, in response to an event, or according to a schedule. Developing server-side code provides you with maximum capability, but in some cases restricts your deployment options. For example, your options for deploying server-side code to SharePoint Online are very limited.

In this module, you will learn about some of the options for developing server-side components, when you might use them, and how you can deploy them. You will also look at how you can store configuration data for your custom components and the implications of various options.

Objectives

After completing this module, you will be able to:

- Describe the process for developing a Web Part.
- Use event receivers to handle SharePoint events.
- Use timer jobs to perform out-of-process and scheduled operations.
- Store and manipulate configuration data for custom components.

Lesson 1

Developing Web Parts

Web Parts were traditionally used to define the user interface for SharePoint solutions. In SharePoint 2013, you will often create an app rather than use a traditional Web Part. However, you can still choose to use a Web Part in your SharePoint solution, and they do have some advantages over alternatives. In this lesson you will learn more about Web Parts, including when you might use a Web Part, and how to develop a simple Web Part.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the purpose of Web Parts.
- Develop a simple Web Part.

Introduction to Web Parts

SharePoint enables you to create Web Parts that you can include on SharePoint pages. Web Parts provide flexibility to users who can add a Web Part to any page. You can make Web Parts that display data relevant to the current page, site, or other contextual component, or you can make Web Parts that display the same content wherever they are deployed.

In SharePoint 2013, much of what you can achieve by using a Web Part can also be achieved by using an app part. In most cases, you should choose to use an app part in preference to a Web Part; however, there are still some circumstances in which choosing a Web Part would be the appropriate choice.


If you do not have an app infrastructure configured for your SharePoint farm, you may choose to develop a Web Part instead of an app part. Another reason you may choose to develop a Web Part is to take advantage of Web Part connections. Web Parts can expose data to other Web Parts on the page, and Web Parts can then consume exposed data. You cannot consume or expose data to other Web Parts by using an app part, so if you need to use this functionality, developing a Web Part would be an appropriate choice.

To develop a Web Part, you normally create a class that inherits from the **WebPart** class (in the **System.Web.UI.WebControls.WebParts** namespace). You then override methods in that class to control how your Web Part is rendered. You then compile the assembly and deploy it to the server. You use a .webpart Web Part definition file to add the Web Part to a Web Part Gallery, from which users, with the appropriate permissions, can add the Web Part to a page.

- Do not require app infrastructure or any special services/service applications to run
- Can expose and consume data with other web parts
- Inherit from the WebPart class
- Deployed as assembly, with a .webpart definition file
- Use app part instead where possible

Understanding the Web Part Life Cycle

Web Parts have a life cycle that integrates with the page life cycle for a SharePoint page, or any ASP.NET webpage. When you develop a Web Part, you typically create a class that inherits from the **WebPart** class (although there are other options, they are not commonly used). In your class, you can override life cycle events. It is important that you understand the order in which life cycle events occur to enable you to develop an effective Web Part.



- OnInit
- OnLoad
- CreateChildControls
 - EnsureChildControls
- SaveViewState
- OnPreRender
- Page.PreRenderComplete
- Render
- RenderContents
- OnUnload

The Web Part life cycle events that you can override are

- *OnInit*. The **OnInit** method is used to initialize the Web Part. You should override this if you need to customize this process.
- *OnLoad*. The **OnLoad** method is also used to initialize the control, but it should not be used to load data.
- *CreateChildControls*. The **CreateChildControls** method is the most commonly overridden method in the Web Part life cycle. This method is responsible for creating controls that appear on the Web Part. You should use this method to add controls to the page's control collection.
- *EnsureChildControls*. The **EnsureChildControls** method ensures that **CreateChildControls** has executed. You do not normally need to override this method, but you should call this method before you attempt to access a control to prevent null reference exceptions from uninitialized controls. You should not call the **CreateChildControls** method directly because this may result in duplicate controls if the method has been called previously (for example, by SharePoint).
- *SaveViewState*. The **SaveViewState** method saves the view states for the Web Part and the controls on the Web Part.
- *OnPreRender*. The **OnPreRender** method handles tasks that must be completed before the page is rendered. For example, to load data in a data-bound control. You should override this if you need to customize this process.
- *Page.PreRenderComplete*. The **Page.PreRenderComplete** event fires after the **OnPreRender** method completes for all of the controls on the Web Part. You should handle this event if you need to customize any of the controls after they load data but before they are rendered.
- *Render*. The **Render** method renders the Web Part by writing HTML to the output stream.
- *RenderContents*. The **RenderContents** method renders the contents of the Web Part by writing to the HTML output stream. This is the second most commonly overridden method. You should override this method if you want to customize how your controls are rendered, or to write HTML directly to the output stream. When you use the **RenderContents** method, you do not output the outer HTML tags and styling, because these are output by the **Render** method.
- *OnUnload*. The **OnUnload** method runs after the request completes, and the rendered Web Part has been output to the HTML stream. You should override the **OnUnload** method if you created any objects that need explicit disposal, such as if you need to close a file or database connection.

If you are writing code to handle a page postback, you may need to handle a control postback event; for example, you may need to handle a button click event. These events are processed after the **CreateChildControls** method, and before the **OnPreRender** method. If you populate a control with data in the **CreateChildControls** method, it may overwrite changes that are included in the postback, and any

data that you process in your control postback method may be inaccurate. If you need to load data, you should load the data in the **OnPreRender** method. This ensures that you do not overwrite data included in a postback.

Visual Web Parts

Visual Web Parts are very similar to normal Web Parts; they inherit from the WebPart class and expose the same life cycle events. However, there are two major differences.

- You define the user interface for the Web Part in a custom control (ascx) file.
- You can choose to add logic to the **Page_Load** method in the control code-behind file to control the Web Part when the custom control loads; typically, you will continue to use the **OnPreRender** method to load data into your control.

- Inherit from the WebPart class
- Define interface in a custom control (ascx) file
- Expose an additional life cycle method *Page_Load*
- Available in both farm and sandboxed solutions

By creating a custom control to define the user interface, you benefit from using the Visual Studio page designer to enable you to easily add controls to the page and lay out the controls according to your needs. This is significantly easier than adding controls and defining layout manually by using the **CreateChildControls** method.

In previous versions of SharePoint, you could not use a Visual Web Part in a sandboxed solution because of the requirement to deploy the custom control file to the file system on the web servers. In SharePoint 2013, this requirement is removed, and Visual Web Parts are now compatible with both farm solutions and sandboxed solutions.

Demonstration: Creating a Visual Web Part

This demonstration uses a farm solution to deploy a Visual Web Part. You will see how to add controls to a Visual Web Part by using the designer, and how you can use breakpoints in Visual Studio to debug your Web Part.

Demonstration Steps

- Start the 20488B-LON-SP-05 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Desktop**.
- On the desktop, on the taskbar, click **File Explorer**.
- In File Explorer, browse to **E:\Democode\VisualWebPart**, and then double-click **VisualWebPart.sln**.
- If you are prompted to choose a program to open the file, click **Visual Studio 2012**.
- In Visual Studio, in Solution Explorer, right-click **VisualWebPart**, point to **Add**, and then click **New Item**.
- In the **Add New Item - VisualWebPart** dialog box, click **Visual Web Part**, and then click **Add**.
- In Solution Explorer, point out the two .cs code-behind files.

- Double-click **VisualWebPart1.ascx.cs**. Explain that students should add their logic to this class.
- In Solution Explorer, double-click **VisualWebPart1.ascx.g.cs**. Explain that this class contains auto-generated code, and is updated by Visual Studio. Make sure students are aware that they should not edit this file. Point out that this is a partial class and is merged with the other code file when the class is compiled.
- On the **VisualWebPart1.ascx** tab, click **Design**.
- From the **Toolbox**, drag a **Button** to the design canvas.
- In **Properties**, in the **Text** box, type **Click Me**.
- On the design canvas, double-click the button.
- Point out that the event handler is added to the code-behind file, similar to developing an ASP.NET web application.
- Right-click the line that contains the opening brace for the **Button1_Click** method, point to **Breakpoint**, and then click **Insert Breakpoint**.
- On the **DEBUG** menu, click **Start Debugging**.
- In the **Debugging Not Enabled** dialog box, click **OK**.
- In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
- If a **Microsoft Visual Studio** dialog box appears, click **Yes**.
- In Internet Explorer, on the **Contoso Development Site**, on the ribbon, on the **PAGE** tab, click **Edit**.
- On the ribbon, on the **INSERT** tab, click **Web Part**.
- In the **Categories** list, click **Custom**.
- Under **Parts**, click **VisualWebPart - VisualWebPart1**, and then click **Add**.
- On the ribbon, click **SAVE**.
- In the **VisualWebPart - VisualWebPart1** Web Part, click **Click Me**.
- In Visual Studio, point out that the breakpoint has been hit, and then click **Continue**.
- Close Internet Explorer.
- Close Visual Studio.

The .webpart File

Typically, you will deploy a Web Part as a pre-compiled assembly, either as part of a farm solution, or a sandboxed solution. You must create a XML Web Part definition file that defines the assembly and type that contain your Web Part code, as well as metadata such as the name, description, and group of the Web Part. When you create a Web Part by using Visual Studio, Visual Studio generates this file, and includes it in a Feature automatically; however, it is important that you are aware of the structure and purpose of the Web Part definition file in case you need to

- XML file containing a Web Part definition
- Stored in Web Part Gallery
- Identifies the assembly and type that contains the Web Part
- Contains Web Part metadata, such as the Web Part name, description, and display group

customize it.

When you add a Web Part to SharePoint, SharePoint stores the Web Part in a Web Part Gallery. When a user with the necessary permission edits a page, they can add Web Parts that appear in the Web Part Gallery. The Web Part Gallery actually contains .webpart files. When a Web Part is added to the page, SharePoint parses the .webpart file and loads the Web Part from the assembly identified in the file.

The following code example shows a .webpart file.

Web Part Definition File Code Example

```
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <!-- Add the fully qualified type name and strong named assembly name -->
      <type name="ContosoWebPartNamespace.ContosoWebPart, WebPartAssembly Strong Name">
      <!-- Add an error message to display if the Web Part does not import correctly -->
      <importErrorMessage>Error importing Web Part.</importErrorMessage>
    </metadata>
    <data>
      <properties>
        <!-- Add properties that define the Web Part metadata -->
        <property name="Title" type="string">Contoso Web Part</property>
        <property name="Description" type="string">This is the Contoso Web
Part</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

Lesson 2

Using Event Receivers

SharePoint solutions, like any other solution, need to respond to external events. In SharePoint, you can use event receivers to respond to events generated external to your customization. In this lesson, you will learn about event receivers, what event you can handle by using an event receiver, and how to develop and deploy an event receiver.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the purpose of an event receiver.
- Develop an event receiver.
- Deploy an event receiver.

Introduction to Event Receivers

When you develop any solution, you will often need to respond to actions performed outside the scope of your solution. In traditional Windows Forms application development, a common example is responding to a user clicking a button. Developing solutions for SharePoint is no different; you will often want to respond to actions performed outside the scope of your solution.

Event receivers in SharePoint enable you to respond to these actions. For example, you can develop an event receiver that SharePoint will call each time an item is added to a list. SharePoint enables you to respond to a variety of events raised by SharePoint components, including:

- Site collections
- Sites
- Lists
- Content types
- Features

The events that you can handle for each of these components, referred to as event hosts, differ—generally, they include an event when it is changed or deleted, and when child items are added, changed, or deleted. The event hosts are hierarchical, and provide scope for your event receivers. For example, you can handle an item being added by using a list as an event host. This will restrict your event handler to run only when an item is added to that particular list. Alternatively, you can register the same event receiver by using the site collection as the even host, causing the event receiver to run every time an item is added to any list in the site collection.

- Handle actions performed outside the scope of your solution
- Event hosts
 - SPSite
 - SPWeb
 - SPList
 - SPContentType
- Event synchronization
 - Before events (ing)
 - After events (ed)



Additional Reading: For more information about events you can handle in SharePoint, see *Using Event Receivers in SharePoint Foundation (Part 1 of 2)* at <http://go.microsoft.com/fwlink/?LinkID=306990>.



Note: Feature receivers are a type of event receiver; however, they are not covered in this module. For more information about Feature receivers, see the *Designing and Managing Features and Solutions* module of this course.

Event handler synchronization

An event may be handled either synchronously or asynchronously. This has important implications for server performance, and functional implications. Synchronous events run in the same thread as the code that raised the event. Frequently, this has a direct impact on SharePoint performance for users, for example, if the event was raised by a user browsing a SharePoint site, a synchronous event will run in the IIS process thread. A major advantage of a synchronous event is that it can change an event, for example by cancelling the event. If you use a synchronous event, the event process will always complete before the web request completes.

Asynchronous events run in a different thread to the main IIS process thread, and can run in parallel on multiple other threads. This has the advantage that they do not block the IIS process thread, and the request returns to the browser as fast as possible. The event handler may continue to run on the server after the web request that triggered the event completes. A significant implication of this is that asynchronous events should not change the event, because the event may have already occurred.

Before and after events

SharePoint exposes two broad types of events, before events and after events. Before events are events that end in “**ing**”; for example, **ItemAdding**. These events are triggered before an action is committed to the database and provide the opportunity for you to cancel the event, or run other necessary code before the change is persisted. After events are events that end in “**ed**”; for example, **ItemAdded**. These events are triggered after an action is committed to the database. You cannot cancel an event in an after event (although you could use code to reverse the change).

Before events always run synchronously, this is necessary in case the event receiver makes changes to the event, such as by blocking the event. After events typically run asynchronously; although this is normally appropriate, you can choose to run an after event synchronously according to your requirements. You specify the event synchronization when you deploy your event receiver.

Remote event receivers

In addition to deploying event receivers on your SharePoint server, you can also use remote event receivers. You can create a remote event receiver by specifying a URL for a remote event handler, or by using an app. More information about how to handle an event by using a remote event receiver in an app is included later in this course.

Developing an Event Receiver

To develop an event receiver, you simply create a type that derives from a base class, and then override an event method. There are different base classes for event receivers including:

- *SPWebEventReceiver*. Use the **SPWebEventReceiver** base class to handle site and site collection life cycle events.
- *SPListEventReceiver*. Use the **SPListEventReceiver** base class to handle list life cycle events, which include events such as a field being added to a list, in addition to a list being created or deleted.
- *SPItemEventReceiver*. Use the **SPItemEventReceiver** base class to handle item life cycle events.
- *SPEmailEventReceiver*. Use the **SPEmailEventReceiver** base class to handle an email message being received by the event host.
- *SPWorkflowEventReceiver*. Use the **SPWorkflowEventReceiver** base class to handle workflow life cycle events.
- *SPSecurityEventReceiver*. Use the **SPSecurityEventReceiver** base class to handle security events, for example a new group being created, or a user being added to a group.

- Inherit from base class
 - SPWebEventReceiver
 - SPListEventReceiver
 - SPItemEventReceiver
 - SPEmailEventReceiver
 - SPWorkflowEventReceiver
 - SPSecurityEventReceiver
- Override methods
 - Cancel before events
 - Without an error message
 - With an error message
 - With a page redirect

You must select the appropriate base class based on the event you need to handle. After you select the appropriate base class, you must override base class methods with your event handler implementation.

The following code example shows an example of how to handle an item being added to a list.

ItemAdding Code Sample

```
public class EventReceiver : SPItemEventReceiver
{
    public override void ItemAdding(SPItemEventProperties properties)
    {
        base.ItemAdding(properties);
    }
}
```

Cancelling an event

When you develop an event receiver for a before event, you may use the event receiver to cancel the event. Most event handler methods accept a **properties** parameter. You should use the **Status** property of the **properties** parameter to cancel the event by setting the **Status** property to a member of the **SPEventReceiverStatus** enumeration. You can choose to cancel an event with or without displaying an error. If you choose to display an error, you should also provide either an error message by setting the **ErrorMessage** property, or the URL of an error page that the user should be redirected to, by setting the **RedirectUrl** property.

The following code example shows how to cancel an event.

Cancelling the ItemAdding Event Code Sample

```
public override void ItemAdding(SPIItemEventProperties properties)
{
    // Cancel an event and do not display an error.
    properties.Status = SPEventReceiverStatus.CancelNoError;

    // Cancel an event and display an error message.
    properties.Status = SPEventReceiverStatus.CancelWithError;
    properties.ErrorMessage = "Event cancelled.";

    // Cancel an event and redirect the user to an error page.
    properties.Status = SPEventReceiverStatus.CancelWithRedirectUrl;
    properties.RedirectUrl = "CustomErrorPage.aspx";
}
```



Note: You can only cancel an event in a before event (events with names that end with "ing").

Deploying an Event Receiver

You can choose to deploy an event receiver manually, by installing the event receiver assembly in the global assembly cache, and then you can use Windows PowerShell to associate, or bind, the event receiver with an event source. However, if you deploy manually, you must deploy the assembly to every server, and the process includes inherent risks (for example, a typographical error). Alternatively, you can use a solution and a Feature to deploy an event receiver. When you use a Feature to deploy an event receiver, you can choose one of two options; you can either bind the event receiver with an event source declaratively, or bind the event receiver with an event source by using code. Whichever approach you choose, you should package the assembly containing the compiled event receiver in the solution.

- Deploy manually (bad practice)
- Deploy with a solution and Feature (best practice)
- Bind event receiver to an event host with code
- Bind event receiver to an event host declaratively
- Common properties
 - Assembly
 - Class
 - Type
 - Name (optional)
 - Synchronization (optional)
 - SequenceNumber (optional)

Regardless of your approach, you will need to provide the same information:

- *Assembly*. You must use the strong name for your handler assembly.
- *Class*. You must use the fully-qualified type name for your event receiver class.
- *Type*. You must specify a value from the **SPEventReceiverType** enumeration that identifies the event your event receiver should handle.
- *Url*. As an alternative to specifying an assembly, class, and type, you can specify a Url of a remote event receiver. If you specify a Url, you do not need to specify an assembly, class, or type; in all other circumstances, these are required.
- *Name*. You can specify a name for your event receiver binding.
- *Synchronization*. You can specify the synchronization for your event receiver. If you omit this property, the default synchronization is used, based on the type of event.

- *SequenceNumber*. You can specify a sequence number for your event receiver. SharePoint uses the sequence number to determine the order to run event receivers. The default value is 10000, and lower numbers are run before high numbers.

Using code to deploy an event receiver

To bind your event receiver to an event host programmatically, you must first obtain a reference to the event host. You then create a new instance of the **SPEventReceiverDefinition** class by adding a new item to the event host's **EventReceivers** collection property. You then set the properties of the **SPEventReceiverDefinition** object, before finally calling the **Update** method to persist the changes.

The following code example shows how to bind an event receiver to an event host programmatically.

Programmatic Event Binding Code Example

```
using (SPSite site = new SPSite("http://portal.contoso.com"))
{
    using (SPWeb web = site.OpenWeb())
    {
        // Obtain a reference to the event host.
        SPList list = web.Lists["Suppliers"];

        // Create a new instance of the SPEventReceiverDefinition class by using the Add method
of the
        // EventReceivers collection of the event host.
        SPEventReceiverDefinition eventRec = list.EventReceivers.Add();

        // Optionally, specify a name for the event receiver binding.
        eventRec.Name = "Event Receiver For ItemAdded";

        // Specify the strong name for the assembly that contains the compiled event receiver.
        eventRec.Assembly =
            "ReceiverAssembly, Version=1.0.0.0, Culture=Neutral, PublicKeyToken=24a5ef6a3fe2c28c";

        // Specify the fully qualified type name for your event receiver class.
        eventRec.Class = "ReceiverNamespace.ReceiverClass";

        // Specify the event that should be handled.
        eventRec.Type = SPEventReceiverType.ItemAdded;

        // Specify whether the event should run synchronously or asynchronously.
        eventRec.Synchronization = SPEventReceiverSynchronization.Synchronous;

        // Optionally, specify a sequence value for the event receiver.
        eventRec.SequenceNumber = 50;

        // Call the Update method to persist the changes.
        eventRec.Update();
    }
}
```

Using declarative markup to deploy an event receiver

You can use declarative markup to associate an event receiver with an event source. To use declarative markup, you add a **Receivers** element to the **Elements** element in an element manifest file. When you use declarative deployment, SharePoint will bind the event receiver to the event host based on the Feature activation scope, for example by binding it to a site if the Feature has the **Web** scope. If you are binding to a list, you can identify the list by using the attributes of the **Receivers** element.

The following code example shows how to bind an event receiver to an event host declaratively.

Declarative Event Binding Code Example

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- Add one Receivers element per event host. -->
  <!-- Set the attributes on the Receivers element to identify the event host. -->
  <Receivers ListUrl="http://portal.contoso.com/lists/suppliers">

    <!-- Add one Receiver element per event you want to handle on this event host. -->
    <Receiver>

      <!-- Optionally, specify a name for the event receiver binding. -->
      <Name>Event Receiver For ItemAdded</Name>

      <!-- Specify the strong name for the assembly that contains the compiled event
receiver. -->
      <Assembly>
        ReceiverAssembly, Version=1.0.0.0, Culture=Neutral,
        PublicKeyToken=24a5ef6a3fe2c28c
      </Assembly>

      <!-- Specify the fully qualified type name for your event receiver class. -->
      <Class>ReceiverNamespace.ReceiverClass</Class>

      <!-- Specify the event that should be handled. -->
      <Type>ItemAdded</Type>

      <!-- Specify whether the event should run synchronously or asynchronously. -->
      <Synchronization>Synchronous</Synchronization>

      <!-- Optionally, specify a sequence value for the event receiver. -->
      <SequenceNumber>50</SequenceNumber>
    </Receiver>
  </Receivers>
</Elements>
```

Discussion Question

The instructor will now lead a discussion around the question: When would you use an event receiver?

• When would you use an event receiver?

Lesson 3

Using Timer Jobs

Some processes take too long to complete to include in a page request; you may need to run other processes on a regularly scheduled basis. Timer jobs enable you to run code in a different process to the web server process, either on a scheduled or one-off basis. In this lesson you will learn about timer jobs, the different types of timer jobs, and how to work with timer jobs.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how timer jobs work.
- Explain how to develop a timer job.
- Deploy a timer job.
- Explain work item timer jobs.
- Develop a work item timer job.

Introduction to Timer Jobs

Some solutions require you to perform processes, which are either too time consuming or too computationally expensive to run each time a page is requested. SharePoint enables you to develop timer jobs that run in a separate process, and run on a scheduled basis. Timer jobs enable you to develop code that runs either once, or regularly, by using a schedule.

When you define a schedule, you can define a time window for the timer job. If you use a time window, you can reduce the instantaneous load on your server farm because SharePoint can run the timer job at different times within the time window on different servers in the farm. This enables some servers to remain responsive while others are running the timer job.

Timer jobs do not run in the IIS worker process; instead, they run in the OWSTIMER.exe process. This means they are not subject to the same timeouts that may apply to web requests in the IIS worker process, and can perform long running calculations and processes. Also, by default, the OWSTIMER.exe process runs under the farm service account. This provides access for code running in a timer job to both the content and configuration databases. Furthermore, whereas by default, the ASP .NET code runs by using an impersonated identity of the current user, which is likely to further restrict your code's access to SharePoint and the server, the timer job uses the process identity to run code, which provides you with the most complete access to SharePoint and the server.

- Long-running processes
- Processor-intensive processes
- One time or regular schedule
- Reduce instantaneous load on server farm
- OWSTIMER.exe

Developing a Timer Job

To develop a timer job definition, complete the following steps:

1. Create a new class that inherits from the **SPJobDefinition** class (in the **Microsoft.SharePoint.Administration** namespace).
2. Add a default constructor to the class.
3. Add a non-default constructor that accepts four parameters:
 - A **string** representing the name of the timer job.
 - A **SPWebApplication** representing the web application that owns the timer job.
 - A **SPServer** representing the server with which this job is associated.
 - A **SPLockType** identifying the lock type that should be used for this job (this enables you to specify whether the job can be run on more than one server at a time, and whether this needs to run for each content database, or just once).
4. Modify the non-default constructor to call the equivalent constructor on the base class.
5. Override the **Execute** method. You should add your custom logic to this method.

- Create a class that derives from **SPJobDefinition**
- Add a default constructor
- Add a non-default constructor:
 - Name (**string**)
 - Web application (**SPWebApplication**)
 - Server (**SPServer**)
 - Lock type (**SPJobLockType**)
- Override the **Execute** method

The following code example shows how to develop a custom timer job.

Custom Timer Job Code Example

```
// Define a class that inherits from the SPJobDefinition class.
public class ContosoTimerJob : SPJobDefinition
{
    // Define a default constructor.
    public ContosoTimerJob()
    {
    }

    // Define a non-default constructor
    public ContosoTimerJob(string name, SPWebApplication webApplication,
        SPServer server, SPJobLockType lockType)
        : base(name, webApplication, server, lockType) // Call the constructor of the base class.
    {
    }

    public override void Execute(Guid targetInstanceId)
    {
        // Add custom logic here.
    }
}
```

Deploying a Timer Job

To deploy a timer job, you must use code. The most common technique to deploy a timer job is to use a Feature with a Feature receiver. You should use the **FeatureActivated** method to install your timer job, and the **FeatureDeactivated** method to uninstall your timer job. You must provide a name for your timer job. You can define a string constant in your Feature receiver class to store the timer job name. You should always check to ensure that a timer job with the same name does not already exist before you install your timer job. If a timer job with the same name already exists, you should uninstall that timer job first.

To uninstall a timer job, you complete the following steps:

1. Obtain a reference to the web application with which the timer job is associated. You can obtain this by using the **properties** parameter passed to the **FeatureDeactivated** method.
2. Iterate over the **JobDefinitions** collection property exposed by the web application object. For each job in the collection, check whether the **Name** property matches your timer job's name. If it does, call the **Delete** method on the job.

When you deploy a schedule, you must create a schedule for the job. The schedule determines how often the job should run, and when the job should run. You create a schedule by creating an instance of one of the following classes:

- **SPMinuteSchedule**
- **SPHourlySchedule**
- **SPDailySchedule**
- **SPWeeklySchedule**
- **SPMonthlySchedule**
- **SPYearlySchedule**

When you create an instance of one of these classes, you define a begin time, an end time, and an interval. The begin time specifies the earliest that the timer job can start running. The end time specifies the latest that the timer job can start running. For example, with a **SPMinuteSchedule**, you can specify the **BeginSecond** and **EndSecond** properties. If you specify a **BeginSecond** of **1** and an **EndSecond** of **5**, the timer job will start running between 16:00:01, and 16:00:05 (assuming that the job was due to run at some point during the minute at 16:00). A timer job can continue to run after the end time is reached; this only specifies the time period during which the job may start, not when the job must finish. The **Interval** property specifies how often the job should run. In the **SPMinuteSchedule** example, an **Interval** of **5** would indicate that the timer job should run every 5 minutes. An advantage of specifying a time window instead of an explicit start time is that SharePoint can run the timer job at different times, within the time window, on different servers in your server farm. This helps to ensure that your farm remains responsive when processor-intensive timer jobs are running.

To install a timer job, you complete the following steps:

1. Obtain a reference to the web application with which the timer job should be associated. You can obtain this by using the **properties** parameter passed to the **FeatureActivated** method.

- Use a Feature and Feature receiver
- Create a new instance of the job
- Create a schedule:
 - **SPMinuteSchedule**
 - **SPHourlySchedule**
 - **SPDailySchedule**
 - **SPWeeklySchedule**
 - **SPMonthlySchedule**
 - **SPYearlySchedule**
- Assign the schedule to the job
- Call the **Update** method

2. Create a new instance of the timer job class, specifying the constant name, and the web application as parameters for the constructor.
3. Create a schedule for the timer job.
4. Set the **Schedule** property of the timer job to the schedule object you created in the previous step.
5. Call the job's **Update** method to persist the changes.

The following code example shows a Feature receiver to install a timer job.

Timer Job Installer Feature Receiver

```
public class TimerJobInstaller : SPFeatureReceiver
{
    // Define a constant with the job name.
    const string JobName = "ContosoTimerJob";

    // Create a method to remove the job if it is installed.
    private void RemoveJob(SPWebApplication webApplication)
    {
        foreach (var job in webApplication.JobDefinitions)
        {
            if (job.Name.Equals(JobName))
            {
                job.Delete();
            }
        }
    }

    // Override the FeatureActivated method to install the job.
    public override void FeatureActivated(SPFeatureReceiverProperties properties)
    {
        // Obtain a reference to the web application.
        SPWebApplication webApplication = ((SPSite)properties.Feature.Parent).WebApplication;

        // Remove the timer job if it is already installed.
        RemoveJob(webApplication);

        // Create a new instance of the timer job.
        ContosoTimerJob timerJob = new ContosoTimerJob(JobName, webApplication);

        // Create a schedule for the job.
        SPMinuteSchedule schedule = new SPMinuteSchedule();
        schedule.BeginSecond = 0;
        schedule.EndSecond = 5;
        schedule.Interval = 10;

        // Assign the schedule to the job.
        timerJob.Schedule = schedule;

        // Call the Update method to persist the changes.
        timerJob.Update();
    }

    // Override the FeatureDeactivated method to uninstall the job.
    public override void FeatureDeactivated(SPFeatureReceiverProperties properties)
    {
        // Obtain a reference to the web application.
        SPWebApplication webApplication = ((SPSite)properties.Feature.Parent).WebApplication;

        // Remove the timer job.
        RemoveJob(webApplication);
    }
}
```

Demonstration: Examining Timer Job Schedules

This demonstration reviews the timer jobs used by SharePoint. You will see how timer jobs are scheduled and how you can use the SharePoint interface to force a timer job to run immediately. Finally, you will review the timer service process, and examine the process name and identity.

Demonstration Steps

- Start the 20488B-LON-SP-05 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, type **Central Administration**, and then press Enter.
- On the **Central Administration** website, click **Monitoring**.
- On the **Monitoring** page, under **Timer Jobs**, click **Review job definitions**.
- On the **Job Definitions** page, point out the standard timer jobs that are used by SharePoint. Point out that in addition to providing extensibility for developers, SharePoint also uses timer jobs extensively for internal operations.
- Click **App Installation Service**.
- On the **Edit Timer Job** page, point out the recurring schedule, and the **Run Now** button.
- Close Internet Explorer.
- On the Start screen, type **Services**, and then click **Services**.
- In **Services**, double-click the **SharePoint Timer Service**.
- In the **SharePoint Timer Service Properties (Local Computer)** dialog box, point out that the internal service name is **SPTimerV4**. Explain the importance of this for when you control services from a command line.
- Point out the path to the executable, and the executable name **OWSTIMER.exe**. Remind students that timer jobs run in this process and do not have a web context.
- On the **Log On** tab, point out that this service runs under the farm account credentials. Explain that, by default, this gives timer jobs full access to the farm, including write permissions on the configuration database.
- Click **OK**.
- Close Services.

Work Item Timer Jobs

Work item timer jobs are a special type of timer job specifically designed to process a set of items. A work item timer job works by processing items in a queue. A work item timer job, like other timer jobs, works on a scheduled basis; however, when a work item timer job is scheduled to run, SharePoint will only execute the timer job logic if at least one work item has been added to the queue for that timer job. This provides a two-part split for work item timer jobs:

- The work item timer job definition, which contains the logic for processing items.
- A queue of items to be processed.

You must develop the work item timer job definition, but you do not need to develop the queue; SharePoint provides a queue, persisted in the database, that you use. You must write code to add items to the queue. Your code to add items to the queue must run in the context of a site collection administrator, or use elevated privileges; for this reason, you cannot add items to a work item timer job queue by using a sandboxed solution.

Work item timer jobs are particularly beneficial if you need to process items across multiple lists, or need to only process particular items from a list, such as items that have changed or been added to the list. Alternatively, you could use an event receiver to process list items that have changes or been added, but work item timer jobs provide advantages over event receivers:

- They run in the OWSTIMER.exe process, which grants them more security permission than an event receiver that may run in the context of the current user, or the web application service account.
- They may run during off peak hours when the impact of processor intensive operations is less significant.

In this example, you might use a work item timer job in conjunction with an event receiver, by using the event receiver to add an added or change item to the work item timer job queue, and then processing the item in the work item timer job according to the work item timer job schedule (or at a later date if you specify a start date in the future when you add the item to the queue).

Finally, work item timer jobs enable you to process either a single item at a time, or to retrieve a set of items that are queued for processing and process them all at the same time. If you choose to implement batch processing, you can set a maximum batch size.

- Process a set (or queue) of items
- Queue managed by SharePoint
- Use code to add items to the queue
- Timer job run on a schedule
- Batch or single item processing

Developing a Work Item Timer Job

To develop a work item timer job, complete the following steps:

1. Create a new class that inherits from the **SPWorkItemJobDefinition** class (in the **Microsoft.SharePoint.Administration** namespace).
2. Add a default constructor to the class.
3. Add a non-default constructor that accepts two parameters:
 - A **string** representing the name of the work item timer job.
 - A **SPWebApplication** representing the web application that owns the timer job.
4. Modify the non-default constructor to call the equivalent constructor on the base class.
5. Override the **WorkItemType** method to return a **Guid** that identifies the work item timer job. This should always return the same value, and you use this when you queue an item to be processed by your work item timer job.
6. Override the **DisplayName** property to return a display name for your work item timer job. This should always return the same value.
7. Optionally, override the **BatchFetchLimit** property to return an **int** identifying the maximum number of work items that should be retrieved from the work item queue each time the timer job runs.
8. Override the **ProcessWorkItem**, or **ProcessWorkItems** method. You should override the **ProcessWorkItem** method to process a single item at a time or the **ProcessWorkItems** method if you will process a batch of items. Add your custom logic to this method. After your custom logic, add code to delete the work item from the work item queue to prevent it from being processed more than once.

- Create a class which derives from **SPWorkItemJobDefinition**
- Add a default constructor
- Add a non-default constructor:
 - Name (**string**)
 - Web application (**SPWebApplication**)
- Override the **WorkItemType** method
- Override the **DisplayName** property
- Optionally override the **BatchFetchLimit** property
- Override either the **ProcessWorkItem** or the **ProcessWorkItems** method

The following code example shows how to develop a custom work item timer job.

Work Item Timer Job Code Example

```
public class WorkItemTimerJob : SPWorkItemJobDefinition
{
    public WorkItemTimerJob()
    {
    }

    public WorkItemTimerJob(string name, SPWebApplication webApplication
        : base(name, webApplication)
    {
    }

    public override Guid WorkItemType()
    {
        return new Guid("{7B1F0F56-2A2D-4AA3-B962-FCAD66ED02C8}");
    }

    public override string DisplayName
    {
        get
        {
            return "Contoso Work Item Timer Job";
        }
    }

    public override int BatchFetchLimit
    {
        get
        {
            return 100;
        }
    }

    public override bool ProcessWorkItem(SPContentDatabase contentDatabase,
        SPWorkItemCollection workItems,
        SPWorkItem workItem,
        SPJobState jobState)
    {
        // Add code to process work item.

        // Delete the work item.
        workItems.DeleteWorkItem(workItem.Id);
    }
}
```

To deploy a work item timer job, you use the same procedure as you use to deploy a normal timer job. After you have deployed a work item timer job, you can add items to the work item queue for your timer job to process. You use the **AddWorkItem** method of an **SPSite** instance to add an item to the queue. You must use elevated privileges, or be logged in to SharePoint as a site collection administrator to call the **AddWorkItem** method. When you call the **AddWorkItem** method, you should provide the Guid that you return in the **WorkItemType** method as the **gWorkItemType** parameter.



Additional Reading: For more information about the **AddWorkItem** method, see *SPSite.AddWorkItem method* at <http://go.microsoft.com/fwlink/?LinkID=307045>.

Lesson 4

Storing Configuration Data

Most real-world SharePoint solutions need to store settings and configuration data. SharePoint provides options for storing configuration data. This lesson introduces the options for storing configuration data and provides guidance on when to choose the different types of configuration data storage.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the main options for storing configuration data.
- Explain how and when to use property bags to store configuration data.
- Use Web.config files to store configuration data.
- Describe how to store hierarchical configuration data.

Configuration Storage Options

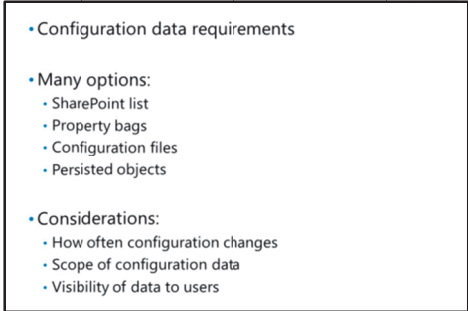
All but the very simplest of solutions need to store configuration data and settings. For example, you may need to store connection information for a database or web service. Alternatively, you may need to persist properties used by your solution. As a developer, you need a mechanism for storing that configuration data. In a Windows application, you could consider storing data in a configuration file, or use the configuration file to store a connection to a database. When you develop SharePoint solutions, you must consider the implications of a multiple server farm; for example, if you store configuration data in a file, that file must be copied to each server in the farm. SharePoint provides mechanisms that you can use to store configuration data—make sure that your data is available to all servers that might run your code in the server farm, including:

- SharePoint lists.
- Property bags.
- Configuration files.
- Persisted objects in the configuration database.

You should carefully consider how you will use your data when deciding the most appropriate storage mechanism. For example, your configuration data may change infrequently, such as a connection string for a database, or it may change regularly, such as a counter that increments each time a process is performed.

Also, the data that you store may be instance-dependent, such as identifying a list on a site, or may be common across all instances of your solution deployed across a web application, such as the URL or a web service used by your solution.

Another important consideration is whether users of the site should be able to interact with your configuration data. If you store your configuration data in a SharePoint list, users are likely to be able to see that data, and more importantly, edit that data. In some cases, this may be the desired behavior;

- 
- Configuration data requirements
 - Many options:
 - SharePoint list
 - Property bags
 - Configuration files
 - Persisted objects
 - Considerations:
 - How often configuration changes
 - Scope of configuration data
 - Visibility of data to users

however, this will often not be the case. If you store your data by using a format that enables a user to easily edit the data, before you attempt to use the data, you should always check to make sure that the data has not been changed in a way that makes it invalid.

Using Property Bags

Several SharePoint components have an associated property bag that you can use to store configuration settings and other data. A property bag is implemented as a hash table, which stores key and value pairs. Data stored in a property bag is stored in the content database (for farm and web application property bags) or configuration database (for site collection and other property bags). Examples of SharePoint components that expose property bags include:

- Farm (**SPFarm**)
- Web application (**SPWebApplication**)
- Site collection (**SPSite**)
- Site (**SPWeb**)
- List (**SPList**)

To add a value to a property bag, obtain a reference to the parent item, and then call the **Add** method on the **Properties** collection property, passing your key and value as parameters. To retrieve a value, you can use array notation on the **Properties** collection, and specify the key as the parameter.

You can update a stored value by using the array notation, and simply assigning a new value. As a security precaution, SharePoint does not permit you to update a value stored in a property bag while processing a page request with the **GET** verb. If you need to perform an update during a page request, you can set the **AllowUnsafeUpdates** property of the parent item to **true**; however, this is not recommended. To remove an item, from the property bag, set the item to **null**.

After you add or update an item in the property bag, you must always call the **Update** method on the **Properties** collection to persist your change to the database.

```
using (SPWeb web = SPContext.Current.Site.RootWeb)
{
    // Check an item does not already exist first.
    if(web.Properties["Key"] == null) {
        web.Properties.Add("Key", "Value"); //Add.
    }
    web.Properties["Key"] = "NewValue"; // Update.
    web.Properties["Key"] = null; // Delete.

    // Update after adding, updating or deleting.
    web.Properties.Update();

    // Retrieve a value.
    string storedValue = web.Properties["Key"];
}
```

The following code example shows how to use the property bag on a site (**SPWeb**).

SPWeb Property Bag Usage Code Example

```
using (SPWeb web = SPContext.Current.Site.RootWeb)
{
    // Always check an item does not already exist before attempting to add an item.
    if(web.Properties["Key"] == null)
    {
        // Add a new item to the property bag.
        web.Properties.Add("Key", "Value");
        web.Properties.Update();
    }

    // Retrieve a value stored in the property bag.
    string storedValue = web.Properties["Key"];

    // Update an item store in a property bag.
    web.Properties["Key"] = "NewValue";
    web.Properties.Update();

    // Remove an item from the property bag.
    web.Properties["Key"] = null;
    web.Properties.Update();
}
```

Manipulating Web.config Files

Another option for storing configuration data is the Web.config files. You can only store string values in the Web.config file. However, you could store a serialized representation of an object if you need to store a more complex object rather than a string.

Configuration files are a common storage location for configuration data; however, you should be aware of the following considerations before using a configuration file to store data for your SharePoint customization:

- You must make changes on all servers in the server farm; this could be time consuming if completed manually, and it could introduce a possibly unacceptable risk. In addition, this risks the farm being in an inconsistent state.
- You must restart the IIS worker process after making a change; this makes this option unsuitable if the data changes frequently.
- You cannot access configuration files from processes that do not run in the IIS worker process. This prevents you from using configuration files for timer jobs, or in sandboxed solutions.
- You must configure each web application separately. This may be an advantage or disadvantage, depending on your requirements, but it introduces risk of the farm being in an inconsistent state.

Despite the risks involved with configuration files, there are some situations where storing a configuration value in a configuration file is an appropriate choice, such as for database connection strings. SharePoint includes the **SPWebConfigModification** class, which you can use to make configuration file changes to the Web.config files for either a web application, or a SharePoint web service. If you use the

- Data with low change frequency
- Requires IIS restart
- Not compatible with timer jobs or sandboxed solutions
- Use **SPWebConfigModification** class to deploy changes to all servers

SPWebConfigModification class to make changes to the Web.config files, SharePoint will ensure that the change is made to all servers within your server farm, thereby dramatically reducing the risk of errors.

To modify the Web.config file for a web application or service application, complete the following steps:

1. Create a new instance of the **SPWebConfigModification** class.
2. Set the properties of the **SPWebConfigModification** class to define your configuration change.
3. Obtain a reference to the **SPWebApplication** instance that the configuration changes should apply to.
4. Add your instance of the **SPWebConfigModification** class to the **WebConfigModifications** collection property of the **SPWebApplication** instance.
5. Call the **Update** method of the **SPWebApplication** instance to persist the changes.
6. Repeat steps 3, 4 and 5 if you need to make the change to the Web.config files for multiple web applications.
7. Obtain a reference to the **SPWebService** that hosts the web applications. Typically, you use the static **ContentService** property to retrieve the appropriate reference.
8. Call the **ApplyWebConfigModifications** method of the **SPWebService** instance.

The following code example shows how to use code to modify the Web.config file for a web application.

Web.config File Modification Code Example

```
// Create a new instance of the SPWebConfigModification class.
SPWebConfigModification modification = new SPWebConfigModification();

// Set properties of the SPWebConfigModification instance.
// For example, modification.Path and modification.Value

// Obtain a reference to the web application for which you need to change the configuration
file.
SPSite site = new SPSite("http://sharepoint.contoso.com");
SPWebApplication webApplication = site.WebApplication;

//Add the SPWebConfigModification instance to the list of modifications for the web application.
webApplication.WebConfigModifications.Add(modification);

// Call the web applications Update method to save the changes.
webApplication.Update();

// Obtain a reference to the content service.
SPWebService service = SPWebService.ContentService;

// Call the ApplyWebConfigModifications method to apply the changes to the farm.
service.ApplyWebConfigModifications();
```

After you call the **ApplyWebConfigModifications** methods, SharePoint will schedule a timer job to process your updates. You must call the **ApplyWebConfigModifications** method in the context of a user who is an administrator on the web servers. The changes will normally be made quickly; however, there may be a delay before the changes are applied.



Additional Reading: For more information about how to add entries to Web.config files, see *How to: Add and Remove Web.config Settings Programmatically* at <http://go.microsoft.com/fwlink/?LinkID=307046>.

Storing Hierarchical Data

You may need to store hierarchical configuration data. SharePoint enables you to store hierarchical storage information in the SharePoint configuration database by using the **SPPersistedObject** base class (from the

Microsoft.SharePoint.Administration

namespace). This is sometimes known as the hierarchical object store.

To store hierarchical configuration data, you must first develop a class to represent your configuration values. To do this, complete the following steps:

1. Create a class that inherits from the **SPPersistedObject** class.
2. Annotate your class with a **Guid** attribute (**System.Runtime.InteropServices** namespace).
3. Add a default constructor to your class.
4. Add a non-default constructor to your class that accepts two parameters:
 - A **string** representing the name of the object.
 - A **SPPersistedObject** representing the parent object in the hierarchy.
5. Modify the non-default constructor to call the equivalent constructor on the base class.
6. Add fields and properties to your class. Avoid automatic properties as only fields can be stored.
7. Annotate any fields that should be persisted with the **Persisted** attribute (**Microsoft.SharePoint.Administration** namespace). You should only store fields that are of a type that is serializable.



Note: SharePoint serializes objects persisted in the hierarchical object store. If you attempt to persist a field that is not serializable, or if you do not include a default constructor, you may corrupt your configuration database.

- Inherit from **SPPersistedObject**, and annotate with **Guid** attribute
- Include default and non-default constructors:
 - `public PersistedClass() { }`
 - `public PersistedClass(string name, SPPersistedObject parent) : base(name, parent) { }`
- Add **Persisted** attributes to fields (not properties)
- Use **SPPersistedObject.GetChild<T>()** to retrieve values
- Require read and write permissions on the configuration database

The following code example shows how to develop a class to store configuration data in the hierarchical object store.

Persisted Object Code Example

```
// Annotate the class with a Guid attribute.
[Guid("EAD5BAE5-3C66-42B4-B0CC-039AA67373A4")]
public class PersistedClass : SPPersistedObject // Inherit from the SPPersistedObject class.
{
    // Always include a default constructor.
    public PersistedClass()
    {
    }

    // Always include a non-default constructor.
    public PersistedClass(string name, SPPersistedObject parent)
        : base(name, parent) // Class the base class constructor.
    {
    }
    [Persisted] // Annotate fields that must be stored.
    string Value1;

    [Persisted] // Annotate fields that must be stored.
    string Value2;

    [Persisted] // Annotate fields that must be stored.
    int Value 3;

    // Add methods and properties.
    ...
}
```

To store your configuration data, you simply create a new instance of your class by specifying a name and a parent object. For the root of your configuration, you will typically use either a **SPFarm** instance or a **SPWebApplication** instance, both of these classes inherit from the **SPPersistedObject** class. After you create an instance of your class, you should set properties on the class, before finally calling the classes **Update** method.

To retrieve values from the hierarchical object store, you use the generic **GetChild** method exposed by all objects that inherit from the **SPPersistedObject** class. You specify the type parameter, and the name of the object you want to retrieve.

The following code example shows how to store and retrieve items in the hierarchical object store.

Accessing Data in the Hierarchical Data Store Code Example

```
// Obtain a reference to the parent item.
SPFarm farm = SPFarm.Local;

// Create a root item.
// Create a new instance of the class by specifying the data item name, and root item.
PersistedClass rootClass = new PersistedClass("ContosoRootItem", farm);

// Set properties on the item.
rootClass.Value1 = "Stored Data";
rootClass.Value2 = "More Store Data";
rootClass.Value3 = 4;

// Call the Update method to persist the changes.
rootClass.Update();

// Create a new item as a child of the root item.
// Create a new instance of the class by specifying the data item name, and root item.
PersistedClass childClass = new PersistedClass("ContosoChildIte2m", rootClass);
```

```
// Set properties on the item.
childClass.Value1 = "Stored Data Item";

// Call the Update method to persist the changes.
childClass.Update();

// Retrieve an item from the hierarchical data store.
//Obtain a reference to the root item.
SPFarm farm = SPFarm.Local;

// Use the generic GetChild method to retrieve the item.
PersistedClass retrievedValue = farm.GetChild<PersistedClass>("ContosoRootItem");
```

Configuration data stored in the hierarchical object store is stored in the SharePoint configuration database. In most deployments, access to this database is restricted to the farm account and farm administrators. This limits when you can use the hierarchical object store as you cannot normally access it from when your code is running in the context of a user or a web application. However, the timer service does normally have access to the configuration database, making the hierarchical data store a convenient storage location when you are developing timer jobs.



Additional Reading: The SharePoint Guidance Library includes a reusable component named the Application Setting Manager. The Application Setting Manager enables you to store hierarchical configuration data by using alternative data stores, such as property bags. For more information about the Application Setting Manager, see *The Application Setting Manager* at <http://go.microsoft.com/fwlink/?LinkID=307047>.

Discussion Question

The instructor will now lead a discussion around the question: When would you choose each of the configuration storage options?

- SharePoint lists
- Property bags
- Hierarchical storage
- Web.config file

- When would you choose each of the configuration storage options?
 - SharePoint lists
 - Property bags
 - Hierarchical storage
 - Web.config file

Lab: Working with Server-Side Code

Scenario

To help Contoso managers monitor departmental expenses, they have decided that all expenses should be tracked in SharePoint. Your colleagues have already created custom lists for each department that employees can use to store expense details. They have also created a simple Web Part that informs users that they should now add their expenses to a SharePoint list.

Management wants to make employees more aware of how much their department spends on expenses. You will amend the Web Part so that it also contains an estimated departmental total. Because the number of expenses per department may grow very large, you do not want to enumerate every expense item in the list each time someone views the Web Part; instead, you will store the current total in the site property bag. You will create an event receiver that updates this total each time an item is added or changed in the expenses list.

Senior managers also want better oversight into departmental spending. To support senior management, you will create a timer job that will run once a day. The timer job will update a list with the latest totals from each department by retrieving the latest data from each departmental site. Both the overview list and the departmental Web Part will only display estimated data; Contoso's finance department will produce accurate expense reports by using alternative software. You will use farm solutions to deploy each component.

Objectives

After completing this lab, you will be able to:

- Develop and deploy an event receiver.
- Store and access configuration data.
- Create and deploy timer jobs.

Estimated Time: 45 minutes

- Virtual machine: 20488B-LON-SP-05
- User Name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Developing an Event Receiver

Scenario

In this exercise, you will create an event receiver. You will add code to the event receiver to handle items being added, updated, or deleted from a list. The code will update a property stored in a site-level property bag. You will then modify the default Feature to associate the event receiver with the correct list. Finally, you will deploy the event receiver to the server.

The main tasks for this exercise are as follows:

1. Create a New Solution
2. Create an Event Receiver
3. Create a Feature and Deploy the Event Receiver

► Task 1: Create a New Solution

- Start the 20488B-LON-SP-05 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.

- Open Visual Studio 2012.
 - Create a new project named **ExpensesEventReceiver** in the **E:\Labfiles\Starter** folder by using the **SharePoint 2013 - Empty Project** template. Configure the project to deploy as a farm solution and use the **http://sales.contoso.com** site for debugging.
- **Task 2: Create an Event Receiver**
- Add an event receiver to the **ExpensesEventReceiver** project named **ContosoExpensesEventReceiver**. The event receiver should handle the item added, item updated, and item deleted list item events.
 - Add a new private method named **UpdatePropertyBag** to the event receiver class. The method should take two parameters, an **SPWeb** object named **web**, and an **double** object named **change**.
 - Add logic to the **UpdatePropertyBag** method to perform the following actions:
 - a. Declares a **string** variable named **keyName** containing the key name **ContosoDepartmentalExpenseTotal**.
 - b. Declares a **double** variable named **currentValue** to store the current expense total. Initialize the variable to **0**.
 - c. Checks if an item currently exists in the property bag for the **SPWeb** instance passed as a parameter to the method that has the key specified in the **keyName** variable.
 - d. If an item already exists with the **keyName** key, parses the value as a **double**, and store the result in the **currentValue** variable.
 - e. If no item exists with the **keyName** key, adds a new item to the property bag by using the **keyName** variable as the key, and an empty string as the value.
 - f. Updates the value stored in the **currentValue** variable by adding the **change** parameter to the current value.
 - g. Stores the string equivalent of the **currentValue** variable in the property bag by using the **keyName** variable as the key.
 - Modify the **ItemAdding** method to perform the following actions:
 - a. Retrieves the value of the expense item that is adding by using the **AfterProperties** property of the **properties** parameter, and by using the indexer with the field name **InvoiceTotal**. You should store the result in a **double** object named **invoiceValue**. You will need to parse the value from a **string**.
 - b. Calls the **UpdatePropertyBag** method by using the **Web** property of the **properties** parameter, and the **invoiceValue** variable as parameters.
 - Modify the **ItemUpdating** method to perform the following actions:
 - a. Retrieves the old value of the expense item that is updating by using the **ListItem** property of the **properties** parameter, and by using the indexer with the field name **InvoiceTotal**. You should store the result in a **double** object named **previousInvoiceValue**.
 - b. Retrieves the new value of the expense item that is updating by using the **AfterProperties** property of the **properties** parameter, and by using the indexer with the field name **InvoiceTotal**. You should store the result in a **double** object named **newInvoiceValue**. You will need to parse the value from a **string**.
 - c. Declares a **double** variable named **change**, and initialize it to the result of subtracting the **previousInvoiceValue** from the **newInvoiceValue**.

- d. Calls the **UpdatePropertyBag** method by using the **Web** property of the **properties** parameter, and the **change** variable as parameters.
- Modify the **ItemDeleting** method to perform the following actions:
 - a. Retrieves the value of the expense item that is deleting by using the **BeforeProperties** property of the **properties** parameter, and by using the indexer with the field name **InvoiceTotal**. You should store the result in a **double** object named **invoiceValue**. You will need to parse the value from a **string**.
 - b. Calls the **UpdatePropertyBag** method by using the **Web** property of the **properties** parameter, and the negated **invoiceValue** variable as parameters.
- On the **BUILD** menu, click **Build Solution**. Correct any errors.
- ▶ **Task 3: Create a Feature and Deploy the Event Receiver**
Open the Elements.xml file.
- Modify the **Receivers** element to handle events on the list with the URL **Lists/Contoso%20Expenses**. You should use a **ListUrl** attribute, and remove all other attributes.
- Deploy the solution.
- Close Visual Studio.

Results: After completing this exercise, you should have developed and deployed an event receiver.

Exercise 2: Updating a Web Part

Scenario

In this exercise, you will modify an existing Web Part to retrieve a value stored in a site-level property bag. The Web Part will check if the value exists, and if it does display the information when the Web Part is rendered. You will deploy the Web Part, and add the Web Part to a page.

Finally, you will test the functionality of the event receiver developed in the previous exercise, and the Web Part by adding expense items to the expense tracking list and verifying that the correct total displays on the Web Part.

The main tasks for this exercise are as follows:

1. Modify a Web Part to Retrieve a Value from the Site Property Bag
2. Deploy the Web Part to the sales.contoso.com Site
3. Test the Web Part and Event Receiver by Adding Items to the Contoso Expenses List

▶ Task 1: Modify a Web Part to Retrieve a Value from the Site Property Bag

- Open the **ExpensesWebPart.sln** solution from the **E:\Labfiles\Starter\ExpensesWebPart** folder in Visual Studio.
- Open the **ExpensesInformationWebPart** Web Part code file.
- Add code to the **RenderContents** method that performs the following actions:
 - a. Retrieves a reference to the current **SPWeb** named **web** by using the **SPContext** object.
 - b. Checks if the property bag for the **web** object contains a property with the key **ContosoDepartmentalExpenseTotal**.

- c. If the property bag contains the key, uses the **Write** method of the **writer** parameter to write the code `<p>Estimated departmental expense total is: $$</p>`. You should replace **\$\$** with the value stored in the property bag.
- On the **BUILD** menu, click **Build Solution**. Correct any errors.

► **Task 2: Deploy the Web Part to the sales.contoso.com Site**

- Deploy the **ExpensesWebPart** Web Part.
- Use Internet Explorer to browse to **http://sales.contoso.com**.
- Add the **ExpensesWebPart - ExpensesInformationWebPart** to the home page of the **http://sales.contoso.com** site.
- Save the changes to the page.

► **Task 3: Test the Web Part and Event Receiver by Adding Items to the Contoso Expenses List**

- Verify that the **ExpensesWebPart - ExpensesInformationWebPart** Web Part does not currently display a total.
- Browse to the **Contoso Expenses** list.
- Add a new item with the following properties:

Property	Value
Title	Professional Services
Invoice Number	AA678
Supplier Name	Fabrikam
Invoice Total	150
Invoice Date	Use today's date

- Browse to the home page. Verify that the **ExpensesWebPart - ExpensesInformationWebPart** Web Part now displays an estimated total for departmental expenses.
- Close Internet Explorer.
- Close Visual Studio.

Results: After completing this exercise, you should have modified and deployed a Web Part, and tested the Web Part and event receiver.

Exercise 3: Creating a Timer Job

Scenario

In this exercise, you will create a timer job that enumerates the expense tracking lists for each department. The timer job will add items to a list of the manager's team site. You will then develop a Feature receiver to install the timer job, and then deploy the timer job to the server. You will test the timer job by viewing the overview list on the manager's team site.

The main tasks for this exercise are as follows:

1. Create a New Solution
2. Create a Timer Job Definition
3. Deploy the Timer Job
4. Test the Timer Job by Viewing the Expenses Overview List on the managers.contoso.com Site

► Task 1: Create a New Solution

- Start Visual Studio 2012.
- Create a new project named **ExpensesTimerJob** in the **E:\Labfiles\Starter** folder by using the **SharePoint 2013 - Empty Project** template. Configure the project to deploy as a farm solution and use the **http://managers.contoso.com** site for debugging.

► Task 2: Create a Timer Job Definition

- Add a new class to the **ExpensesTimerJob** project named **ContosoExpensesOverviewTimerJob**.
- Add **using** statements for the **Microsoft.SharePoint.Administration** and **Microsoft.SharePoint** namespaces.
- Modify the **ContosoExpensesOverviewTimerJob** to be **public**, and to inherit from the **SPJobDefinition** class.
- Add a **public** default constructor to the class.
- Add a **public** constructor to the class that accepts the following parameters:

Type	Name
string	name
SPWebApplication	webApplication
SPSever	server
SPJobLockType	lockType

- Modify the non-default constructor to call the non-default constructor of the base class.
- Add an override for the **Execute** method that performs the following actions:
 - a. Obtains a reference to the **http://managers.contoso.com** site collection named **managerSite**.
 - b. Obtains a reference to the root site in the **http://managers.contoso.com** site collection name **managerWeb**.
 - c. Obtains a reference to the **Expenses Overview** list on the **managerWeb** site names **overviewList**.

- d. Removes any existing items from the **overviewList** list.
- e. For each site collection in the current web application, it performs the following actions:
 - i. Obtains a reference to the root site in the site collection named **departmentWeb**.
 - ii. Retrieves the **Contoso Expenses** list, if it exists. If the site does not contain a list named **Contoso Expenses**, it takes no further action.
 - iii. Declares a **double** variable named **departmentTotal**, and sets the variable to the result of summing the **InvoiceTotal** fields for each item in the **Contoso Expenses** list.
 - iv. Retrieves the department name from the site URL. All of the sites use the naming convention `http://departmentname.contoso.com`.
 - v. Adds a new item to the **overviewList**. It sets the **Title** field to the department name, and the **Expense Total** field to the **departmentTotal** variable.
- On the **BUILD** menu, click **Build Solution**. Correct any errors.

► **Task 3: Deploy the Timer Job**

- Add a Feature named **ContosoExpensesOverviewTimerJobInstaller** to the **ExpensesTimerJob** project with the following properties:

Property	Value
Title	Contoso Expenses Overview Timer Job Installer
Scope	Site

- Add a Feature receiver for the **ContosoExpensesOverviewTimerJobInstaller** Feature.
- Add a **using** statement to the Feature receiver class for the **Microsoft.SharePoint.Administration** namespace.
- In the **ContosoExpensesOverviewTimerJobInstallerEventReceiver** class, declare a constant **string** variable named **timerJobName**, and initialize it to **ExpensesOverviewJob**.
- Add a private method named **deleteJob**, which accepts an **SPWebApplication** object as a parameter and performs the following action:
 - a. For each job definition in the **JobDefinitions** property of the web application reference passed as a parameter, if the **Name** property of the job definition matches the **timerJobName** variable, it calls the **Delete** method on that timer job definition.
- Implement the **FeatureActivated** method by adding code that performs the following actions:
 - a. Retrieves a reference to the current web application named **webApplication** by using the properties of the **Feature** property of the **properties** parameter.
 - b. Calls the **deleteJob** method by using the **webApplication** reference as the parameter.
 - c. Creates a new instance of the **ContosoExpensesOverviewTimerJob** class named **timerJob** by using the following parameters:

Parameter	Value
name	timerJobName
webApplication	webApplication

Parameter	Value
server	null
lockType	SPJobLockType.Job

- d. Creates a new instance of the **SPMinuteSchedule** class named **schedule**.
- e. Sets the following properties on the schedule object:

Property	Value
BeginSecond	1
EndSecond	5
Interval	2

- f. Sets the **Schedule** property of the **timerJob** object to the **schedule** object.
 - g. Calls the **Update** method of the **timerJob** object.
- Implement the **FeatureDeactivating** method by adding code that performs the following actions:
 - a. Retrieves a reference to the current web application named **webApplication** by using the properties of the **Feature** property of the **properties** parameter.
 - b. Calls the **deleteJob** method by using the **webApplication** reference as the parameter.
 - Deploy the **ExpensesTimerJob** project.



Note: In this task, you create a schedule that runs every two minutes. This is to make testing easier because the job will run frequently. This would be replaced with a more appropriate schedule in a production version of this timer job.

► Task 4: Test the Timer Job by Viewing the Expenses Overview List on the managers.contoso.com Site

- Use Internet Explorer to browse to the **http://managers.contoso.com** site.
- On the **http://managers.contoso.com** site, browse to the **Expenses Overview** list.
- Verify that the **Expenses Overview** list contains a list item for each department. You may need to wait up to two minutes for the timer job to run and populate the list.
- Close Internet Explorer.
- Close Visual Studio.

Results: After completing this exercise, you should have developed, deployed, and tested a timer job.

Module Review and Takeaways

In this module, you learned how to create custom SharePoint components that run code on the server. You learned how you can write code to respond to events in SharePoint, and how you can write code that runs on a scheduled basis. Finally, you learned some of the techniques that you can use to store configuration data for custom components, together with some of the advantages and disadvantages of each approach.

Review Question(s)

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
A visual Web Part cannot be installed in a sandboxed solution.	

Test Your Knowledge

Question	
Which of the following identifies the default synchronization for events?	
Select the correct answer.	
<input type="checkbox"/>	Both before and after events run synchronously.
<input type="checkbox"/>	Both before and after events run asynchronously.
<input type="checkbox"/>	Before events run synchronously, and after events run asynchronously.
<input type="checkbox"/>	Before events run asynchronously, and after events run synchronously.
<input type="checkbox"/>	Before events run synchronously, after events do not have a default synchronization.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
Timer jobs run with full control, in a separate process from the web worker process.	

Test Your Knowledge

Question	
Which of the following statements is not true about storing configuration data in a property bag?	
Select the correct answer.	
<input type="checkbox"/>	Properties stored in a property bag can only be accessed by the process that added them to the property bag.
<input type="checkbox"/>	Sandboxed solutions can use property bags.
<input type="checkbox"/>	Property bags can store any object that can be serialized as XML.
<input type="checkbox"/>	Configuration values stored in a property bag are saved in the SharePoint databases.
<input type="checkbox"/>	A property bag is implemented as a hash table, which stores key and value pairs.

Module 6

Managing Identity and Permissions

Contents:

Module Overview	6-1
Lesson 1: Understanding Identity Management in SharePoint 2013	6-2
Lesson 2: Managing Permissions in SharePoint 2013	6-11
Lab A: Managing Permissions Programmatically in SharePoint 2013	6-17
Lesson 3: Configuring Forms-Based Authentication	6-19
Lesson 4: Customizing the Authentication Experience	6-28
Lab B: Creating and Deploying a Custom Claims Provider	6-34
Module Review and Takeaways	6-40

Module Overview

In any system that deals with business-critical or confidential data, it is essential to control access to that data. For example, if you store employees' home addresses, you should ensure that customers cannot access that data while human resources workers can access and modify that data. Such access control requires two processes. First, the system must positively identify each user. This process is known as authentication and often requires a username and password from the user or other credentials such as those contained on a smart card. Second, the system must determine what level of access the user should receive for each resource. This is known as authorization and often involves checking user identity information against a set of permissions on the resource. In this module, you will see how SharePoint performs authentication and authorization and how administrators and developers can manage and configure these processes.

Objectives

After completing this module, you will be able to:

- Describe how authentication and identity management work in SharePoint 2013.
- Verify and manage permissions programmatically in SharePoint 2013.
- Create and configure custom membership providers and role managers for forms-based authentication.
- Create claims providers and customize the sign-in experience.

Lesson 1

Understanding Identity Management in SharePoint 2013

SharePoint can identify users through a variety of mechanisms, including Windows Integrated authentication, ASP.NET Forms-Based Authentication (FBA), and federated authentication. Once a user has been positively identified, SharePoint can assemble a set of affirmed facts about the user into a signed security token. These facts are called claims. In this lesson, you will learn about these authentication mechanisms and claims so that you can add custom claims. You will also learn how to use impersonation in code to alter access levels for a user.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how classic authentication mode differs from claims authentication mode.
- List the components of a SharePoint farm that support authentication.
- Select the most appropriate authentication method for a given scenario.
- Describe the programmatic classes that SharePoint uses to represent users.
- Use impersonation to run code under the security context of a specific user.
- Describe scenarios in which impersonation is useful.

Authentication in SharePoint

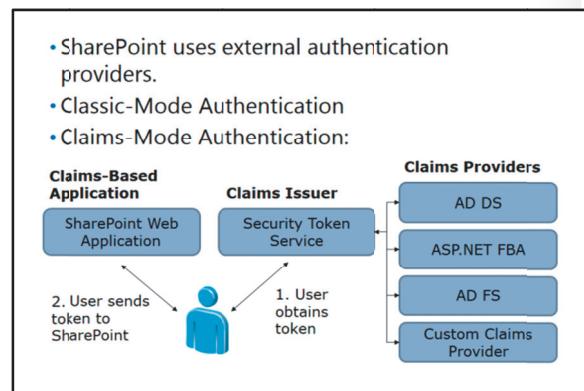
In SharePoint, as in many other systems, authentication is the process by which a user is positively identified. The user must provide credentials, which SharePoint can check with an authentication provider, in order to authenticate. Credentials can include:

- A username and password.
- A smart card that contains a digitally signed certificate for the user's identity.
- A fingerprint, iris scan or some other biometric data.

The credentials required are set by the authentication provider. SharePoint itself does not include an authentication provider. Instead it uses a trusted external provider, such as Active Directory Domain Services (AD DS) or an ASP.NET forms authentication provider.

Claims Authentication Mode

Claims authentication mode is supported by SharePoint 2010 and later versions. It was introduced with the Windows Identity Foundation (WIF). In this mode, credentials are sent to the authentication provider and checked in just the same way as in classic mode. However, the information returned to SharePoint for authorization is more flexible because it includes claims. A claim is a signed affirmation about the authenticated user. For example, a claim may affirm that the user's account name is alfredh@contoso.com or that the user is a member of the security group Managers. In fact, a claim can affirm any information about the user. For example, a claim can affirm that the user is over 18 or in North America.



By using claims in SharePoint, you can authorize access to resources more flexibly than with classic authentication. For example, you could create a custom claim provider that affirms whether a user possesses a driver's license by checking the HR database. You could then restrict access to the Car Pool SharePoint site to those users who have licenses.

In claims authentication, there are three roles taken by system components to authenticate and authorize users:

- *Claims Provider.* The claims provider is the application or component that returns a list of claims that about the user. SharePoint includes a number of claims providers out of the box. Each claim provider works with a different authentication provider and returns different claims. For example, the AD DS claims provider returns claims to confirm a user's AD DS account and security group membership. The ASP.NET Forms-Based Authentication (FBA) claims provider returns claims to confirm a user's FBA membership and roles.
- *Claims Issuer.* The claims issuer is the application or component that requests credentials from the user, forwards the credentials to a claims provider, and packages the claims returned into a security token. In SharePoint, the claims issuer is the Security Token Service (STS).
- *Relying Party.* The relying party is the application or component that trusts the claims that the user presents in a security token. Because the security token is signed by a claims issuer that the relying party trusts, the relying party accepts the claims and uses them to authorize access to resources. In SharePoint, the relying party is the SharePoint web application.

By supporting claims authentication, SharePoint ensures that you can integrate with a wide range of standards-based authentication providers and enables you to comply with local regulations in a flexible way.

The Security Token Service (STS) must be running in your SharePoint farm in order to enable claims authentication.

Classic Authentication Mode

Classic authentication mode is supported by every version of SharePoint. In this mode, the user enters credentials and SharePoint sends them to the authentication provider. The provider checks the credentials against those it has stored for that user. If the credentials are correct, the provider returns a range of information to SharePoint, which SharePoint will use to authorize access to resources. This information can include:

- The username.
- The security groups that the user is a member of.

SharePoint permissions are usually granted to user accounts or security groups.

Classic mode authentication is simple and useful when both SharePoint and the authentication provider are run by the same organization. However, classic mode authentication has limitations that arise when you want to trust another organization to authenticate users. For example:

- *Legislation Compliance.* In many jurisdictions, laws prevent an organization from sharing full user data with another organization.
- *Standards Compliance.* SharePoint can only trust an external authentication provider to validate users if both SharePoint and the external provider support the same standards for authentication.
- *Office Web Apps.* You cannot Office Web Apps in web applications that use classic mode authentication.

- **Authentication Providers.** SharePoint can only use AD DS to authenticate credentials in a classic-mode web application. You cannot use classic-mode authentication with ASP.NET FBA or federated authentication.

Because of these limitations, classic mode authentication is no longer recommended in SharePoint 2013 and is supported for backwards-compatibility only.



Best Practice: Always use claims authentication mode unless you have some custom component that cannot be used with claims authentication. Classic authentication is only supported in SharePoint 2013 for backwards compatibility with custom components written for earlier versions of SharePoint.

You cannot create web applications that use classic authentication in SharePoint Central Administration. Instead you must use the New-SPWebApplication cmdlet in PowerShell. For more information see the following link:



Additional Reading: For more information on how to create web applications that use classic mode authentications, see *Create web applications that use classic mode authentication in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=313080>.

Authentication Types and Methods

SharePoint supports several different authentication providers. Each of these authentication providers supports a different type of authentication. For example, an ASP.NET authentication provider support Forms-Based Authentication (FBA). In order to choose the most appropriate provider and authentication type for your scenario, you must understand their capabilities, advantages, and disadvantages.

Windows Authentication

If you use Windows Authentication in your SharePoint farm, SharePoint uses AD DS as the authentication provider. The authentication process is as follows:

1. The client makes an anonymous request to access a SharePoint resource.
2. If the SharePoint resource permissions do not allow anonymous access, SharePoint requests authentication credentials from the client.
3. The client returns the credentials to the SharePoint server. If the client is Internet Explorer and the user is already authenticated under Windows, then these credentials are sent in the background. Otherwise, the browser prompts the user for credentials. The credentials can be sent by using one of three methods:
 - a. *Kerberos.* The Kerberos protocol is an advanced and highly secure method of encrypted credential exchange. This is the default Windows authentication method.

- **Windows Authentication**
 - Only supported by Internet Explorer
 - Requires an AD DS user account
 - Authentication methods include Kerberos, NTLM, and Basic
- **FBA Authentication**
 - Uses ASP.NET membership providers for authentication
 - Uses ASP.NET role provider for group membership
 - Many providers available
- **SAML Authentication**
 - Supports federated authentication providers such as AD FS

- b. *NT LAN Manager*. The NT LAN Manager (NTLM) protocol is a secured method of credential exchange that was introduced with Window NT and is supported by all versions of Internet Explorer.
 - c. *Basic*. In basic authentication, credentials are exchanged in plain text. This is not a recommended authentication method, because malicious users can intercept the credentials and use them to compromise SharePoint security.
4. SharePoint server sends the credentials to an AD DS domain controller, which authenticates them and returns a Windows security token.
 5. SharePoint server queries the AD DS domain controller for a list of the security groups to which the user belongs.
 6. The SharePoint STS uses the user account and group information to create a claims security token. This is cached for the whole SharePoint farm.
 7. SharePoint checks the claims in the security token against the permissions on the requested resource. If the user is permitted access, the resource is returned to the browser and displayed to the user.
 8. Windows authentication is a secure and flexible authentication type. However, you can only rely on it, when you know that all users have Internet Explorer and an account in AD DS.

Forms-Based Authentication

FBA is an authentication type that is supported by ASP.NET. Since SharePoint is built on ASP.NET, you can use FBA to identify users when they try to access SharePoint resources. The authentication process is as follows:

1. The client makes an anonymous request to access a SharePoint resource.
2. If the SharePoint resource permissions do not allow anonymous access, SharePoint redirects the browser to a logon page. This page is usually an ASP.NET Web Forms page whose address you can configure in web.config. You should use Secure Sockets Layer (SSL) to encrypt this page and ensure that credentials are not exchanged in plain text.
3. The user fills in the form and the browser returns the credentials to the SharePoint server.
4. SharePoint sends the credentials to an ASP.NET membership provider. A membership provider is a component that checks credentials against its store of user accounts. ASP.NET includes membership providers for many different kinds of account stores.
5. If the credentials are correct, SharePoint queries the ASP.NET role provider to determine the roles that the user account is a member of. ASP.NET roles are similar to groups in AD DS.
6. The SharePoint STS uses the user account and roles to create a claims security token. This is cached for the whole SharePoint farm.
7. SharePoint checks the claims in the security token against the permissions on the requested resource. If the user is permitted access, the resource is returned to the browser and displayed to the user.

FBA is very flexible because all browsers support the forms that FBA uses to collect credentials. Furthermore, because there are many membership providers available, you can use many different types of store for user accounts. For example, membership providers are available for Microsoft SQL Server, Window Azure SQL Database, Access, Oracle, and other databases. Alternatively you can build a custom membership provider if you want to store user accounts in some other information store. Similarly, there are many role providers available and you can build a custom role provider.



Note: In lesson 3 of this module, you will learn how to create custom membership providers and roles providers for FBA authentication.

Security Assertion Markup Language Authentication

Security Assertion Markup Language (SAML) is a standard for federated authentication. In federated authentication, SharePoint uses a federated authentication provider to check credentials. The federated provider trust some other external system to authenticate credentials. Active Directory Federation Services (AD FS) is an example of a federated provider. AD FS can trust AD DS or it can trust many other types of authentication provider.

Before you can use SAML authentication in SharePoint you must establish two trust relationships:

- Federation Provider to Authentication Provider
- SharePoint Provider to Federation Provider

The authentication process is as follows:

1. The client makes an anonymous request to access a SharePoint resource.
2. If the SharePoint resource permissions do not allow anonymous access, the SharePoint server redirects the browser to the federated provider to obtain a SAML login page.
3. The user types the credentials and sends them to the federated provider with a request for a SAML security token.
4. The federated provider validates the credentials with the authentication provider.
5. If the credentials are valid, the federated provider creates a signed SAML security token and returns it to the browser.
6. The client makes a new request to access the SharePoint resource. This request includes the SAML token.
7. The SharePoint STS uses the information in the SAML token to create a claims security token. This is cached for the whole SharePoint farm.
8. SharePoint checks the claims in the security token against the permissions on the requested resource. If the user is permitted access, the resource is returned to the browser and displayed to the user

SAML authentication is flexible because federated authentication is done using wide-support authentication protocols. A federated authentication provider such as AD FS supports a wide range of authentication providers. By using AD FS, you can enable SharePoint to check credentials against many third party authentication systems that are not supported by Windows or FBA authentication.

How SharePoint Represents Users

In order to work with authenticated users programmatically, it is important to understand the classes that SharePoint uses to represent those users. In this topic, you will learn about these classes and see simple examples of their use.

Security Principal Classes

A security principal is an object to which you can assign permissions to a resource. Sometimes you might want to assign permissions directly to a user. For example, in a user profile site, you might want to assign full control permission to the "My Projects" list. However, it is often more manageable to assign permissions to group objects such as AD DS security groups or ASP.NET roles. For example, in the Human Resources site, you might assign read and change permissions on the Employees list to the HRUsers group. When a new HR user starts with your company, as long as you place their user account in the HRUsers group, the new user automatically receives access to the list by virtue of their group membership.

SharePoint uses the following classes to represent security principals. All these classes are in the **Microsoft.SharePoint** namespace:

- *SPUser*. This class represents a user account. You can access the display name of the user by using the **Name** property and the user account name by using the **LoginName** property. Depending on the authentication type, the **SPUser** object may represent an AD DS user account, an ASP.NET membership provider account, or an account from a federated authentication provider.
- *SPGroup*. This class represents a group or role. You can access the display name of the group by using the **Name** property and the members of the group by using the **Users** property. Depending on the authentication type, the **SPGroup** object may represent an AD DS security group, an ASP.NET role, or a group from a federated authentication provider.
- *SPPrincipal*. Both **SPUser** and **SPGroup** inherit from the class **SPPrincipal**. This class represents a general security principal.

To get the **SPUser** object for the current user, you can use the following code:

Getting the Current User

```
SPUser user = SPContext.Current.Web.CurrentUser;

labelCurrentUsername.Text = "The current user is: " + user.Name;
```

The users of a SharePoint site include users who have been granted permissions directly, users who have been granted permissions through a group who have then visited the site, and users who have been referenced in a person field in the site, such as being assigned a task. Obtain this list of users with the following code:

Obtaining the Users of a Site

```
SPUserCollection users = SPContext.Current.Web.AllUsers;

labelUserCount.Text = "There are " + users.Count + " users in this site.";
```

```
• SPUser
• SPGroup
• SPPrincipal

SPUser user =
    SPContext.Current.Web.CurrentUser;

SPUserCollection users =
    SPContext.Current.Web.AllUsers;
```


Impersonation

Under most circumstances, once the user has been authenticated, you run code under the security context of their user account. In this way, you ensure that permissions granted to that account, or to groups in which that account is a member, are applied.

Occasionally you may wish to run code under a different security context. For example, you may wish to enable a user to execute a specific operation that requires administrative privileges but you do not wish to make the user an administrator. For such scenarios, developers can use impersonation.

• Using Elevated Privileges

```
SPSecurity.RunWithElevatedPrivileges(delegate () {
    using (SPSite site = new SPSite("http://site")) {
        //Execute operations here }
    });
```

• Impersonating a Specific User

```
using (SPSite site =
    new SPSite(SPContext.Current.Site.Url, accessToken))
{
    //Operations executed in this using block have the
    //permissions of the SpecialAccess account
}
```



Best Practice: The entire security infrastructure relies on permissions applied to security principals such as user accounts and groups. When you use impersonation carelessly, it is possible to compromise this infrastructure. You are responsible for any security breaches that result from impersonation. Ensure that you use impersonation only when it is absolutely necessary and ensure that you properly validate user input.

Running Code with Elevated Privileges

You can use the **SPSecurity.RunWithElevatedPrivileges** method to complete any operation that requires a high level of access to SharePoint content. Any code run with this method executes under the security context of the SharePoint system account. By impersonating the system account in this way, you effectively grant the code full control access to all SharePoint content. This a very powerful technique that must be used with care.

The following code shows how to use the **RunWithElevatedPrivileges** method. In this example the code deletes a list from the site, even though the current user only has read and change permissions to that list.

Running Code with Elevated Privileges

```
SPSecurity.RunWithElevatedPrivileges(delegate () {
    using (SPSite site = new SPSite("http://intranet.contoso.com"))
    {
        using (SPWeb web = site.OpenWeb())
        {
            SPList discussionList = web.Lists["Discussion"];
            discussionList.Delete();
        }
    }
});
```

Remember the following issues when you use the **RunWithElevatedPrivileges** method.

Common Issues and Troubleshooting Tips

Common Issue	Troubleshooting Tip
Elevated privileges do not apply to an SPSite object you obtain outside the RunWithElevatedPrivileges method.	
Elevated privileges do not apply if you use the SPContext.Current.Site property to obtain the site collection.	
Updates to items do not apply.	

Running Code as a Specific User

The **RunWithElevatedPrivileges** method always grants the permissions required to perform any task on SharePoint content, because the code within the method runs in the security context of the SharePoint system. However, sometimes you may want to impersonate a specific user who is less privileged than the system account. You can do this by obtaining and using the user token associated with that account.

Use this technique when you want a different set of permissions from the current user but you don't want to grant full control permissions to all SharePoint content.

The following example shows how to run code under the security context of an account named "SpecialAccess".

Impersonating a Specific User

```
//First, get the user token for the account named "SpecialAccess"
SPUser specialAccessUser = SPContext.Current.Web.AllUsers["@CONTOSO\SpecialAccess"];
SPUserToken specialAccessToken = specialAccessUser.UserToken;

//Now get the site collection, passing the user token
using (SPSite site = new SPSite(SPContext.Current.Site.Url, specialAccessToken))
{
    //Operations executed in this using block have the permissions of the SpecialAccess account
}
```

Discussion: Scenarios for Impersonation

Read through the following scenarios. In each case, discuss whether impersonation is necessary and, if so, which impersonation method is most appropriate.

Setup Code in a Farm Solution

You have a farm solution that, for auditing purposes, records all operations in a list in the top level SharePoint site for the site collection in which it is installed. You want to create the list during installation by running code in a feature receiver.

- Discuss the following scenarios:
- Setup Code in a Farm Solution
 - Setting Permissions in a SharePoint List
 - Recording Approvals

However, you want to ensure that no SharePoint user accounts, including accounts belonging to farm administrators, have full control over this list.

Setting Permissions in a SharePoint List

You have an HR solution that uses a SharePoint list to store employee information. You want to ensure that HR managers can grant read and change permissions to HR employees for that list.

Recording Approvals

You are writing a solution that manages an authoring process. Authors, who are not employees of your company, place their documents in a SharePoint list, where they are reviewed and approved by your team of editors. You want to ensure that each author cannot determine who reviewed their document but can be sure that it was approved.

Lesson 2

Managing Permissions in SharePoint 2013

Authentication is only the first stage of the process by which you secure SharePoint resources. Once SharePoint has identified a user, the next stage is to determine whether that user should have access to the item, document, or resource that they have requested. This process is known as authorization and it is performed by checking permissions. If appropriate permissions have been granted to the user account or to a group that the account is a member of, the operation is permitted to proceed. In this lesson, you will see how to manage these permission in code.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the classes that you can use to programmatically manage permissions in SharePoint 2013.
- Configure permissions inheritance and anonymous access in a custom solution.
- Check permissions in managed code.
- Assign permissions and create permission levels in managed code.

Permissions Classes

In order to assign permissions to security principals in SharePoint 2013, you must first understand the classes that SharePoint uses represent lists, libraries, permission levels, and permission assignments. SharePoint permissions work in the following way:

- Permissions can be set on any list, library or SharePoint site. An object that you can restrict access to with permissions is known as a securable object.
- By default, permissions are inherited by child objects. For example, a permission set on a SharePoint site is inherited by a list within that site.
- Individual permissions can be combined into permission levels. For example, the default **Read** permissions level, includes the **View List Items** and **OpenItems** permissions. You can create custom permissions levels in code.
- To grant a permission level to a user or group, you must create an assignment for that permission level.

- `SPSecurableObject` represents a list, library, website or item
- `SPRoleDefinition` represents a permissions level
- `SPRoleAssignment` represents the assignment of a permission level to a security principal such as a user or group
- `SPRoleDefinitionBinding` defines the role definitions bound to a role assignment object

To manage permissions in code, you need to understand the following classes

- *SPSecurableObject*. This class represents a list, library, SharePoint site or any other securable object. This is an abstract class. Any securable object inherits properties and methods from this class. For example, on any securable object you can call the **CheckPermissions** method to determine the permissions level for the current user.

- *SPRoleDefinition*. This class represents a permission level. Each SharePoint site has a collection of these objects that represents the default permissions levels. You can add new **SPRoleDefinition** objects to this collection to define custom permissions levels.
- *SPRoleAssignment*. This class represents a permission level assigned to a user or group. For a securable object you can use **SPRoleAssignment** objects to grant a permission level to a security principal.
- *SPRoleDefinitionBinding*. This class defines the role definitions that are bound to a role assignment object.

The following code is a simple example that shows how to use the above objects to assign a permission level named "SpecialAccess" to a user account named "HRManager".

Using Permission Management Classes

```
using(SPWeb hrWeb = SPContext.Current.Site.AllWebs["HumanResources"])
{
    //Get the permission levels defined for this website
    SPSRoleDefinitionCollection roleDefinitions = hrWeb.RoleDefinitions;

    //Get the collection of role assignments
    SPSRoleAssignmentCollection roleAssignments = hrWeb.RoleAssignments;

    //Create a new role assignment
    SPSRoleAssignment newRoleAssignment = new
        SPSRoleAssignment("CONTOSO\HRManager", "HRManager@contoso.com", "HR Manager", "");

    //Get the permission level binding for the new role assignment
    SPSRoleDefinitionBindingCollection roleDefinitionBindings =
        newRoleAssignment.RoleDefinitionBindings;

    //Add the role definition to the bindings for the assignment
    roleDefinitionBindings.Add(roleDefinitions["SpecialAccess"]);

    //Add the new role assignment to the collection of assignments in the site
    roleAssignments.Add(newRoleAssignment);
}
```

Checking Permissions

Permissions on SharePoint resources apply at all times. If your code attempts an operation that is not permitted, the user will see access denied message. Therefore it is not necessary to add permission checking code to your solution to maintain security.

However, in some scenarios, you may wish to check permissions and take actions to help the user. For example, if the user does not have the necessary permission for an operation, you can disable a button or other user interface element.

```
if (website.DoesUserHavePermissions(user.LoginName,
    SPBasePermissions.EditListItems))
{
    //User can edit items in lists
    editButton.Visible = true;
}
else
{
    //User cannot edit items in lists
    editButton.Visible = false;
}
```

The **SPSecurableObject** base class defines the following methods to help you check permissions. You can use this methods on any SharePoint class that inherits from the **SPSecurableObject** class. This includes **SPSite**, **SPWeb**, **SPList**, and other classes:

- *DoesUserHavePermissions*. Call this method on any securable object and pass a base permission from the **SPBasePermissions** enumeration. For example, to check if the user can modify items in a list, pass **SPBasePermissions.EditListItems**. If the user has the correct permissions, the method returns true.
- *CheckPermissions*. This method is very similar to **DoesUserHavePermissions** and takes a member of the **SPBasePermissions** enumeration as a parameter. However, if the user does not have the required permissions, the method throws an exception of the type **UnauthorizedAccessException**.
- *GetUserEffectivePermissionInfo*. This method returns an **SPPermissionInfo** object. You can use the properties of this object to determine the permissions and role assignments possessed by the current user on the securable object.
- *GetUserEffectivePermissions*. This method returns all the effective permissions that the current user has on the securable object. The return value is a bitwise combination of values from the **SPBasePermissions** enumeration.

The following code shows how to check if the current user has the **EditListItems** base permission in the current website.

Checking for a Specific Permission

```
//Get the current web site and user
using (SPWeb website = SPContext.Current.Web)
{
    SPUser user = website.CurrentUser;

    if (website.DoesUserHavePermissions(user.LoginName, SPBasePermissions.EditListItems))
    {
        //User can edit items in lists
        editButton.Visible = true;
    }
    else
    {
        //User cannot edit items in lists
        editButton.Visible = false;
    }
}
```

Assigning Permissions

SharePoint base permissions include individual operations, such as **ApproveItems**, **AddListItems**, **ManageWeb** and so on. The SharePoint base permissions are all members of the **SPBasePermissions** enumeration. A permissions level consists of a common set of base permissions. For example, the **Read** permission level includes the base permissions **ViewListItems**, **OpenItems**, and **ViewPages**. A permissions level is also known as a role definition. For this reason,

- Assigning a Permissions Level
 - Create a new **SPRoleAssignment**
 - Add a role definition binding to the assignment
 - Add the assignment to the **RoleAssignments** collection on the securable object
- Creating a Custom Permissions Level
 - Create a new **SPRoleDefinition**
 - Add permissions to the **BasePermissions** collection
 - Add the role definition to the **RoleDefinitions** collection on the website

you use the **SPRoleDefinition** class to manage permissions levels.

In this topic, you will see how to assign a default permissions level to a user and how to create a custom permissions level.

Assigning a Default Permissions Level

To assign a permission level to a security principal, such as a user or group, you must create a new instance of the **SPRoleAssignment** class and pass the security principal to the constructor. You then add a role definition binding to the role assignment and the role assignment to the role assignments collection on the securable object.

The following code shows how to assign the **Read** permissions level to a group named "Viewers".

Assigning a Permissions Level

```
//Get the current web site
using (SPWeb website = SPContext.Current.Web)
{
    //Create the role assignment object
    SPSRoleAssignment assignment = new SPSRoleAssignment(website.SiteGroups["Viewers"]);

    //Add a binding to the Read permissions level
    assignment.RoleDefinitionBinding.Add(website.RoleDefinitions["Read"]);

    //Add the new role assignment to the website's assignments collection
    website.RoleAssignments.Add(assignment);

    //Update the site
    website.Update();
}
```

Creating a Custom Permissions Level

To keep permissions simple, you should use the default permissions levels wherever possible. However, sometimes, if you have very specific security requirements, you may need to create a custom permissions level. For example, you may want a permissions level that allows used to add, edit, and view items, but not to delete items.

The following code shows how to create a custom permissions level for a SharePoint website. You complete this operation by using the **SPRoleDefinition** class to represent the new permissions level.

Creating a Custom Permissions Level

```
//Get the current web site
using (SPWeb website = SPContext.Current.Web)
{
    //Create the new permissions level
    SPSRoleDefinition customPermissionsLevel = new SPSRoleDefinition();

    //Set the name and base permissions for the permissions level
    customPermissionsLevel.Name = "EditButNotDelete";
    customPermissionsLevel.BasePermissions = SPBasePermissions.AddListItems |
        SPBasePermissions.OpenItems | SPBasePermissions.ViewListItems |
        SPBasePermissions.EditListItems;

    //Add the permissions level to the website
    website.RoleDefinitions.Add(customPermissionsLevel);
    website.Update();
}
```

Managing Access to Resources


In order to write code that assigns correct permissions, you must understand two other concepts: permissions inheritance and anonymous access.

Using Permissions Inheritance

SharePoint base permissions propagate down the hierarchy of SharePoint objects. For example, you can assign the **AddListItems** base permission to a SharePoint site, even though list items cannot appear in the web site itself. This base permission is inherited by all the lists in the web site.

Therefore, by assigning this base permission at the site level, you ensure that a user or group can add list items throughout the site. Before you use the permission assignment coding techniques you saw in the previous topic, make sure you know what permissions are inherited from parent objects such as the SharePoint site.

- Permissions Inheritance
 - Breaking inheritance
 - Restoring inheritance
- Anonymous Access
 - Enabling anonymous users to access a site
 - Assigning permissions to anonymous users

 **Best Practice:** If you create many role assignments on multiple lists, libraries, items, and other securable object at different levels in the hierarchy, you can create a very complex permissions configuration. This can be difficult to administer because, when there is incorrect assignment, it can be difficult to diagnose which role assignment on which securable object has caused the problem. Keep permissions simple by using inheritance as much as possible and assigning a small number of at the highest level in the hierarchy. Also, carefully document the role assignments you create.

You can use the following properties and methods of any securable object to manage permissions inheritance:

- *SPSecurableObject.HasUniqueRoleAssignments*. This Boolean property is true when an object has role assignments in its own collection. If it is false then all permissions are inherited from an object higher in the hierarchy.
- *SPSecurableObject.BreakRoleInheritance()*. This method breaks the inheritance of permissions from parent objects. If you pass true to this method, the object starts with all the assignments it would have inherited from above. If you pass false, the object starts with no permissions and you must create a set of role assignments or no users will get access.
- *SPSecurableObject.ResetRoleInheritance()*. This method restores the inheritance of permissions from the parent object and removes all role assignments from the object's collection.

Enabling Anonymous Access

By default, SharePoint requires all users to log on before they can access SharePoint resources. However, sometimes you want to allow anonymous users partial access. For example, if you are using SharePoint to run an internet site and manage its content, anonymous users must be able to access at least some of your content, such as the home page, the "About Us" page, and your product catalog.

To enable anonymous access to a site, take the following steps:

1. In Central Administration, under **Application Management**, click **Manage web applications**.
2. Select the appropriate web application, and then click **Authentication Providers** on the ribbon.

3. Click the **Default** zone and then select the **Enable anonymous access** check box.
4. Click **Save**.
5. In the appropriate site collection, click **Site Settings**.
6. Under **Users and Permissions**, click **People and groups**.
7. On the ribbon, click **Anonymous Access**.
8. Select the objects you want to grant access to. You can choose **Entire web site**, **Lists and libraries**, or **Nothing**.
9. Click **OK**.

When the above procedure is complete, a new site group named **Anonymous Users** is created. By can grant **SPRoleAssignment** objects to this group just as you would for any other security principal. In this way, you can finely control what anonymous users can do within your SharePoint site.

Lab A: Managing Permissions Programmatically in SharePoint 2013

Scenario

Contoso plan to add a document library named Financials to every project site on the company intranet portal. Because this document library will contain sensitive financial data, you must restrict who can access the library. Only the site owners group of each project site, together with the members of the Managers security group, should be able to view documents in the Financials library

Objectives

After completing this lab, you will be able to:

- Create a feature receiver that programmatically edits list permissions.
- Write code that breaks permission inheritance in a SharePoint library or list.

Estimated Time: 15 minutes

- Virtual Machine: 20488B-LON-SP-06
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Managing List Permissions Programmatically

Scenario

A colleague has created a new SharePoint project in Visual Studio and added the Financials document library to the project. You have been asked to add code to this project that ensures that only site owners and members of the Managers group can access documents in the Financials library when the solution is deployed to any SharePoint site.

The main tasks for this exercise are as follows:

1. Add a Feature Receiver to the Solution
2. Add Code that Breaks Permissions Inheritance
3. Grant Permissions to Site Owners
4. Grant Permissions to Managers
5. Test the Financials Library Project

► Task 1: Add a Feature Receiver to the Solution

- Start the 20488B-LON-SP-06 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
- Open Visual Studio 2012 and open the following solution
E:\Labfiles\LabA\Starter\FinancialsLibrary\FinancialsLibrary.sln.
- Verify that the **FinancialsLibrary** project contains a library named **Financials**.
- Add a new event receiver to the **FinancialsLibrary** feature.

► Task 2: Add Code that Breaks Permissions Inheritance

- In the new event receiver, uncomment the **FeatureActivated** method.

- In the **FeatureActivated** method, create a new **SPWeb** object named **parentWeb**. Set the value of **parentWeb** to the value of **properties.Feature.Parent** and cast the value as a **SPWeb** object.
 - Create a new **SPDocumentLibrary** object named **financialsLibrary** and set the value of it to be the Financials list from the **parentWeb**.
 - Add an **if** code block that checks if the **financialsLibrary** object is not null and then cast the value as a **SPDocumentLibrary** object.
 - In the **if** statement you just created, write a line of code that breaks role inheritance for the **financialsLibrary** object. Do not copy role assignments.
 - Update the **financialsLibrary** object.
- ▶ **Task 3: Grant Permissions to Site Owners**
- Create a new **SPRoleAssignment** object named **ownersAssignment**. Pass the associated owner group property of the **parentWeb** object to the new object's constructor.
 - Add a new role definition binding to the **ownersAssignment**. Pass the **Full Control** role definition from the **parentWeb**.
 - Add the **ownersAssignment** to the **financialsLibrary** role assignments collection.
 - Update the **financialsLibrary** object.
- ▶ **Task 4: Grant Permissions to Managers**
- Add the **CONTOSO\Managers** AD DS security group to the **parentWeb.AllUsers** collection.
 - Create a new **SPRoleAssignment** object named **managersAssignment**. Pass the **CONTOSO\Managers** entry in the **parentWeb.AllUsers** collection to the new object's constructor.
 - Add a new role definition binding to the **managersAssignment**. Pass the **Contribute** role definition from the **parentWeb**.
 - Add the **managersAssignment** to the **financialsLibrary** role assignments collection.
 - Update the **financialsLibrary** object.
- ▶ **Task 5: Test the Financials Library Project**
- Start the solution without debugging and log on as **Administrator**.
 - Open the **Financials** document library.
 - Verify that the permissions for the library are set as you expect.
 - Close Internet Explorer and Visual Studio.

Results: When you have completed this exercise, you will have built a complete SharePoint solution that includes a feature receiver. The feature receiver code will set permissions level on the Financials library.

Question: In the Task 2, when you called the **BreakRoleInheritance** method, you passed the value **false**. What would happen if you passed the value **true** instead?

Question: In Task 4, you granted **Contribute** permissions to members of the **Managers** AD DS security group. What other method could you use to grant this permission when permission inheritance is enabled?

Lesson 3

Configuring Forms-Based Authentication

Forms Based Authentication (FBA) is a versatile and extendible authentication type to use in a SharePoint farm because it has a large range of providers that can store user accounts and roles in different locations. You can also create custom providers that enable SharePoint to authenticate users against a unique store of credentials you have in your organization. In this lesson, you will see how to configure FBA and create custom providers.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how SharePoint checks credentials with an FBA membership provider.
- Register new membership and role providers for use in SharePoint 2013.
- Create a custom membership provider and use it to authenticate users in SharePoint.
- Create a custom role provider and use it to group user accounts for SharePoint.

Forms-Based Authentication Overview

FBA is an ASP.NET authentication mechanism in which you build ASP.NET forms to collect authentication credentials from users. Because these forms are rendered as HTML pages, any web browser can authenticate by using this method. FBA also includes mechanisms by which users can reset their password, recover a forgotten password, and complete other common management tasks. You can also use FBA to implement roles – these are groups of user accounts much like AD DS security groups.

- Architecture
 - Membership Providers
 - Role Providers
 - Credential Stores
- Advantages and Disadvantages

Architecture

FBA has a modular architecture that enables you to store user accounts and roles in many different locations. An FBA implementation includes the following objects:

- *Membership providers.* SharePoint uses FBA membership providers as authentication providers. This means that SharePoint passes the credentials each user enters in the login form to a membership provider. Each membership provider works with one kind of credential store. For example, the SQL Server membership provider checks credentials against those stored in a Microsoft SQL Server database.
- *Role Providers.* SharePoint uses FBA role providers as a store of security groups. You can organize FBA user accounts into roles to reflect job titles, project responsibilities, or other shared properties. You can assign SharePoint permissions levels to these roles. Users receive permissions to access resources because they are members of a role that has been granted a permissions level.
- *User Account and Role Stores.* Each provider must check credentials and role membership against some kind of store. This store is often a database but you could also use an LDAP directory system, a flat file, or some other store of data. Usually the membership provider and role provider use the same store but it is possible to use a directory to store user accounts and a database to store roles.

FBA is a claims-based authentication type. As for other authentication types, the SharePoint Security Token Service (STS) creates a signed claims token when authentication is complete. The default claims include the identity of the user and the roles the user is a member of.

In FBA the HTTP protocol is used to transfer user credentials to the server without any encryption. You should use Secure Sockets Layer (SSL) to encrypt this communication and ensure that the credentials are protected from malicious third parties as they are transferred across the network. SSL also helps users to know that they are sending their credentials to the right website. For more information on configuring SSL for SharePoint 2013 see the following link:



Additional Reading: For more information on configuring SSL for SharePoint, see *Configure SSL for SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=311820>.

Advantages and Disadvantages

When you plan to use FBA, consider the following advantages:

- Because there are several default membership providers and extra membership providers available from third parties, you can store user accounts in a wide variety of locations when you use FBA.
- If you have user accounts in a store that is not supported by an existing membership provider, you can still use FBA by developing a custom membership provider for that store.
- Almost all browsers support FBA, because the forms it uses are rendered in standard HTML.
- You can create your own FBA logon pages which are branded and share a look and feel with your SharePoint site.

You should also consider the following limitations of FBA:

- In FBA, users must always type their credentials into the login page. Compare this with Windows Authentication: here, if the browser is Internet Explorer and the user has a Windows account, SharePoint can authenticate the user without prompting for credentials.
- You must ensure security by encrypting the login page with the SSL protocol. This requires a certificate preferably purchased from a trusted internet Certificate Authority.

Creating Custom Membership Providers

There are a wide range of membership providers included with ASP.NET, which you can make use of in SharePoint 2013. However sometimes you may find that these do not satisfy your requirements so you must create a custom membership provider. In the following example situations, you can choose to create a custom membership provider:

- You have user accounts in an unusual store, which you cannot access with any existing membership provider.
- You want to use an unusual combination of credentials, such as a username and password with a PIN number or other data.

To create a custom membership provider:

1. Inherit the **System.Web.Security.MembershipProvider** class
2. Override the following methods:
 - GetUser
 - FindUsersByEmail
 - FindUsersByName
 - GetAllUsers
 - ValidateUser

You can implement a custom membership provider by completing the following steps:

1. In Visual Studio 2013, create a new project by using the **SharePoint 2013 Project** template.
2. Add a new feature with the **Farm** scope.
3. Add a reference to the namespace **System.Web.ApplicationServices**.
4. Add a new class that inherits from the **System.Web.Security.MembershipProvider** class.
5. Override the **GetUser** methods. These methods obtain a **MembershipUser** object to represent the user account. In these method, query the credentials store for user metadata.
6. Override the **FindUsersByEmail** and **FindUsersByName** methods. These methods return a collection of user accounts with a specific email address or account name.
7. Override the **GetAllUsers** method. This method returns a collection of all the user accounts in the store.
8. Override the **ValidateUser** method. This is the method that compares the credentials passed by the user with those in the credential store. The method returns true if the credentials match.

The implementation of each method depends on the store your provider works with. For example, if the store of credentials is SQL Server database, you could use ADO.NET with LINQ to SQL to implement each method.

The following code illustrates how to implement a custom **MembershipProvider** class. The details of the communication with the credentials store are omitted because these depend on the nature of the store you are using.

A Custom Membership Provider

```
using System.Text;
using System.Threading.Tasks;
using System.Web.Security;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration.Claims;

namespace Contoso.ClaimsProvider
{
    public class Members : MembershipProvider
    {
        public override MembershipUser GetUser(string username, bool userIsOnline)
        {
            MembershipUser returnedUser = null;

            //Query for the user properties from the store here by using the name.

            //Use the queried properties to set the properties of the returnedUser object

            return returnedUser;
        }

        public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)
        {
            MembershipUser returnedUser = null;

            //Query for the user properties from the store here by using the provider unique key.

            //Use the queried properties to set the properties of the returnedUser object

            return returnedUser;
        }
    }
}
```

```
public override MembershipUserCollection FindUsersByEmail(string emailToMatch,
    int pageIndex, int pageSize, out int totalRecords)
{
    MembershipUserCollection users = new MembershipUserCollection();

    //Query for the user accounts from the store here by using the emailToMatch parameter
    //For each matching user account add a new MembershipUser object to the users collection

    return users;
}

public override MembershipUserCollection FindUsersByName(string usernameToMatch,
    int pageIndex, int pageSize, out int totalRecords)
{
    MembershipUserCollection users = new MembershipUserCollection();

    //Query for the user accounts from the store here by using the usernameToMatch parameter
    //For each matching user account add a new MembershipUser object to the users collection

    return users;
}

public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize,
    out int totalRecords)
{
    MembershipUserCollection allUsers = new MembershipUserCollection();
    totalRecords = 0;

    //Query for all the user accounts from the store here

    //For each user account add a new MembershipUser object to the allUsers collection
    //and increment the totalRecords parameter

    return allUsers;
}

public override bool ValidateUser(string username, string password)
{
    //Query for a user account here by using the username parameter.

    //If there is a matching user account, check the password parameter against the stored
password

    //If the passwords match return true - the user is validated
    //Otherwise return false - the user failed authentication

}
}
}
```

Creating Custom Role Providers

An FBA role provider is a component that looks up role membership for an authenticated user. FBA roles are groups of user accounts much like AD DS security groups. You can ease administration by assigning permission levels to roles instead of individually to each user. In this way, you can drastically reduce the number of permission assignments that are required.

As for membership providers, the wide range of role providers included with ASP.NET and created by third parties makes FBA roles very flexible. Most often, you can find an existing role provider that meets your needs. However, when you have unusual requirements, you can implement a custom role provider to store and retrieve roles from a custom store.

Custom role providers often match custom membership providers and can be implemented in the same Visual Studio project. If you build the two providers in a single project in this way, you can deploy them in a single farm solution and a single SharePoint feature. The following steps assume, instead, that you want a separate Visual Studio project, SharePoint solution, and SharePoint feature for your role provider.

To create a custom role provider, take the following steps:

1. In Visual Studio 2013, create a new project by using the **SharePoint 2013 Project** template.
2. Add a new feature with the **Farm** scope.
3. Add a reference to the namespace **System.Web.ApplicationServices**.
4. Add a new class that inherits from the **System.Web.Security.RoleProvider** class.
5. Override the **GetRolesForUser** method. This method returns all the roles that the requested username is a member of. The roles are returned as an array of strings.
6. Override the **RoleExists** method. This method returns true if a role with the requested name exists.

Again, the implementation of each method depends on the store your provider works with.

The following code illustrates how to implement a custom **RoleProvider** class. The details of the communication with the roles store are omitted because these depend on the nature of the store you are using.

A Custom Role Provider

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Security;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration.Claims;

namespace Contoso.ClaimsProvider
{
    public class RoleProvider : System.Web.Security.RoleProvider
    {
        public override string[] GetRolesForUser(string username)
        {
```

To create a custom role provider:

1. Inherit the **System.Web.Security.RoleProvider** class
2. Override the following methods:
 - GetRolesForUser
 - RoleExists

```

//Use this array to returns the role names as strings
string[] roles = new string[]{};

//Query the role store here for the roles that the username is a member of

//Add the name of each role to the roles array

return roles;
}

public override bool RoleExists(string roleName)
{
//Query the role store here for a role with the requested name

//If such a role exists return true

//Otherwise return false

}
}
}

```

Registering Providers

You can deploy your custom providers to a SharePoint farm in two ways:

- In the development environment, you can deploy the providers by starting the application in debugging mode within Visual Studio.
- In the staging or production environment, you can deploy the providers by uploading and deploying the SharePoint farm solution.

Regardless of the deployment method, there are two further tasks that you must complete before the provider can be used:

- Create a web application that is configured to use Form Authentication.
- Configure web.config files to use the new provider.

Configuring the Web Application

To create a web application that uses your new providers, complete the following steps:

1. In the Central Administration home page, under Application Management, click **Manage web applications**.
2. In the **Contribute** section of the ribbon, click **New**.
3. In the **Create New Web Application** dialog box, under **Claims Authentication Types**, select **Enable Forms Based Authentication (FBA)**.
4. In the **ASP.NET Membership provider name** box, type the name of your membership provider.
5. In the **ASP.NET Role manager name** box, type the name of your role provider.
6. At the bottom of the page click **OK**.

To use a custom membership provider or role provider:

- Deploy the provider
- Create a new web application and configure it to use the provider
- Configure the web.config files for:
 - The Central Administration site
 - The Secure Token Server
 - The new web application

7. When the new web application has been created, click **OK**.

Updating Web.config Files

In order to support the custom membership provider and custom role provider throughout your SharePoint farm, it is necessary to update three different web.config files:

- *The Central Administration web.config File.* You can edit this file by opening IIS Manager and exploring the **SharePoint Central Administration v4** website.
- *The Secure Token Service web.config File.* You can edit this file by opening IIS Manager and exploring the **SecurityTokenServiceApplication** website.
- *The New Web Application web.config File.* You can edit this file by opening IIS Manager and exploring the new web application's website.

The following code shows how to configure a web.config file to support a custom membership provider and a custom role provider.

Adding Custom Providers to Web.Config

```
<Configuration>
  <system.web>
    <membership>
      <providers>
        <add name="ContosoMembership"
          type="Contoso.ClaimsProvider.MembershipProvider, Contoso.ClaimsProvider,
Version=1.0.0.0,
          Culture=neutral, PublicKeyToken=yourkeyhere" />
      </providers>
    </membership>
    <roleManager enabled="true" >
      <providers>
        <add name="ContosoRoleManager"
          type="Contoso.ClaimsProvider.RoleProvider, Contoso.ClaimsProvider,
Version=1.0.0.0,
          Culture=neutral, PublicKeyToken=yourkeyhere" />
      </providers>
    </roleManager>
  </system.web>
</Configuration>
```

Creating a Custom Login Page

When you configure FBA, SharePoint automatically presents a login page to anonymous users when they attempt to access a secured object. This login page has text boxes in which the username and password credentials can be edited. There is also a "Remember Me" check box which the user can check to store their credentials in a cookie on the computer. The stored credentials are then used for subsequent logins.

The standard login page is often sufficient, but sometimes you may want to display a different authentication page. For example:

You may want to collect credentials other than a username and password.

To create a custom login page for FBA:

1. Create a new empty SharePoint project.
2. Add a new application page to the project.
3. Add references to **Microsoft.SharePoint.Security.dll** and **Microsoft.SharePoint.IdentityModel.dll**
4. Use the **SPClaimsUtility.AuthenticateFormsUser** method to log the user in.
5. Package and deploy the solution.
6. In Central Administration, configure a web application to use the new login page.

You may want to use a different layout than the default page.

You may want to use a page with custom branding or other design elements.

Creating an Authentication Page

To create a custom FBA login page for SharePoint 2013, you can create a new SharePoint project in Visual Studio 2012 and add an application page to it. Take the following steps:

1. In Visual Studio 2012, click **File**, click **New**, and then click **Project**.
2. In the list of templates, select **SharePoint 2013 – Empty Project**.
3. Select **Deploy as a farm solution**, and then click **Finish**.
4. In the Solution Explorer pane, right-click the SharePoint project, click **Add**, and then click **New Item**.
5. In the list of templates, select **Application Page**.
6. In the **Name** box, type a suitable file name, and then click **Add**.

You must also add new references to the **Microsoft.SharePoint.Security.dll** and the **Microsoft.SharePoint.IdentityModel.dll** files:

1. In the Solution Explorer pane, right-click **References** and then click **Add Reference**.
2. Click **Browse**, and then browse to the folder C:\Program Files\Common Files\microsoft shared\web server extensions\15\ISAPI.
3. Click **Microsoft.SharePoint.Security.dll** and then click **Add**.
4. Click **Browse**, and then browse to the folder C:\Windows\Microsoft.NET\assembly\GAC_MSIL\Microsoft.SharePoint.IdentityModel\
5. Click **Microsoft.SharePoint.IdentityModel.dll**, click **Add**, and then click **OK**.

After the Visual Studio project is set up with the correct references, you can authenticate user accounts in your code by using the **SPClaimUtility.AuthenticateFormsUser** method.

The following code shows how to authenticate a user in a custom SharePoint login page by using claims.

Authenticating a User

```
//Attempt to authenticate the user, passing the referring page, the username and the password
bool loginStatus = SPClaimsUtility.AuthenticateFormsUser(Context.Request.UrlReferrer,
    txtUsername.Text, txtPassword.Text);

if (loginStatus)
{
    //The authentication was successful. Redirect to the most appropriate page
    if (Context.Request.QueryString.Keys.Count > 1)
    {
        Response.Redirect(Context.Request.QueryString["Source"].ToString());
    }
    else
    {
        Response.Redirect(Context.Request.QueryString["ReturnUrl"].ToString());
    }
}
else
{
    //The authentication was unsuccessful
    lblMessage.Text = "The authentication was unsuccessful. " +
        "Did you enter the right username and password?";
}
```

Deploying and Configuring the Login Page

Because you have created the login page as a SharePoint project, Visual Studio will package it as a standard SharePoint farm solution file. Therefore a SharePoint administrator can deploy this solution to a SharePoint farm in the same way as any other solution. However, once the solution is deployed, you must configure the web application to use the new login page. To do this, complete the following steps:

1. In Central Administration, under **Application Management**, click **Manage web applications**.
2. In the list of web applications, select the web application that you want to configure.
3. On the ribbon, click **Authentication Providers**.
4. In the list of zones, click the zone that is enabled for FBA. If there is only one zone, click **Default**.
5. Scroll down to locate the **Sign In Page URL** section.
6. Select **Custom Sign In Page**, and then in the box, type `~/_layouts/yourloginpage.aspx` when *yourloginpage* is the name of the login ASP.NET page you created.

Discussion: Federation and Custom Provider Scenarios

Discuss the following scenario with the class and answer the questions posed:

Scenario – A Custom Credential Store

You are a SharePoint developer in a publishing organization. You have a successful SharePoint farm that you use to manage document authoring, editing, and reviews. User accounts are stored in AD DS.

Your company has acquired a smaller publishing house and you have been asked to investigate how you can enable users in that company to access the SharePoint farm and take full part in the content development process. User accounts in the acquired company are stored in a custom directory system that is not standards-compliant.

Scenario: A Custom Credential Store

Read the scenario and then discuss the following questions:

1. Can user accounts in the custom directory system be used to access SharePoint without migrating them into AD DS?
2. Can forms authentication be used to check credentials that are stored in the custom directory system?
3. Can federated SAML authentication be used to check credentials that are stored in the custom directory system?

Questions

A senior developer has asked you to write a report that addresses the following questions:

1. Can user accounts in the custom directory system be used to access SharePoint without migrating them into AD DS?
2. Can forms authentication be used to check credentials that are stored in the custom directory system?
3. Can federated SAML authentication be used to check credentials that are stored in the custom directory system?

Lesson 4

Customizing the Authentication Experience

In this lesson, you will see two methods with which you can customize the user login experience in SharePoint 2013. The first method is to create a custom claims provider – you can use this to add extra claims to a user's security token. SharePoint can use these claims to authorize access to resources or to show extra information about users in the people picker control. The second method is to build a custom login page. If you are using FBA to authenticate users, a custom login page enables you to collect custom information or present extra content to users as they authenticate.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how a custom claims providers can enrich the information that SharePoint can verify for a user identity.
- Use Visual Studio to create and code a custom claims provider.
- Deploy a custom claims provider to a SharePoint farm.
- Create a custom FBA authentication page.

What is a Claims Provider?

As you have seen, SharePoint 2013 uses claims authentication in almost all circumstances. In claims authentication, SharePoint trusts a claims provider to certify small pieces of information, known as claims, about each user. The SharePoint Secure Token Service (STS) packages these claims into each user's claims security token when authentication is complete. In SharePoint 2013, a user's claims may include:

- The username and other identity information, such as the full name.
- The AD DS security groups that the user is a member of.
- The ASP.NET roles that the user is a member of.
- Information from a federated authentication provider that is included in the SAML token.

SharePoint uses these claims for two purposes:

- *Authorization.* When an authenticated user attempts to access a SharePoint resource such as an item, list, or library, SharePoint checks the claims against the resource permissions to determine whether to permit the operation.
- *Display.* Throughout the SharePoint user interface, administrators and users must select from a list of their fellow users. For example, in a document management workflow, a user might need to select a user to send the document to for approval. SharePoint uses the People Picker control to present information that helps you to select the right user. The People Picker control can display information from each user's claims.

A claims provider is a component that formulates the claims that SharePoint incorporates into the user's security token at authentication.

SharePoint uses claims to:

- Authorize access to resources.
- Help users to pick from a list of their fellow users.

Many companies store additional information about users that is not in AD DS, the FBA credentials store, or a federated authentication provider. For example, a user's age may be stored in a human resources database. By default, such information cannot be used in SharePoint for authorization or display. If you want to enable SharePoint to use such extra information in these ways, you should consider creating a custom claims provider. A custom claims provider can query a data store outside of SharePoint for extra information about a user and include that information in the claims in the security token.

Creating a Claims Provider

Consider creating a custom claims provider if you want to enhance the information that the SharePoint STS incorporates into the user's claims security token at logon. For example, suppose your company does not store each user's age in AD DS but does store the age in the human resources database. You would like to restrict certain content in SharePoint to users who are over 21. To achieve this, you could create a custom claims provider that checks the user's age against the HR database and certifies that the user is either over or under 21. SharePoint could use these claims to deny access to users who are too young. Notice that the claim need not include the age of the user – only a Boolean value that confirms or denies that the user is old enough.

To create a custom claims provider, derive from **SPClaimProvider** and implement:

- `SupportsEntityInformation`
- `FillClaimsForEntity()`
- `FillSchema`
- `FillClaimTypes`
- `FillClaimValueTypes`
- `FillEntityTypes`

To create a custom claims provider, in Visual Studio create a class that derives from the **SPClaimProvider** class. This class is in the **Microsoft.SharePoint.Administration.Claims** namespace. You must implement the **Name** property for all custom claims providers.


If you want to add a claim for authorization, you must implement the following members of the **SPClaimProvider** base class:

- *SupportsEntityInformation*. Set this Boolean value to true to support authorization.
- *FillClaimsForEntity*. Use this method to add claims to the user token. Often, your code looks up data in an external store, such as an HR database, to determine what claims to add. For each claim, call the **CreateClaim** helper method.

If you want to add a claim for displaying information in the People Picker control, you must implement the following methods:

- *FillSchema*. Use this method to fix the metadata about a user that the claims provider returns to the People Picker control.
- *FillClaimTypes*. Use this method to fix the claim types that the claims provider returns to the People Picker control. For example, claim types may include classes that represent group membership, ages, or geographical locations.
- *FillClaimValueTypes*. Use this method to fix the format of the metadata that the provider returns the People Picker control. For example, if the schema include a value that certifies that the user is over 21, use this method to specify that this is Boolean value.
- *FillEntityTypes*. Use this method to fix the type of entities that this claim applies to. For example, you can specify that this claim applies to user accounts, AD DS security groups, or FBA roles.

You may also need to implement other methods to support full functionality in the People Picker control. For full details, see the following location:

 **Additional Reading:** For more information on how to create a custom claims provider in SharePoint 2013, see *How to: Create a claims provider in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=311821>.

The following code example, shows a simple custom claims provider. This simple example only supports claims augmentation and adds a new claim if the username is "SuperUser".

A Custom Claims Provider

```
public class SimpleClaimsProvider
{
    //Tell SharePoint that this provider supports augmenting claims in the security token
    public override bool SupportsEntityInformation
    {
        get { return true; }
    }

    //This method adds security claims
    protected override void FillClaimsForEntity(Uri context, SPClaim entity, List<SPClaim> claims)
    {
        //Check that a security entity has been supplied
        if (entity == null)
        {
            throw new ArgumentNullException("entity");
        }
        //Check that a list of claims has been supplied
        if (claims == null)
        {
            throw new ArgumentNullException("claims");
        }

        //Add the claim for the user account
        SPClaim userAccountClaim = SPClaimProviderManager.DecodeUserIdentifierClaim(entity);

        //If this account is CONTOSO\SuperUser add an extra claim.
        if (userAccountClaim.Value.ToUpper() == "CONTOSO\\SUPERUSER")
        {
            //Use the CreateClaim helper method to create a claim
            SPClaim superUserClaim = CreateClaim(Contoso.Claims.Role, Contoso.Claims.SuperUser,
                Microsoft.IdentityModel.Claims.ClaimValueTypes.String);

            //Add the new claim to the claims collection
            claims.Add(superUserClaim);
        }
    }
}
```

Deploying a Claims Provider

Use the SharePoint features infrastructure to deploy a completed claims provider to a SharePoint farm for staging or production. SharePoint 2013 includes a specialized base class, named **SPClaimProviderFeatureReceiver** that you can use for deployment. Make sure that your feature receiver derives from this class and not the stand **SPFeatureReceiver** base class. The **SPClaimProviderFeatureReceiver** includes the following properties, which you can use to describe your claim provider to SharePoint administrators:

To deploy a claims provider, create a feature receiver:

- Derive the feature receiver from the **SPClaimProviderFeatureReceiverClass**
- Override the following properties
 - ClaimProviderAssembly
 - ClaimProviderType
 - ClaimProviderDisplayName
 - ClaimProviderDescription

- *ClaimProviderAssembly*. Use this property to specify the full name of the assembly that contains the claims provider class.
- *ClaimProviderType*. Use this property to specify the class name of the claims provider.
- *ClaimProviderDisplayName*. Use this property to specify a user-friendly name for the claims provider.
- *ClaimProviderDescription*. Use this property to specify a description for the claims provider.

You can include the **SPClaimProviderFeatureReceiver** instance in a SharePoint solution with other features or you can create a dedicated solution file that only deploys the claim provider.

The following code shows an example of a feature receiver that deploys a claims provider

A Claims Provider Feature Receiver

```
using System;
using System.Security.Permissions;
using Microsoft.SharePoint.Administration;
using Microsoft.SharePoint.Administration.Claims;
using Microsoft.SharePoint.Security;

namespace Contoso.SharePoint.Security
{
    [SharePointPermission(SecurityAction.Demand, ObjectModel = true)]
    public sealed class AgeClaimProviderFeatureReceiver : SPClaimProviderFeatureReceiver
    {
        private void ExecBaseFeatureActivated(Microsoft.SharePoint.SPFeatureReceiverProperties
properties)
        {
            base.FeatureActivated(properties);
        }

        //This property returns the fully qualified name of the claim provider assembly.
        public override string ClaimProviderAssembly
        {
            get{ return typeof(AgeClaimProvider).Assembly.FullName; }
        }

        //This property specifies the claim provider class name
        public override string ClaimProviderType
        {
            get{ return typeof(AgeClaimProvider).FullName; }
        }

        //This property specifies the display name of the claim provider
    }
}
```

```

public override string ClaimProviderDisplayName
{
    get{ return "Contoso Age Claim Provider"; }
}

//This property specifies a description for the claims provider
public override string ClaimProviderDescription
{
    get
    {
        return "This claim provider adds age information to the user's security claims
token.";
    }
}

public override void FeatureActivated(Microsoft.SharePoint.SPFeatureReceiverProperties
properties)
{
    {
        ExecBaseFeatureActivated(properties);
    }
}
}
}

```

Demonstration: A Custom Claims Provider

In this demonstration, you will see when the following two methods execute in a Custom Claims Provider:

- FillClaimsForEntity()
- FillSearch()

You will code these methods in the lab.

Demonstration Steps

- Start the 20488B-LON-SP-06 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Windows Start screen, click **Computer**.
- Browse to E:\LabFiles\LabB\Solution\ContosoClaimsProvider folder.
- Double-click **ContosoClaimsProvider.sln**.
- In the **How do you want to open this type of file (.sln)?** dialog box, click **Visual Studio 2012**.
- In the Solution Explorer, double-click **ContosoClaimsProvider.cs**.
- In the **ContosoClaimsProvider.cs** code file, locate the following line of code:

```
protected override void FillClaimsForEntity(Uri context, SPClaimEntity entity, List<SPClaim>
claims)
```

- Right-click the located code, click **Breakpoint**, and then click **Insert breakpoint**.
- Locate the following line of code:

```
protected override void FillSearch(Uri context, string[] entityTypes, string searchPattern,
string hierarchyNodeID, int maxCount, SPPProviderHierarchyTree searchTree)
```


- Right-click the located code, click **Breakpoint**, and then click **Insert breakpoint**.
- In the Windows Start page, click **SharePoint 2013 Management Shell**.
- Type **IISReset** and then press Enter.
- In Visual Studio, on the **DEBUG** menu, click **Start Debugging**.
- If a **Debugging Not Enabled** dialog box appears, click **OK**.
- Before you log on to SharePoint, switch to Visual Studio.
- On the **DEBUG** menu, click **Attach to Process**.
- Select the **Show processes from all users** checkbox.
- In the **Available Processes** list, click the **w3wp.exe** process with the username **CONTOSO\SPFarm**.
- Click **Attach**, and then in the **Attach Security Warning** dialog, click **Attach**.
- Switch to Internet Explorer.
- In the **Windows Security** dialog, in the **User name** box, type **Administrator**.
- In the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
- Visual Studio interrupts execution in the **FillClaimsForEntity** method. Explain that this method executes whenever a user authenticates with SharePoint and adds claims to the user token.
- On the **DEBUG** menu, click **Continue**.
- In Internet Explorer, click the **Settings** icon, and then click **Site settings**.
- Under **Users and Permissions**, click **Site permissions**.
- Click **Contoso Development Site Visitors**.
- Click **New**.
- In the **Add people to the Contoso Development Site Visitors group** box, type **North**.
- Visual Studio interrupts execution in the **FillSearch** method. Explain that this method executes whenever a user searches for a user or group in the People Picker control.
- On the **DEBUG** menu, click **Delete All Breakpoints**.
- In the **Microsoft Visual Studio** dialog box, click **Yes**.
- On the **DEBUG** menu, click **Continue**.
- Click **North America** and then click **Share**.
- Close Internet Explorer.
- Close Visual Studio.

Lab B: Creating and Deploying a Custom Claims Provider

Scenario

The research team at Contoso is working on some highly confidential research. The team wants to be able to restrict access to information based on where a user logs on. Your task is create a custom claims provider that augments the user's claim token with location information.

Objectives

After completing this lab, you will be able to:

- Create a SharePoint claims provider that augments the claims in a user's security token.
- Create a claims provider that supports search and resolve in the SharePoint people picker dialog.
- Create a feature receiver to deploy a claims provider to any SharePoint farm.

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-06
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Creating a Custom Claims Provider

Scenario

In this exercise, you will create a custom claims provider and implement the code required to support claims augmentation.

The main tasks for this exercise are as follows:

1. Create a New SharePoint Project in Visual Studio
2. Add a Claims Provider Class to the Project
3. Create Internal Properties for the Claims Provider
4. Implement Claims Augmentation

► Task 1: Create a New SharePoint Project in Visual Studio

- Start the 20488B-LON-SP-06 virtual machine.
- Log on to the **LONDON** machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
- Create a new Visual Studio project for your app. Use the following information:
 - Language: Visual C#
 - Template: SharePoint 2013 - Empty Project
 - Name: ContosoClaimsProvider
 - Location: E:\Labfiles\LabB\Starter
- Use the **http://dev.contoso.com** SharePoint site for debugging and deploy the project as a farm solution.
- Add a reference to the **Microsoft.IdentityModel** assembly.
- Add a reference to the **Microsoft.SharePoint.Security** assembly.

► **Task 2: Add a Claims Provider Class to the Project**

- Add the following code file to the **ContosoClaimsProvider** project:
 - E:\LabFiles\LabB\C# Code\ContosoClaimsProvider.cs
- Ensure that the **LocationClaimsProvider** class derives from the **SPClaimProvider** base class.
- Save your changes.

► **Task 3: Create Internal Properties for the Claims Provider**

- In the **ContosoClaimsProvider.cs** class file, replace the "`///
Add internal and private properties here`" comment, with a new internal, static, read only, string property named **ProviderDisplayName**. Return the string **Location Claims Provider** for this property.
- Add a new internal, static, read only, string property named **ProviderInternalName**. Return the string **LocationClaimsProvider** for this property.
- Add a new private, static, read only, string property named **LocationClaimType**. Return the string **http://schema.contoso.com/location** for this property.
- Add a new private, static, read only string property named **LocationClaimValueType**. Return the **Microsoft.IdentityModel.Claims.ClaimValueType.String** value for this property.
- Save your changes.

► **Task 4: Implement Claims Augmentation**

- Remove the existing code from the **FillClaimsForEntity** method. Instead add code that throws an **ArgumentNullException** if the **entity** parameter is null.
- Add code that throws an **ArgumentNullException** if the **claims** parameter is null.
- Determine the user's location by passing the **entity** parameter to the **getLocation** method. Store the result in a new **string** variable named **currentLocation**.
- Create a new **SPClaim** object named **newClaim**. Set the object's value to the result of the **CreateClaim** method, passing the following parameters:
 - Claim Type: **LocationClaimType**
 - Value: **currentLocation**
 - Value Type: **LocationClaimValueType**
- Add the **newClaim** object to the **claims** collection.
- Ensure that the **SupportsEntityInformation** property returns **true**.
- Save your changes.

Results: A claims provider that can add claims to the user's security token based on their location.

Exercise 2: Supporting Search and Resolve in a Claims Provider

Scenario

In this exercise, you will add the code necessary to support People Picker functionality, including search and resolve operations.

The main tasks for this exercise are as follows:

1. Implement Search Functionality
2. Implement Resolve Functionality

► Task 1: Implement Search Functionality

- Remove the existing code from the **getPickerEntity** method. Instead add code that calls the **CreatePickerEntity** method and stores the result in a new **PickerEntity** object named **newEntity**.
- To set the **newEntity.Claim** value, call the **CreateClaim** method with the following values:
 - Claim Type: **LocationClaimType**
 - Value: **ClaimValue**
 - Value Type: **LocationClaimValueType**
- To set the **newEntity.Description** value, concatenate **ProviderDisplayName**, a colon, and **ClaimValue**.
- Use the **ClaimValue** parameter to set the **newEntity.DisplayText** value.
- Use the **ClaimValue** parameter to set the **newEntity.EntityData** value with the data key **DisplayName**.
- Set the **newEntity.EntityType** value to **SPClaimEntityTypes.FormsRole**.
- Mark the **newEntity** as resolved.
- Set the **newEntity.EntityGroupName** value to the string **Location**.
- Return the **newEntity** object.
- In the **FillSearch** method, if the **entityTypes** parameter does not contain a **SPClaimEntityTypes.FormsRole** return nothing.
- Create a new integer variable named **locationNode** and set it to equal **-1**.
- Create a new **SPProviderHierarchyNode** object named **matchesNode** and set its value to **null**.
- Create a **foreach** loop that loops through all the strings in the **possibleLocations** array.
- Within the **foreach** loop, increment the **locationNode** variable.
- Create an **if** statement that executes if the **location** value in lower case starts with the **searchPattern** value in lower case.
- Pass the location value to the **getPickerEntity** method. Store the returned **PickerEntity** object in a variable named **newEntity**.
- Create an **if** statement that executes if the **searchTree** does not have a child node with key **locationKeys[locationNode]**. Include an **else** code block.
- In the new **if** statement, set the **matchesNode** object to a new **SPProviderHierarchyNode** object. Pass the following parameters to the constructor:
 - Provider Name: **ProviderInternalName**

- Name: **locationLabels[locationNode]**
- Hierarchy Node ID: **locationKeys[locationNode]**
- Is Leaf: **true**
- Add the **matchesNode** object to the **searchTree**.
- In the **else** code block, use the **Where** method to locate a member of the **searchTree.Children** collection that has a **HierarchyNodeID** that matches **locationKeys[locationNode]**
- After the end of the **else** code block, add **newEntity** to the **matchesNode** collection.
- Ensure that the **SupportsSearch** property returns **true**.
- Save your changes.
- ▶ **Task 2: Implement Resolve Functionality**
- Locate the **FillResolve** method which currently throws a **NotImplementedException** exception.
- In the **FillResolve** method, if the **entityTypes** parameter does not contain a **SPClaimEntityTypes.FormsRole**, return nothing.
- Add a **foreach** code block that loops through all the strings in the **possibleLocations** array.
- Within the **foreach** code block, create an if statement that executes if the **location** value in lower case matches the **resolveInput** value in lower case.
- Within the **if** code block, pass the **location** string to the **getPickerEntity** method. Store the result in a new **PickerEntity** object named **newEntity**.
- Add the **newEntity** object to the **resolved** collection.
- Ensure that the **SupportsResolve** property returns **true**.
- Save your changes.

Results: A claims provider that can respond to user searches in the People Picker control.

Exercise 3: Deploying and Testing a Claims Provider

Scenario

In this exercise, you will create a feature and feature receiver that can deploy the Contoso location claims provider. You will also test the claims provider in the development site.

The main tasks for this exercise are as follows:

1. Create a Feature for the Solution
2. Create a Feature Receiver to Deploy the Claims Provider
3. Test the Solution

▶ Task 1: Create a Feature for the Solution

- Add a new feature named **ContosoClaimProviderFeature** to the project.
- Set the display name for the feature to **Contoso Location Claim Provider** and the scope to **Farm**.
- Save your changes.

► Task 2: Create a Feature Receiver to Deploy the Claims Provider

- Add an event receiver to **ContosoClaimProviderFeature**.
- Add a **using** statement for the **Microsoft.SharePoint.Administration.Claims** namespace.
- Add an attribute to the **ContosoClaimProviderFeatureEventReceiver** class to grant permission for the **Demand** action.
- Ensure that the **ContosoClaimProviderFeatureEventReceiver** class derives from the **SPClaimProviderFeatureReceiver** base class.
- Override the **ClaimProviderAssembly** property. Return the full name of the assembly that contains the **LocationClaimsProvider** class.
- Override the **ClaimProviderDescription** property. Return the string **A claim provider that certifies the user's location**.
- Override the **ClaimProviderDisplayName** property. Return the **ProviderDisplayName** property of the **LocationClaimsProvider** class.
- Override the **ClaimProviderType** property. Return the full name of the type of the **LocationClaimsProvider** class.
- Create a new method. Use the following information:
 - Scope: **private**
 - Return type: **void**
 - Name: **ExecBaseFeatureActivated**
 - Parameter: an **SPFeatureReceiverProperties** object named **properties**
- In the **ExecBaseFeatureActivated** method, pass the **properties** parameter to the **base.FeatureActivated** method.
- Create a new **override** method. Use the following information:
 - Scope: **public**
 - Return type: **void**
 - Name: **FeatureActivated**
 - Parameter: an **SPFeatureReceiverProperties** object named **properties**
- In the **FeatureActivated** method, call the **ExecBaseFeatureActivated** method and pass the **properties** object.
- Save your changes.

► Task 3: Test the Solution

- Start the solution without debugging mode and log on as **Administrator**.
- Add users in **Europe** to the **Site Visitors** group. The People Picker dialog searches for claims as soon as you enter three or more characters.
- Stop debugging.
- Start the solution in debugging mode. Before you log on to the site, attach the debugger to the **w3wp.exe** process for the **CONTOSO\SPFarm** account.
- Insert a breakpoint in the **FillClaimsForEntity** method on the line that calls **claims.Add()**.

- Return to Internet Explorer and log on with the following credentials:
 - User name: **CONTOSO\Alexei**
 - Password: **Pa\$\$w0rd**
- When Visual Studio interrupts execution at your breakpoint, check the value of the **currentLocation** variable, and then continue execution. SharePoint displays the site, because Alexei Eremenko is located in Europe.
- Stop debugging and close Visual Studio.

Results: A completed SharePoint claims provider project.

Question: Why can you not use a class that derives from **SPFeatureReceiver** to deploy a claims provider?

Question: You want to create a claims provider that augments claims in the user's security token but does not show up in the People Picker dialog. Which methods should you implement in the **SPClaimProvider** class?

Module Review and Takeaways

In this module, you have learned how to manage authentication and authorization in SharePoint. You have also seen how to configure FBA and create custom membership providers and role providers that work in a SharePoint farm. You also learned about SharePoint claims providers and implemented a custom claims provider.



Best Practice: By using custom claim providers to restrict access to resources, you can secure your SharePoint farm and comply with legislation in your legal jurisdiction without using or publishing confidential information about your users.

Review Question(s)

Question: You are writing a SharePoint farm solution that must reassign permissions for the **Financials** library. The farm solution is deployed under the security context of your personal user account. You find that the solution is prevented from reassigning the permissions required. How can you ensure that the solution can always overcome these restrictions?

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false: To enable SharePoint to authenticate user credentials against a custom user store, you must create a custom FBA role provider.	

Module 7

Introducing Apps for SharePoint

Contents:

Module Overview	7-1
Lesson 1: Overview of Apps for SharePoint	7-2
Lesson 2: Developing Apps for SharePoint	7-16
Lab: Creating a Site Suggestions App	7-26
Module Review and Takeaways	7-30

Module Overview

SharePoint 2013 introduces apps for SharePoint, a new way to customize SharePoint functionality. Like farm solutions and sandboxed solutions, apps for SharePoint enable you to develop custom applications that use SharePoint functionality such as lists, document libraries, workflows, and so on. Unlike farm solutions and sandboxed solutions, apps for SharePoint run entirely outside any SharePoint processes and share no memory with SharePoint itself or with other apps. Instead, apps for SharePoint run either in the user's browser or on a web server such as Internet Information Services (IIS) or Windows Azure. This isolated architecture enables apps for SharePoint to run in the same way whether on-premises or in the cloud. It also means that an app is less dependent on the version of SharePoint that your organization runs.

Objectives

After completing this module, you will be able to:

- Describe apps for SharePoint and compare them to SharePoint farm solutions and sandboxed solutions.
- Describe how to develop apps for SharePoint 2013 that work on-premises and in the cloud.

Lesson 1

Overview of Apps for SharePoint

The SharePoint 2013 app model provides a more flexible way to build custom functionality that runs in SharePoint organizations both on-premises and in Office 365. However, developing apps for SharePoint is very different from developing farm solutions and sandboxed solutions. You must understand how to construct apps for SharePoint that support multiple tenants, how to provide starting points for the user interface, how to access data in SharePoint, and many other technical challenges. In this lesson, you will see a broad overview of the app model. Later lessons and modules go into all these subjects in greater depth.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the key features of apps for SharePoint and contrast them with farm solutions and sandboxed solutions.
- Describe how to host a SharePoint app within SharePoint or in a cloud location.
- Describe how to provision auto-hosted and provider-hosted apps.
- Choose a language and a tool set with which to build a SharePoint app.
- Describe how apps communicate with their host SharePoint sites.
- Choose whether to implement start pages, app parts, or custom user interface (UI) actions in a SharePoint app.
- Describe how to access data stored in SharePoint and external stores from a SharePoint app.
- Choose a location to publish a completed SharePoint app.

What is a SharePoint App?

A SharePoint app is a custom solution for SharePoint. Like farm solutions and sandboxed solutions, a SharePoint app can access and modify resources in SharePoint such as lists, documents, libraries, individual items, and other content. However, apps for SharePoint differ from farm solutions and sandboxed solutions in that their code runs entirely outside any SharePoint server processes; instead, a SharePoint app runs either on the user's browser or on a web server outside the SharePoint farm.

- A SharePoint app is a custom solution for SharePoint that runs entirely outside any SharePoint server processes
- App model design goals:
 - Apps must be supported both on-premises and in the cloud
 - App code never runs within the SharePoint host environment
 - App code accesses SharePoint data through web services
 - App code is authenticated and runs under a distinct identity
 - Apps are published in app catalogs and Office Store

SharePoint App model design goals

The SharePoint app model has been designed to provide developers with an easy way to develop custom applications based on SharePoint that will run smoothly on both SharePoint on-premises installations and in the cloud. The app model also removes an app's dependence on the version of SharePoint that you run, improves security, and makes it easy for users and administrators to locate apps that can help them. The app model was designed to address the following goals:

- Apps must be supported both on-premises and in the cloud. Office 365 is gaining in popularity, but many organizations will prefer on-premises farms for the foreseeable future. Developers need to know that their solutions can be used in both these contexts.
- App code never runs within the SharePoint host environment. This improves stability, because an app cannot interfere with SharePoint processes or memory. For example, an app cannot hang SharePoint processes or affect other apps that run in the sandbox.
- App code accesses SharePoint data through web services. The SharePoint web services insulate the app from any SharePoint version-specific requirements, so an app can run against SharePoint 2013 or any later version. In the past, SharePoint solutions have prevented companies from upgrading SharePoint because the solutions had to be rewritten for the new SharePoint version. These types of delays will not be caused by apps for SharePoint.
- App code is authenticated and runs under a distinct identity. This means that you can assign permissions specifically to an app. Such assignments are not possible for sandboxed solutions, which run under the identity of the current user. Such assignments are also not possible for farm solutions, which can either run as the current user or obtain elevated permissions.
- Apps are published in app catalogs and the Office Store. You can maintain an app catalog in each on-premises web application or in your Office 365 tenancy. Your users can browse this catalog to find apps that suit their needs. Alternatively, users can obtain apps for SharePoint from third parties by searching the Office Store.

Apps for SharePoint and Office apps

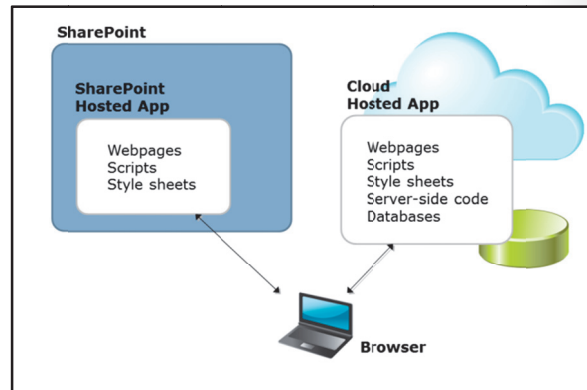
Both apps for SharePoint and Office apps are types of custom applications that use web technologies to extend the functionality and user interface available in SharePoint and Office applications. However, they are separate technologies and, as a developer, you must not confuse them. The following table highlights differences and similarities between apps for SharePoint and Office apps.

Aspect	Apps for SharePoint	Office apps
Entry point	Users access apps for SharePoint from SharePoint sites in the browser, either through a start page, a custom menu shortcut, or a control within a SharePoint page.	Users access Office apps from within an Office desktop application or from within an Office Online web version of an Office application. Office applications appear within task panes, within Excel spreadsheets, or at the top of Outlook email messages.
Interaction with the current Office document	Because apps for SharePoint do not run in the context of a single Office document, you cannot modify the current document.	Office apps can read information from and write information to the current document or email message.
Interaction with SharePoint content	You can use SharePoint web services to access SharePoint content from a SharePoint app.	You can use the same SharePoint web services to access SharePoint content from an Office app.
Client software	Apps for SharePoint can be accessed through most web browsers.	Office apps can only be accessed through Office 2013 applications or later versions of Office.

Hosting Apps for SharePoint

There are two important patterns that you can use when you design your apps for SharePoint. The pattern you choose depends on where you want to store code and where you want code to execute:

- *SharePoint Hosted Apps.* In this pattern, all of the objects created by developers are stored within the SharePoint content database. All code in a SharePoint-hosted app must be script that executes in the browser.
- *Cloud Hosted Apps.* In this pattern, all of the objects created by developers are stored outside SharePoint on another web server. This could be an IIS server, another kind of on-premises web server, or a cloud solution such as Windows Azure. In cloud-hosted apps, you can use a combination of client-side scripts and server-side code, such as ASP.NET Web Forms.



Choosing SharePoint Hosted Apps

When you build a SharePoint-hosted app, you store all the app's files within a SharePoint site. This site is dedicated to the app and is called the app web. Because you cannot execute any code on the SharePoint server, you can only use client-side code that executes in the user's browser. You can use the following client-side technologies to build your application:

- *HTML pages.* As the basic format that browsers display, you build the user interface of your app with HTML pages. If you know users have up-to-date browsers, you can consider using HTML5 tags, such as `<audio>` and `<canvas>` to provide rich content without requiring browser plug-ins.
- *Style sheets.* Websites use Cascading Style Sheets (CSS) to create a flexible and consistent look and feel. Style sheets can be used in precisely the same way within SharePoint-hosted apps to impose fonts, colors, and layouts on all pages.
- *JavaScript files.* Because SharePoint-hosted apps run code exclusively on the browser, you must use a scripting language that is supported by all your users' browsers for all business logic, validation, interactivity, and so on. JavaScript is the scripting language with the widest support in different browsers. JavaScript also has a wide range of script libraries available.
- *Script libraries.* A script library is a JavaScript file that contains a range of functions you can call from your own code. Script libraries make common tasks, such as locating page elements and modifying their content or style, very easy. They also take account of differences in the way different browsers interpret JavaScript. By using one or more script libraries, you can save developers a lot of time and effort. jQuery is a popular script library and is broadly used with Microsoft web development technologies. The SharePoint Client-Side Object Model (CSOM) is a JavaScript library designed specifically for accessing SharePoint content.
- *Rich web application environments.* You can use rich web technologies such as Adobe Flash and Microsoft Silverlight to create a compelling user interface for your SharePoint app. However, these technologies require extra plug-ins to be installed in the user's browser. The HTML5 `<canvas>` tag provides an alternative rich environment that does not require a plug-in for the latest browsers.

SharePoint-hosted apps usually require access to SharePoint content such as items or documents. You can use the CSOM to read and update SharePoint content from JavaScript. In SharePoint 2013, the CSOM has been enhanced with extra functions so that you can achieve almost all content access tasks from JavaScript code.

If you have no external server to host server-side code, and if you can create all the required functionality with client-side code and script libraries, you should choose to create a SharePoint-hosted app.

Choosing Cloud Hosted Apps

Sometimes you cannot build all the required functionality for your app by using client-side technologies. For example, suppose you want to build a knowledge base of tips and tricks. You first want to publish the app in the Office Store, and then for all customers who purchase the app, you want them to be able to add to a centralized list of tips and tricks stored in a Windows Azure SQL Database. You cannot build this as a SharePoint-hosted app because the centralized database cannot be stored in individual SharePoint content databases. Furthermore, you must build server-side components such as a web user interface or a web service, to publish the database to users.

When you cannot build an app as client-side code hosted within a SharePoint site, you cannot build a SharePoint-hosted app, and you must build a cloud-hosted app instead. You may also choose to build a cloud-hosted app if your team is more skilled in website development than in SharePoint development. In a cloud-hosted app, a web server hosts all the required files, including HTML pages, style sheets, JavaScript files, and JavaScript libraries. In addition, because the web server is outside the SharePoint installation, you can run server-side code, such as ASP.NET Web Forms, ASP.NET MVC, and PHP. Any web server can host the cloud app, but IIS is a popular choice because it is a Microsoft technology. Cloud-hosted apps can be hosted by on-premises web servers, but developers often use a cloud solution, such as Windows Azure, instead.

Although a cloud-hosted app consists of webpages and other files hosted on a web server outside SharePoint, you should not consider a cloud-hosted app to be simply an external website presented within a SharePoint site. Cloud-hosted apps have the following capabilities that are not available in websites:

- SharePoint can identify and authorize access to a cloud-hosted app. A website calling SharePoint services would not have this unique security context.
- A cloud-hosted app can integrate closely with the SharePoint user interface. For example, you can show a cloud-hosted app within a SharePoint dialog box or when users click a menu command.

Autohosted and Provider Hosted Apps

Cloud-hosted apps can include both client-side and server-side components, which are stored on a web server outside SharePoint. There are two types of cloud-hosted apps: autohosted and provider-hosted apps. The type of cloud-hosted app that you choose depends on how you want to isolate data from different tenants.

For example, consider these scenarios:

- You want to provide access to a single database of technical information about your products for your partner organizations. Because many partners use SharePoint, you

- The need for tenant isolation
- Building and provisioning a provider-hosted app
- Building and provisioning an autohosted app

want to provide a SharePoint app that publishes a single centralized database. You want all partners to access the same database so that changes need only be applied once.

- You want to build a Customer Relationship Management (CRM) app and publish it in the Office Store. Each customer will be able to create and manage their own database of leads and customers.

Provider-hosted apps

A provider-hosted app is a cloud app in which the developers take charge of isolating data from different tenants. In the first scenario, every partner shares the same product database because there is no need to isolate each tenant's data. By implementing a provider-hosted app, you can share data between all clients in this way.

In many real scenarios, you may want to share some tables in your database between all tenants, whereas other tables should be separated between tenants. Provider-hosted apps can implement such complex isolation because the developers write authorization code. This is more flexible, but it requires extra development time.

If you build a provider-hosted app, you must address several infrastructure requirements:

- You must provide a web server and ensure high availability.
- You must deploy the web files to the web server.
- You must create and populate any required databases.

Autohosted apps

If you have simple isolation requirements in which each tenant must be entirely separate from all others, you can create an autohosted app. Autohosted apps can only be built by using Windows Azure to host the web files. When a tenant installs your autohosted web app, Windows Azure automatically creates a new website, and often a dedicated database for that tenant.

Autohosted apps automatically ensure complete isolation for each tenant and reduce the administrative effort required for new tenants because websites and databases are automatically created. Development costs may also be lower because developers need not write isolation code or authorize all operations.

Developer Tools and Technologies

Because apps for SharePoint are based on common web technologies, you can choose from a wide range of languages and tools when you build them. The approach you choose will depend on the skillset of your team of developers as well as the hosting architecture you have selected. Microsoft provides a rich set of tools to ease the development process.

Client-side technologies

You can use any scripting language supported by all of your users' browsers to code client-side portions of apps for SharePoint. However, JavaScript is supported by a much wider range of browsers than any other scripting language. Furthermore, most script libraries, including jQuery, are only available in JavaScript. For these reasons, you should always consider developing client-side code in JavaScript for apps for SharePoint.

- Client-side technologies
 - Scripting language
 - Script libraries
- Server-side technologies
 - ASP.NET
 - Webpages
 - Web Forms
 - MVC
 - PHP
 - Ruby on Rails
- Development Environments
 - Visual Studio 2012
 - Napa

It is important to note that there are slight differences in how each browser interprets JavaScript, and these differences can add a significant extra load to developers. To ease development and handle browser differences, use a script library such as the following:

- *jQuery*. jQuery is one of the most widely used JavaScript libraries and is well supported in Visual Studio. Some Microsoft script libraries require jQuery, such as the ASP.NET unobtrusive AJAX library.
- *Prototype*. Prototype is a JavaScript library used by approximately four percent of all Internet sites, which makes it one of the most popular libraries.
- *jsPHP*. jsPHP is a JavaScript library that makes the PHP API functions available on the browser for client-side code. If your team of developers has good PHP skills, you may consider using jsPHP to take advantage of their knowledge with scripting.
- *Enyo*. The Enyo library can be used to develop code that runs in a browser, on mobile devices, and on desktop computers.

Server-side technologies

You cannot use server-side technologies if you choose to develop a SharePoint-hosted app. However, if you have decided to create a cloud-hosted app, you must choose a server-side execution environment and a language in which to code.

Server-side execution environments include:

- *ASP.NET*. This is Microsoft's server-side execution environment and is broadly used for websites. ASP.NET websites must be hosted on IIS or Windows Azure. Unlike other server-side execution environments, ASP.NET gives developers a choice of three programming models:
 - *Web Pages*. This simple model enables developers to embed C# or Visual Basic code within HTML pages. This code is executed on the server before the page is sent to the browser.
 - *Web Forms*. This more advanced model is designed to make webpage development more like desktop development. Developers can drag controls from a tool box onto a webpage and code against events from those controls such as click events. SharePoint itself is built from ASP.NET Web Forms.
 - *MVC*. This model separates code into model classes, which model data, controllers, which intercept user actions, and views, which implement a user interface.
- *PHP*. This is a general purpose scripting language, but it is most widely used to run scripts on a web server before a page is served to a browser. PHP websites can be hosted on almost all web servers including IIS and Apache.
- *Ruby on Rails*. This server-side execution environment uses the same MVC programming model that you can choose when you develop an ASP.NET website. You must use the Ruby language in a Ruby on Rails website.

You can develop server-side portions of a cloud-hosted SharePoint app using any of these execution environments. However, ASP.NET is most commonly used environment because of its deep integration in Visual Studio, its choice of language, and its choice of programming model.

Visual Studio

Visual Studio is Microsoft's premier Integrated Development Environment (IDE). For SharePoint developers, Visual Studio 2012 provides a great deal of help when you develop apps for SharePoint. For example:

- Visual Studio includes project templates for SharePoint-hosted apps, provider-hosted apps, and autohosted apps.

- Visual Studio includes item templates for apps for SharePoint such as list items, documents libraries, start pages, application manifest files, and other content.
- Visual Studio can package and publish a SharePoint-hosted app to a SharePoint farm or the Office Store.
- Visual Studio can package and publish cloud-hosted apps to IIS web servers or Windows Azure.
- Visual Studio has advanced debugging tools, such as IntelliTrace, that can accelerate debugging.
- Visual Studio can support Test Driven Development (TDD). This support is most often used in MVC applications to intercept and remove bugs throughout the development process by ensuring that critical unit tests are repeatedly passed.

Napa

Microsoft also provides the Napa tool as a free alternative to Visual Studio. You can use Napa to develop Office apps or apps for SharePoint. All development takes place within the browser and an Office 365 developer site. This approach means that you do not need to install a dedicated IDE or any other software on your computer. It also makes it easy to develop apps from non-Microsoft operating systems and mobile devices.



Additional Reading: For more information about Napa, see *Create apps for Office and SharePoint by using "Napa" Office 365 Development Tools* at <http://go.microsoft.com/fwlink/?LinkID=307093>.



Note: In this course, JavaScript, jQuery, C#, ASP.NET MVC and Visual Studio will be used in all SharePoint app labs, examples, and demonstrations.

Host Webs, App Webs, and Remote Webs

When users install a SharePoint app, they install the app into a specific SharePoint site. This site is known as the host web. If the app is a SharePoint-hosted app, a subsite of the host web is created; this subsite is called the app web. All the webpages, script files, style sheets, lists, document libraries, and other resources that the SharePoint-hosted app needs are installed into this subsite. If a user uninstalls the app, the app web is deleted so that all app files are removed cleanly. No files or modifications remain in the host web.

If the SharePoint app is a cloud-hosted app, there is no app web. Instead, all of the webpages, code files, and other resources are stored in a website on the external web server. This website is known as the remote web.

- Host webs
- App webs
- Remote webs
- App installation scope
- App web domains

`http://contosotenant-aa46c3ffd61233.apps.contoso.com`

Tenancy

APPUIID

App Web Hosting Domain

SharePoint tenancies and app installation scope

SharePoint 2013 is a multi-tenant application; in other words, it is designed for hosted environments in which multiple customers subscribe to SharePoint services from a service provider. In such environments, there is one SharePoint farm, but each customer must be isolated from the others to ensure that no sensitive data is inadvertently published to unauthorized users from other customers. To achieve this isolation, SharePoint supports tenancies. Each customer has a single tenancy and cannot see data or documents stored in other tenancies.

If you run an on-premises SharePoint farm exclusively for your own organization, you use a single tenancy for the entire organization. If you subscribe to Office 365, you receive a single tenancy.

Tenancies are important for apps, because you can install an app into the following scopes:

- *Site scope.* When you install an app into a site, there is a single host web. The app web is created as a subsite of the host web. The app is available only from within that host web. If you want to use the app in another host web, you must install it a second time. Each installation of the app has a separate security identity and access permissions.
- *Tenancy scope.* When you install an app in the tenancy scope, a single installation of that app can be accessed from any site in the tenancy. In this case, the app web is created within a special SharePoint site called the app catalog site. There is one app catalog site for each tenancy. When you use tenancy scope, there may be multiple host webs all using the same app web. Because there is only one installation, the app has the same security identity, no matter which host web the user is connected to.

App web domains

SharePoint creates a separate DNS domain for each app installation. This behavior may be counterintuitive, so you must be careful to understand it fully. For example, consider a host web within SharePoint that users can access through the following URL:

<http://intranet.contoso.com/>

You create a new SharePoint hosted app with a start page named "default.cshtml" in the "pages" list. You install the app in the preceding host web and name it "exampleapp." You might expect the app home page to be found at the following URL:

<http://intranet.contoso.com/exampleapp/pages/default.cshtml>

However, this is not the URL where the app's home page is located because SharePoint creates and uses a new unique domain each time it installs a new instance of a SharePoint hosted app. The correct URL looks like the following:

<http://contosotenant-aa46c3ffd61233.apps.contoso.com/exampleapp/pages/default.cshtml>

SharePoint creates unique domains for each app in this way for two security reasons:

- To separate app webs from host webs. By placing these in separate domains, cross-site scripting techniques cannot be used to circumvent security restrictions from app web to host web.
- To ensure that calls to SharePoint web services can be identified as originating from an app installation. This helps to ensure that permissions applied to an app installation cannot be circumvented.

Notice that the above SharePoint app URL includes the following parts:

- *contosotenant.* The first part of the unique domain is the tenancy in which the app is installed.
- *aa46c3ffd61233.* The second part of the unique domain is a unique 14-character identifier called the APPUIID. This number is unique to this installation of the app.
- *apps.contoso.com.* The third part of the unique domain is the app web hosting domain. You can configure this domain in Central Administration for an on-premises farm. In Office 365, the app web hosting domain is always sharepoint.com.

App Entry Points

Because an app entry point is the first thing that users see when they begin using a SharePoint app, it has an important effect on the user's experience of the app. You can choose between three types of app entry points:

- *Start page.* A start page can be thought of as the home page for your app. Every app must have a start page.
- *App part.* An app part enables users to add the app to a SharePoint page in the host web in the same way they would add a Web Part.
- *UI custom actions.* A UI custom action enables developers to add a control to the ribbon or to a menu in the host web.

- App entry points
 - Start page
 - App parts
 - UI custom commands
- The chrome control
 - Use the chrome control to inherit style and links from the host web in a cloud app

Although every app must have a start page, app parts and UI custom actions are optional. The following sections provide guidance on planning your app entry points.

Start page

A start page is required for every SharePoint app. In SharePoint-hosted apps, the start page is stored in the app web, often in a folder named **pages**. In cloud-hosted apps, the start page is stored in the remote web.

When users click on the app in the host web, SharePoint forwards them to the app start page. Users see only the start page with no SharePoint chrome—this is known as a "full immersion experience." It is up to you as a developer to implement the SharePoint app user interface guidelines, which requires a link back to the host web on every app page. The SharePoint hosted app Visual Studio template includes a master page called App.master that includes this link. However, if you choose not to use this master page, you must ensure that you provide the link to the host app. You must also implement this link yourself when you develop a cloud-hosted app.

App part

You can choose to implement an app part in your SharePoint app. If you provide an app part, users can include your app on a page in the host web by using the same method to add a Web Part. In this case, the app is presented on the host webpage by using an IFrame HTML element.

To include an app part in your SharePoint app, add a new client Web Part to your project in Visual Studio. This will add a new file called Elements.xml to the project. You can configure the page that the app part should display in this Elements.xml file.

UI custom actions

You can also choose to implement one or more UI custom actions as entry points for your app. A UI custom action appears in the host web as a control in one of two locations:

- On the ribbon.
- On the Edit Control Block (ECB). This is the menu that is displayed for individual items in SharePoint lists or individual documents in document libraries.

In Visual Studio, you can add a custom action to your app by adding an item based on the UI Custom Action template. Like app parts, this template adds a new Elements.xml file to your app. In this Elements.xml file, you can specify properties for the control, such as the menu location where the control will appear and the title.



Note: The code required to implement these entry points is presented in the next lesson.

The chrome control

The user interface for an app should closely match the user interface of the host web so that the app feels like a natural extension of the host web. As you have seen, the Visual Studio template for SharePoint-hosted apps includes the App.master master page and other files for this purpose. However, if you are building a cloud-hosted app, it can be more difficult to create a look and feel that extends the host web. To ease this problem, you can use the chrome control.

The chrome control enables the remote web to:

- Inherit header elements from the host web. Because the header elements include links to style sheets, this ensures that colors, fonts, and graphics are automatically inherited.
- Render a link to the host web. This implements the UI requirement that users see a link to the host web on every page of the app.
- Optionally, render a drop-down list box similar to the **Site Settings** menu in a SharePoint site. This is a good place to render configuration options for your app.

The chrome control is contained in the sp.ui.controls.js JavaScript library in the SharePoint LAYOUTS directory. You can copy this library to your remote web to use it.

Data Storage and Access

When you design any application, data storage and data access are key concerns. If you are building a SharePoint app, much of the data you work with may be stored in SharePoint, either in an app web or in a host web. To gain access to SharePoint, your app must first authenticate with SharePoint. Alternatively, if you are building a cloud-hosted app, you can include a database to be hosted on a database server or in Windows Azure SQL Database.

- Client-side object model (CSOM)
- .NET Framework object model
- REST API
- App authentication
 - Internal authentication
 - External authentication
- Data outside SharePoint

The client-side object model

The client-side object model (CSOM) is a library of objects and functions that enable developers to access almost the entire range of SharePoint functionality from a client application. There are two versions of CSOM included with SharePoint 2013:

- *The JavaScript CSOM.* This is a JavaScript library for accessing SharePoint from browsers, mobile apps, and other scripting environments. The library is stored in the sp.js file, found in the SharePoint LAYOUTS folder.

- *The Managed CSOM.* This is a dynamic link library (DLL) that you can include as a reference in desktop applications and other .NET Framework applications. The managed CSOM consists of two files, Microsoft.SharePoint.Client.dll and Microsoft.SharePoint.ClientRuntime.dll, both of which are stored in the SharePoint ISAPI folder.

Both the JavaScript CSOM and the managed CSOM call SharePoint through a Windows Communication Foundation (WCF) service called Client.svc. The CSOM has been greatly expanded in SharePoint 2013. You can access not only the core SharePoint functionality, such as lists and document libraries, but also all SharePoint workloads, including enterprise content management, web content management, record management, search, and communities. This expansion helps ensure apps have full access to SharePoint and can become the principal customization method available to SharePoint developers.

The .NET Framework object model

In a SharePoint-hosted app, you can only run client-side script code, so you must use the JavaScript CSOM to access SharePoint. However, if you are building a cloud-hosted app, you can use ASP.NET to run .NET Framework managed code, such as C# or Visual Basic code, which executes on an IIS web server or in Windows Azure. In this code, you can reference and call the SharePoint server-side object model.



Note: Accessing the SharePoint hierarchy from server-side code was covered earlier in Module 2, where you created and accessed lists and libraries by using the SharePoint server-side object model.

The REST API

Representational State Transfer (REST) is a standard for web services that eases development. REST web services are characterized by:

- *URIs for every operation.* These URIs are logical and user-friendly. For example, to access the properties of a site in SharePoint, you can use the URI `/_api/site`.
- *HTTP verbs.* RESTful services use standard HTTP verbs, such as GET, POST, PUT, and DELETE to specify create, read, update, and delete operations.
- *Standard response formats.* RESTful services usually return data in JavaScript Object Notation (JSON) Atom Publishing Protocol (AtomPub) format. These formats are simple to use in JavaScript code.

The Client.svc WCF service used by the CSOM is also a RESTful service, because it implements these standards. This means it is very easy to call the service directly from JavaScript code, without using the CSOM. jQuery also has extra support for REST calls.

Authentication in SharePoint

In a SharePoint farm solution or a SharePoint sandboxed solution, all code runs in the security context of the current user. This means developers do not have to worry about authentication when developing code, because SharePoint has already authenticated the user.

In a SharePoint app, authentication is not as simple. Because the app itself is a security principal, it must be authenticated and authorized to access SharePoint data.

The authentication method you use depends on the hosting model for your app, as well as your security requirements. Your app can only authenticate against SharePoint when you access through CSOM or the REST API. If any other access method is used, you can only authenticate the user.

SharePoint supports two types of authentication for the app. The first is known as internal authentication. In this type of authentication, the app makes a call to a SharePoint service in the context of the app web. SharePoint can use internal authentication in two scenarios:

- *For SharePoint-hosted apps.* All SharePoint hosted apps can use internal authentication because the request is always from within the app web.
- *For cloud-hosted apps that use the cross domain library.* The cross domain library is a client-side JavaScript library that enables cloud-hosted apps to make calls from within the context of the app web. Cloud-hosted apps do not usually require an app web—instead, they use a remote web on an external web server. If you want to use internal authentication with your cloud-hosted app, you must create both a remote web and an app web and use the cross domain library.

In any other scenario, SharePoint authenticates the app by using external authentication. External authentication is the name for app authentication in SharePoint when the call is outside the context of an app web. Such calls originate from cloud-hosted apps that do not use the cross domain library. SharePoint performs external authentication by using one of two methods:

- *OAuth.* This open authentication standard requires the Windows Azure Access Control Service (ACS).
- *Server to Server authentication.* Also known as S2S, this form of authentication requires a trust relationship between the SharePoint web servers and the web server that hosts the remote web.



Note: Internal authentication is covered in greater detail in Module 9, and external authentication is examined in Module 10.

Data outside SharePoint

SharePoint is an excellent store of content, so most apps will access and update SharePoint items or documents. However, it is also possible to create an app that works with data stored in non-SharePoint systems. Often such external systems are databases, but other stores are also possible.

If you are building a SharePoint hosted app, you will usually work exclusively with SharePoint data because you cannot store and access a database within a SharePoint app web. However it is possible to work with a database by using an architecture such as:

- A database hosted on a database server such as SQL Server or Windows Azure SQL Database.
- A web service hosted on a web server or Windows Azure. The web service publishes the database and provides objects and functions for data access and modification through HTTP calls.
- Client-side code in the SharePoint-hosted app that calls the web service. Some script libraries, such as jQuery, make it easy to call a web service from JavaScript code running in the browser.

Alternatively, you may prefer to use a cloud-hosted app in which server-side code accesses and manipulates data. An example of one such architecture is:

- A database hosted on a database server such as SQL Server or Windows Azure SQL Database.
- A remote web in which server-side code accesses and manipulates the data. You can use ASP.NET Web Forms or MVC to create the remote web. Data access technologies such as LINQ and Entity Framework make it very easy to code data interactions.

Packaging and Publishing Apps

Apps for SharePoint make it very easy to encapsulate your app into a package, which is a single file of a standard format. A package contains all the files and information necessary to run the app in another SharePoint system.

Packaging apps

After you complete your app, you must create an app package, which includes all the information that SharePoint needs to run the app.

If you have built a SharePoint-hosted app or an autohosted app, the app package includes all the webpages, script files, configuration files, and other data that SharePoint and Windows Azure require to run the app. In these cases, the app package is entirely stand-alone and users need nothing else to install and run the app. However, if you have built a provider-hosted app, the app package contains all the information SharePoint requires to connect you to the remote web, but you must provision the remote web and any associated databases or other services yourself.

A package is a compressed file with the .app file name extension. The contents of the package file varies, depending on whether you have built a SharePoint-hosted or cloud-hosted app and on the details of your design. However, the following files are often present in the .app file:

- *AppManifest.xml*. This file is required for every SharePoint app. It includes essential configuration data that SharePoint needs to run the app, including the URL of the app start page and the app authentication mechanism.
- *AppIcon.png*. This file is an icon that SharePoint uses to display the app in catalogs and host webs. Although this file is optional, it is recommended that you include it to help create an identity for your app.
- *Solution file*. If your app includes SharePoint objects such as lists, document libraries, items, and pages, they are zipped into a SharePoint solution package. This is a compressed file with a .wsp file name extension. Solution files may be familiar if you have created farm solutions or sandboxed solutions. However, in an app, the solution file is itself compressed into the .app file.
- *Data-tier application package*. If you have created an autohosted app, the app package includes a data-tier application package, which is a file with a .dacpac file name extension. This file is used by Windows Azure SQL Database to provision a new database every time a customer installs your app.

Publishing apps

After you create an app package, you must choose a location to publish it so that users can locate and install it. Packages can be published to the Office Catalog or app catalogs so that users can find and install them.

The Office Store

Publish the app to the Office Store when you want to make it available to all SharePoint 2013 installations, both on-premises and in the cloud. Global SharePoint administrators and users can purchase and install your app from the Office Store by downloading the app package.

You must have a dashboard seller account for the Office Store to publish your apps. The dashboard seller account helps you to manage licenses for your app, and it supports collecting credit card payments. You can publish apps after your dashboard seller account is approved.

- Packaging apps
 - .app file name extension
 - A package typically includes the following files:
 - AppManifest.xml
 - AppIcon.png
 - Solution file
 - Data-tier application package
- Publishing apps
 - Office Store
 - App catalogs

After you publish an app, there is an approval process that must be completed before users can see the app in the Office Store. In this process, Microsoft verifies that the app includes only valid resources and conforms to certain requirements. The process may take several days, but it ensures that Office Store apps are of high quality.

App catalogs

If you have developed an app for a specific SharePoint on-premises farm or Office 365 tenancy, you can also consider publishing the app in an app catalog. An app catalog is a special type of document library specifically designed for apps.

In an Office 365 tenancy, there is one app catalog that is automatically created when you subscribe to the service. In an on-premises farm, no app catalog is created by default. Instead, administrators must create the app catalog. Each app catalog has web application scope, so you must create one for every web application where you need to publish apps.

Discussion – Choosing Technologies

Read through the following scenario. Then discuss with the class the design questions listed.

A photo management solution

You are a software developer working for a magazine. The company uses an on-premises SharePoint farm to manage content such as articles, editorials, and reviews. However, photos are currently managed outside SharePoint because the default SharePoint workflows do not closely match the photo management process. You want to create a photo management app to enable the magazine to manage all content, including photography, in SharePoint.

Creating a photo management app for a magazine

Design decisions:

- Hosting model?
- Development tool?
- Photo storage location?
- Photo storage web?
- Entry points?
- Publishing location?

Design decisions

Discuss the following questions with the class:

- Should you create a SharePoint-hosted, autohosted, or provider-hosted app for photo management?
- Should you use Visual Studio or Napa to develop your app?
- Should photos be stored in the SharePoint content database or a database outside SharePoint?
- Should photos be stored in the host web, app web, or remote web?
- What entry points are required for the app?
- Where should you publish the completed app?

Lesson 2

Developing Apps for SharePoint

Now that you have an overall understanding of apps for SharePoint and how they are built, you need to know how to use Visual Studio to build a SharePoint app and communicate with SharePoint. This lesson examines the code needed to implement common tasks that are often necessary in apps for SharePoint.

Lesson Objectives

After completing this lesson, you will be able to:

- Use CSOM and the REST API to connect to SharePoint and access data from an app.
- Choose the correct Visual Studio project template and item template for a specific scenario.
- Describe the structure of an app project in Visual Studio.
- Use cross-domain calls to access SharePoint information from an app.
- Manage licenses for an app published in the Office Store.

Communicating with SharePoint

When you communicate with SharePoint from a client, you can choose from a range of endpoints. The endpoint you choose depends on where the code runs and the language you want to use.

The JavaScript CSOM

The JavaScript CSOM is a script library that you can include in a SharePoint-hosted or cloud-hosted app. The library enables you to call almost every SharePoint 2013 function, including advanced workload functions from code that runs within a web browser.

To access data in SharePoint through the CSOM, you must first load the current context.

- Connecting to a SharePoint site by using the managed CSOM:

```
//Obtain the App Web URL
string appWebUrl = Page.Request["SPAppWebUrl"];
//Obtain the client context
using (ClientContext context = new
    ClientContext(appWebUrl))
{
    Site siteCollection = context.Site;
    context.Load(siteCollection);
    context.ExecuteQuery();
    string url = siteCollection.Url;
}
```


The following example shows how to access the current site, through the JavaScript CSOM.

Accessing SharePoint through the JavaScript CSOM

```
"use strict";

var Contoso = window.Contoso || {};

Contoso.SiteCollection = function() {
    var siteCollection,
        context,

        getSiteCollectionURL = function() {
            var context = new SP.ClientContext.get_current();
            var siteCollection = context.get_site();
            context.load(siteCollection);
            context.executeQueryAsync(onSuccess, onFailure);
        },

        onSuccess = function () {
            alert("Site Collection URL: " + siteCollection.get_url());
        },

        function onFailure() {
            alert("Could not load the site collection");
        }

    return {
        GetURL: getSiteCollectionURL
    }
}();

$(document).ready(function () {
    Contoso.SiteCollection.GetURL();
});
```

Notice that queries to the Client.svc service are sent asynchronously. This ensures that the browser does not freeze while it waits for a response.

The .NET Framework CSOM

The .NET Framework CSOM is a compiled dynamic link library (DLL) that you can use when you build desktop applications or certain types of mobile applications. Because the .NET Framework CSOM is a set of managed .NET Framework objects, you get advanced IntelliSense help as you write SharePoint access code in Visual Studio. You can also choose to use any .NET Framework language including C# and Visual Basic.

The following example shows how to access the current site through the .NET Framework CSOM.

Accessing SharePoint through the .NET Framework CSOM

```
string appWebUrl = Page.Request["SPAppWebUrl"];
using (ClientContext context = new ClientContext(appWebUrl))
{
    Site siteCollection = context.Site;
    context.Load(siteCollection);
    context.ExecuteQuery();
    string url = siteCollection.Url;
}
```

As for the JavaScript CSOM, queries from the .NET Framework CSOM are sent asynchronously.

The REST API

Both the JavaScript CSOM and the .NET Framework CSOM call a SharePoint web service called Client.svc. This web service also includes a very wide range of RESTful operations, which make it easy to call the service directly, without using either CSOM. In addition, SharePoint publishes the Client.svc web service under the alias `_api`. Therefore, if you have a SharePoint site with the following URL:

```
http://mysite.contoso.com/
```

you can access the Client.svc service at the following URL:

```
http://mysite.contoso.com/_api
```

For example, to obtain information about the SharePoint site itself, such as the site name, you would use the following URL:

```
http://mysite.contoso.com/_api/site
```

jQuery can also simplify access to Client.svc, because jQuery includes extensive support for RESTful services.

The following example shows how to get information about the current SharePoint site by using the REST API and jQuery.

Accessing Site Information by Using the REST API

```
$(document).ready( function () {
    $.getJSON(_spPageContextInfo.webServerRelativeUrl + "/_api/site", function (response) {
        $("#sitename").text('The site name is' + response.d.Title);
    });
});
```

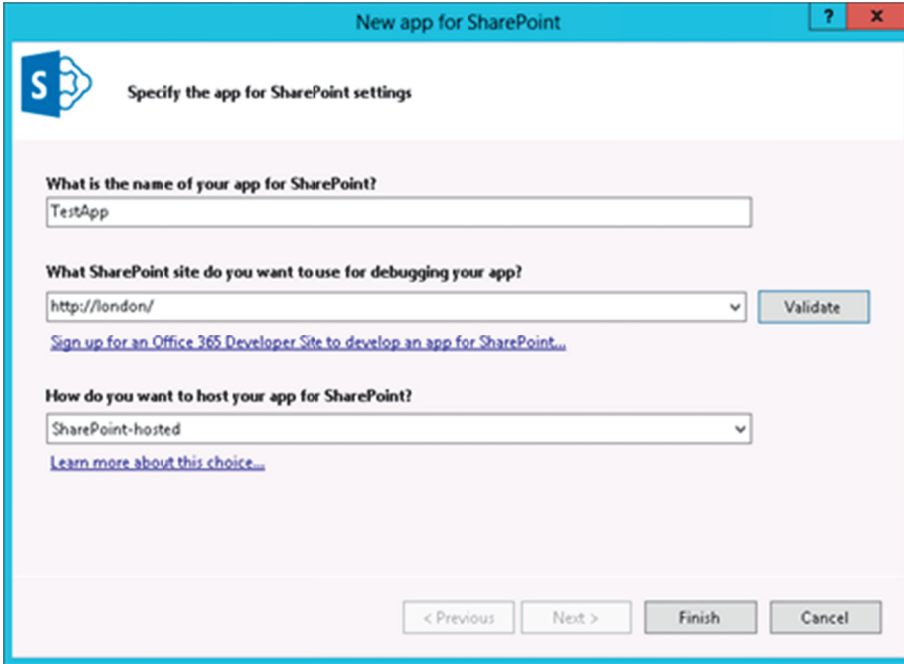
Visual Studio Templates for Apps for SharePoint

When you use Visual Studio to create a new SharePoint app, there is a single project template to use: App for SharePoint 2013. When you select this template for a new project, Visual Studio requests that you specify three properties for the app:

- The app name.
- The SharePoint site to use for debugging the app. When you press F5 to run the app with debugging, Visual Studio compiles, packages, and deploys the app to this SharePoint site. You should use a dedicated development farm for debugging and not a production environment.
- The hosting model. You can choose to create a SharePoint-hosted, autohosted, or provider-hosted app.

- The App for SharePoint 2013 project template
- Default files in an app project
- Visual Studio item templates

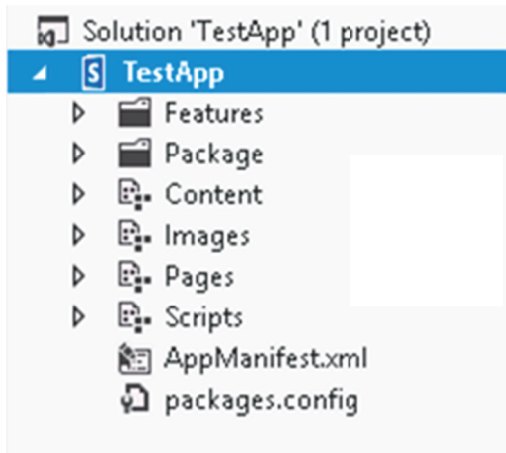
The SharePoint app creation wizard in Visual Studio is shown in this screen shot:



When you provide values for each property and click **Finish**, SharePoint creates the new project and adds the required files, depending on the hosting model you selected. These files may include:

- *AppManifest.xml*. Use this file to configure essential app values, such as title, app icon, and start page. When you open this file, Visual Studio displays a special designer to ease configuration, but you can also choose to edit the XML code directly.
- *AppIcon.png*. The default app icon is a simple PNG file that you can edit within Visual Studio and is found in the Images folder. Alternatively, you can replace this file with your own graphic, or you can specify a different icon in the AppManifest.xml file.
- *default.aspx*. SharePoint-hosted apps include this file as the default start page for the app. This is an ASP.NET Web Forms page that implements the `<asp:Content>` tags that SharePoint requires.
- *app.js*. This file provides some example JavaScript functions that work with the JavaScript CSOM. It is a good place to write code that implements your core functionality.
- *jQuery-1.7.1.js*. This is the jQuery script library. If you prefer, you can upgrade to the latest version of the library by replacing this file.

This screen shot shows the default structure of a SharePoint hosted app as it appears in Visual Studio Solution Explorer.



To continue to build your app, you can add new items based on the following Visual Studio item templates:

- *Client Web Part*. Use this template to build an app part entry point for your app.
- *Menu Item Custom Action*. Use this template to build a UI custom action entry point for your app. This template adds controls to the ECB.
- *Ribbon Custom Action*. Use this template to build a UI custom action entry point for your app. This template adds controls to the host site ribbon.
- *List*. Use this template to create a SharePoint list in the app web.
- *Content Type*. Use this template to create a custom SharePoint content type in the app web.
- *Workflow*. Use this template to create a workflow that models a business process for your organization.
- *Site Column*. Use this template to create a custom site column for use in lists, libraries, and content types in the app web.

In a SharePoint hosted app, you can also add HTML pages, style sheets, JavaScript files, images and other web content.

Demonstration: How to Create a SharePoint App from a Visual Studio Template

In this demonstration, you will see:

- How to create a new SharePoint hosted app project in Visual Studio 2012
- The default structure and important files within a SharePoint app
- How to deploy and debug an app in the development SharePoint farm

Demonstration Steps

- Start the 20488B-LON-SP07 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.

- In Visual Studio 2012, create a new project using the **App for SharePoint 2013** template. Use the following information:
 - Project Name: **HelloWorldApp**
 - App Name: **Hello World**
 - SharePoint site: **http://dev.contoso.com**
 - Hosting model: **SharePoint-hosted**
- Describe to the students the structure of the project as it is shown in the Solution Explorer pane.
- Describe the script files that the app includes by default.
- On the **default.aspx** start page, set the page title to be **Hello World**.
- Start the app in debugging mode. If you are prompted to log on, use the Administrator's credentials.



Note: Point out to the students that the page title is Hello World and that the current user's name is displayed at the bottom of the page.

- Navigate to the Site Contents for the Contoso Development Site site.



Note: Point out to the students that the Hello World app is listed with the label **new!**

- Stop debugging and close Visual Studio

Cross Domain Calls

The SharePoint app model places a strong emphasis on client-side scripting. However, browsers place certain restrictions on client-site scripts to prevent cross-site scripting (XSS) attacks. In a cross-site scripting attack, a malicious user injects a script into a page, perhaps by using an unchecked text input box. One precaution browsers take to prevent cross-site scripting is to prevent any script that originates from a domain that is different from the current webpage. This is a problem for apps for SharePoint because app webs and remote webs are in a different domain from host webs. To enable scripts in app webs and remote webs to run, you must use one of the following secure cross-site scripting techniques that are available to SharePoint developers: the cross domain library or the web proxy

- Using the cross-domain library
 - Access content in the app web from JavaScript in a remote web
 - SP.RequestExecutor.js
 - AppWebProxy.aspx
- Using the web proxy
 - Access content in SharePoint or elsewhere from JavaScript in a remote web
 - SP.WebRequestInfo
 - Trusting domains for cross domain calls

The cross-domain library

The cross-domain library enables JavaScript code in a cloud-hosted app's remote web to access data in the SharePoint app web, such as list items or documents. The cross-domain library is a JavaScript library. You can find it in the SharePoint **LAYOUTS** folder in a file named **SP.RequestExecutor.js**. The cross-domain library calls methods in the **AppWebProxy.aspx** page, which is also found in the **LAYOUTS** folder.

The following JavaScript code shows how to use the cross-domain library to get the details of an item in an app web.

Using the Cross Domain Library

```
"use strict";

// Get the Contoso object or create a new one if it doesn't exist.
var Contoso = window.Contoso || { };

// Create an object for cross domain calls.
Contoso.CrossDomain = function () {
  // The load function queries data across domains.
  var load = function () {
    var appWebUrl = getQueryStringParameter("SPSourceAppUrl");
    var executor = new SP.RequestExecutor(appWebUrl);
    var requestUrl = appWebUrl + "/_api/web/lists/getByTitle('Comments'/items/" +
      "?$select=Id,Subject,Body";

    executor.executeAsync(
      {
        url: requestUrl,
        method: "GET",
        headers: { "Accept": "application/json;odata=verbose" },
        success: onSuccess,
        error: onError
      }
    ),

    onSuccess = function (response) {
      // Parse returned data here.
    },

    onError = function (response, errorCode, errorMessage) {
      // Handle the problem here.
    };

    return {
      load: load
    }
  }();

  // Call the load method when the page finishes loading.
  $(document).ready(function () {
    Contoso.CrossDomain.load();
  });
};
```

The web proxy

A second method for making cross-domain calls securely is to use the web proxy. When you use the web proxy, you can call any endpoint in SharePoint or any external location, not just endpoints in the current app web. For this reason, the web proxy is an excellent tool to use when you want to create mash-ups of data from disparate sources.

To use the web proxy, you must create an instance of the `SP.WebRequestInfo` object, which is included in the `sp.js` JavaScript library.

The following JavaScript code demonstrates how to use the web proxy to access items in a SharePoint list.

Using the Web Proxy

```
"use strict";

// Get the Contoso object or create a new one if it does not exist
var Contoso = window.Contoso || { };
Contoso.ResponseDocument;

// Create an object for cross domain calls.
Contoso.CrossDomain = function () {
    // The load function queries data across domains.
    var load = function () {
        var context = SP.ClientContext.get_current();
        var request = new SP.WebRequestInfo();
        var requestUrl = appWebUrl + "_api/web/lists/getByTitle('Comments')/items/" +
            "?$select=Id,Subject,Body";

        request.set_url(requestUrl);
        request.set_method("GET");

        window.Contoso.ResponseDocument = SP.WebProxy.invoke(context, request);
        context.executeQueryAsync(onSuccess, onError);
    },

    onSuccess = function () {
        var xmlDocument = $.parseXML(window.Contoso.ResponseDocument.get_body());
        // Parse the xmlDocument here to display results.
    },

    onError = function (error) {
        // Resolve the error here.
    };

    return {
        load: load
    }
}();

// Call the load method when the page has finished loading.
$(document).ready(function () {
    Contoso.CrossDomain.load();
});
```

To access content in a SharePoint or external domain through the web proxy, you must configure the app to trust that domain. This is simple to do in the AppManifest.xml file.

Configuring the Web Proxy to Trust a Domain

```
<RemoteEndpoints>
  <RemoteEndPoint Url="http://intranet.contoso.com" />
</RemoteEndpoints>
```

Licenses for Apps for SharePoint

If you publish your app in the Office Store, you can use the app licensing framework to keep track of customer licenses and enforce legal use. The licensing framework itself does not enforce app licenses, but you can add code to your app that checks and acts on that information. For example, your code could prevent any access for unlicensed customers, or it could prevent access to advanced features for unlicensed users. This approach can enable customers to evaluate an app before purchase.

The license framework includes:

- *The Office Store*. This is where customer acquire licenses. The store handles payment through credit cards or other mechanisms.
- *A license store*. This stores valid licenses and assists customers when they renew licenses.
- *License APIs*. Developers can use these APIs to check and act on the license that the current user possesses.
- *A license web service*. The license APIs call this web service to check a license.



Additional Reading: For more information about the licensing framework and the mechanisms by which users acquire and renew licenses, see *Licensing apps for Office and SharePoint* at <http://go.microsoft.com/fwlink/?LinkID=307094>.

How to check licenses

To check that a customer is licensed to use your app, you should write code to perform the following actions:

1. Download the license tokens from the SharePoint server.
2. Pass the license tokens to the Office Store license verification web service for validation.

Your code should perform these steps when a user first attempts to access an app feature that requires a valid license.

- The Office Store license framework
- Downloading license tokens from SharePoint
- Checking license tokens with the Office license verification web service
- Using test licenses

The following C# code demonstrates how to obtain license tokens from SharePoint and check them with the verification web service.

Downloading and Verifying License Tokens

```
// The URL of you SharePoint app web or host web.
string webUrl = "http://localhost";

productId = new Guid("123346"); // Place the app's product ID here
using(ClientContext context = new ClientContext(webUrl))
{
    // Get the license tokens from the SharePoint server.
    ClientResult<AppLicenseCollection> licensecollection =
        Microsoft.SharePoint.Client.Utilities.Utility.GetAppLicenseInformation(context, productId);
    context.ExecuteQuery();

    foreach (AppLicense license in licensecollection.Value)
    {
        VerifyLicenseToken(license.RawXMLLicenseToken);
    }
}

private void VerifyLicenseToken(string rawLicenseToken)
{
    // Set up required objects.
    VerificationServiceClient service = null;
    VerifyEntitlementTokenResponse = null;
    VerifyEntitlementTokenRequest request = new VerifyEntitlementTokenRequest();
    request.RawToken = rawLicenseToken;


    // Send the current token to the Office license verification service.
    try
    {
        service = new VerificationServiceClient();
        var result = service.VerifyEntitlementToken(request);
    }
    catch (Exception ex)
    {
        // Handle errors here.
    }

    if (result.EntitlementType.ToUpper() == "PAID")
    {
        // The customer has paid for a license. Enable the full feature set.
    }
}
```

Using test licenses

License-dependent functionality in your application can be quite complex, with multiple features available, depending on the type of license that the customer has purchased. To prevent any licensing problems with apps after they are published to the app store, you must test the code that checks licenses very carefully to ensure that it functions as designed. Any coding errors can result in subscribed customers being unable to use the full features of the app they paid for.

To help you test license checking code in your application, you can create test licenses and import them into your development SharePoint farm.

 **Additional Reading:** To learn more about test licenses and how to set them up in your test environment, see *How to: Add license checks to your app for SharePoint* at <http://go.microsoft.com/fwlink/?LinkID=307095>.

Lab: Creating a Site Suggestions App

Scenario

The management team at Contoso wants to ensure that the new SharePoint 2013 intranet deployment meets the needs of end users. The team has asked you to investigate ways of capturing user feedback consistently across a variety of site collections. In this lab, you will develop an app that enables users to submit feedback and to view the feedback submitted by other users.

Objectives

After completing this lab, you will be able to:

- Create a SharePoint hosted app.
- Use the JavaScript CSOM to create and retrieve list items.

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-07
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Creating a New SharePoint App

Scenario

You have been asked to build a site suggestions app that enables users to submit feedback on the design and facilities available in a SharePoint site. You have decided to create the app in Visual Studio and add site columns, a content type, and a list to store feedback items. In this exercise, you will complete these tasks to create your app web infrastructure.

The main tasks for this exercise are as follows:

1. Create a New SharePoint Hosted App
2. Add Site Columns to the App
3. Create a Content Type for Suggestions
4. Create the Suggestions List

► Task 1: Create a New SharePoint Hosted App

- Start the virtual machine, and log on with the following credentials:
 - Virtual Machine: **20488B-LON-SP-07**
 - User name: **CONTOSO\Administrator**
 - Password: **Pa\$\$w0rd**
- Create a new Visual Studio project for your app. Use the following information:
 - Language: **Visual C#**
 - Template: **App for SharePoint 2013**
 - Name: **SiteSuggestionsApp**
 - Location: **E:\Labfiles\Starter**
- Set the following properties in the **New App for SharePoint** dialog box:

- App Name: **Site Suggestions**
- Debugging Site: **http://dev.contoso.com**
- Hosting Model: **SharePoint-hosted**
- ▶ **Task 2: Add Site Columns to the App**
 - Add a new site column named **SuggestionSubject** to the **SiteSuggestionsApp** project.
 - Add a new site column named **Feedback** to the **SiteSuggestionsApp** project.
 - Configure the **Feedback** site column as a multi-line text field, by setting the **type** property to **Note**.
 - Save your changes.
- ▶ **Task 3: Create a Content Type for Suggestions**
 - Add a new content type, named **Suggestion** to the **SiteSuggestionsApp** project.
 - Add the following columns to the **Suggestion** content type:
 - Suggestion Subject
 - Feedback
 - Save your changes.
- ▶ **Task 4: Create the Suggestions List**
 - Add a new list named **Suggestions** to the SiteSuggestionsApp project.
 - Bind the **Suggestion** content type to the **Suggestions** list. Remove the **Item** and **Folder** content types from the **Suggestions** list.
 - Set the list description to **Use this list to store site suggestions**.
 - Save your changes.

Results: A simple SharePoint hosted app with custom site columns, a content type and a list instance.

Exercise 2: Using the Client-Side Object Model

Scenario

Now that you have created the Site Suggestions SharePoint app and the site columns, content type, and list instance that it requires, you can add the user interface elements and the JavaScript code that enable users to create new feedback items and to browse feedback from other people.

You will start by adding an App.js file that has the necessary functions and the Contoso namespace defined. However, these functions are incomplete. You will add JavaScript CSOM code to complete the app.

The main tasks for this exercise are as follows:

1. Add the Contoso Framework
2. Build New Suggestion Functionality
3. Build the Display of Existing Suggestions
4. Build the Suggestion Details Display
5. Run the App

► Task 1: Add the Contoso Framework

- Delete the default **App.js** file from the **Scripts** folder in your project.
- Add the following file to the **Scripts** folder:
- E:\Labfiles\JavaScript Code\App.js

► Task 2: Build New Suggestion Functionality

- Copy all the code in the following text file to the clipboard:
- E:\Labfiles\Web Page Markup\CreateSuggestionsUI.txt
- Paste the copied code into the default webpage for your app. Paste the code at the end of the last **<asp:Content>** element.
- Create the following global variables in the **Contoso.SuggestionsApp** function in the **App.js** file:
 - suggestionsList
 - allSuggestions
 - currentSuggestion
- In the createSuggestion function, insert a line of code that creates a new variable. Use the following information:
- Name: **itemCreateInfo**
- Type: **SP.ListItemCreationInformation()**
- Pass the **itemCreateInfo** object to the **suggestionsList.addItem** method. Store the returned object in the **currentSuggestion** variable.
- Use the **currentSuggestion.set_item()** method to set the value of the **Subject** column to value of the page element with ID **subject-input**.
- Use the **currentSuggestion.set_item()** method to set the value of the **Feedback** column to value of the page element with ID **feedback-input**
- Update the **currentSuggestion** object.
- Pass the updated **currentSuggestion** object to the **context.load()** function.
- Save your changes.

► Task 3: Build the Display of Existing Suggestions

- Add a **<p>** element to the bottom of the **default.aspx** page with the following content: **Here are the current suggestions:**
- After the new **<p>** element, insert a **<div>** element with the ID **suggestions-list**.
- In the **getSuggestions()** function, insert a line of code that uses the **getByTitle()** function to obtain the Suggestions list. Store the list in the suggestionsList variable.
- Pass a new **SP.CamlQuery** object to the **suggestionsList.getItems()** function to get all the items in the list. Store the items in the allSuggestions variable.
- Pass the allSuggestions variable to the **context.load()** method.
- In the **\$(document).ready()** function, insert a call to the **getSuggestions()** function.
- Save your changes.

► **Task 4: Build the Suggestion Details Display**

- Copy all the code in the following text file to the clipboard:
- E:\Labfiles\Web Page Markup\DisplaySuggestionUI.txt
- Paste the copied code into the default webpage for your app. Paste the code at the end of the last **<asp:Content>** element.
- In the **displaySuggestion()** function, insert a line of code that passes the **suggestionId** parameter to the **suggestionList.getItemById()** method. Store the result in the **currentSuggestion** variable.
- Pass the **currentSuggestion** variable to the **content.load()** method.
- Copy all the code in the following text file to the clipboard:
- E:\Labfiles\Style Code\App Styles.txt
- Paste the copied code into the **Content/App.css** style sheet.
- Save your changes.

► **Task 5: Run the App**

- Start the Site Suggestions app in debugging mode and log on as Administrator.
- Create a new test suggestion and click it in the suggestions list.
- Stop debugging and close all open windows.


Results: A simple SharePoint app in which users can create new suggestions with feedback for site improvements. The app will also enable users to browse suggestions from other users.


Question: In Exercise 2, Task 3 you added styles to the **App.css** style sheet. How would the app function if you had not performed this step?

Question: When you click a suggestion in the list, the suggestion fades in smoothly. How is this fade achieved in the code?

Module Review and Takeaways

This module has presented an overview of the new apps for SharePoint development framework. The framework enables developers to create apps that work in both Office 365 and in on-premises SharePoint installations. Apps for SharePoint also have a more flexible security model and will be less dependent on the version of SharePoint that your organization runs.

 **Best Practice:** If you can develop your custom functionality as an app, you should do so in preference to developing farm solutions or sandboxed solutions. The reason for this recommendation is that apps have better isolation, and therefore greater stability, than solutions. Also, a completed app can be marketed in the Office Store.

 **Best Practice:** Always use JavaScript in combination with script libraries for client-side code. JavaScript is widely supported and script libraries help to circumvent awkward browser differences that can needlessly increase a developer's workload.

Common Issues and Troubleshooting Tips

Common Issue	Troubleshooting Tip
When you call the <code>jQuery()</code> or <code>\$()</code> function, you see an error message.	

Review Question(s)

Question: Your company has created an app that manages company assets such as equipment. You have been asked to integrate this with the host web so that users can place the app on a Web Part page. What kind of entry point does this require?

Tools

- *Visual Studio*. This is the primary IDE to use for developing apps of all kinds.
- *Napa*. This is a web-based IDE that is designed to make it very easy to create simple apps.

Module 8

Client-Side SharePoint Development

Contents:

Module Overview	8-1
Lesson 1: Using the Client-Side Object Model for Managed Code	8-2
Lab A: Using the Client-Side Object Model for Managed Code	8-9
Lesson 2: Using the Client-Side Object Model for JavaScript	8-14
Lesson 3: Using the REST API with JavaScript	8-26
Lab B: Using the REST API with JavaScript	8-33
Module Review and Takeaways	8-38

Module Overview

In the apps for SharePoint infrastructure, you cannot build apps that execute code within SharePoint processes. Instead, code must run in the browser, in other client-side applications, or on other web servers. In this module, you will see how to write code that executes on a client and manipulates SharePoint data. The client is often a web browser, but it also may be a desktop application, a mobile application, or another type of package. You can choose to use the JavaScript client-side object model (CSOM) or the managed code CSOM—these are libraries of classes that make it easy for you to write code that calls SharePoint. Alternatively, you can choose to call the REST API directly.

Objectives

After completing this module, you will be able to:

- Use the client-side object model for managed code to interact with a SharePoint deployment.
- Use the client-side object model for JavaScript to interact with a SharePoint deployment.
- Use the REST API with JavaScript or C# to interact with a SharePoint deployment.

Lesson 1

Using the Client-Side Object Model for Managed Code

When you develop client-side solutions, if your client-side code runs in a .NET application, such as for example, a desktop application or a mobile app, you can use the .NET Managed CSOM to access SharePoint. Alternatively, if you develop solutions to access SharePoint from code running in the browser, or other JavaScript interpreters you can use the JavaScript CSOM. The .NET Managed CSOM has a very similar set of classes and functions to the JavaScript CSOM. Therefore, when you work with SharePoint data from the Managed CSOM, you take a very similar approach, but the code looks different because of the differences between JavaScript and the .NET managed code language you choose. In this lesson, you will see C# examples that illustrate these differences.

Lesson Objectives

After completing this lesson, you will be able to:

- Choose whether to use the JavaScript CSOM or the Managed CSOM in your client application.
- Use the **ClientContext** object to discover information about the current site, site collection, and user.
- Load objects and run queries with the Managed CSOM.
- Read data from SharePoint lists and libraries.
- Modify SharePoint data by creating, updating, and deleting items.
- Handle server-side errors.

Overview of the Managed CSOM

The Managed CSOM has many similarities to the JavaScript CSOM and you will notice that you take the same approach to solve the same coding problem when using each CSOM. The code differences arise mostly because of the differences between JavaScript and a fully object-oriented .NET Framework language, such as Visual C#.

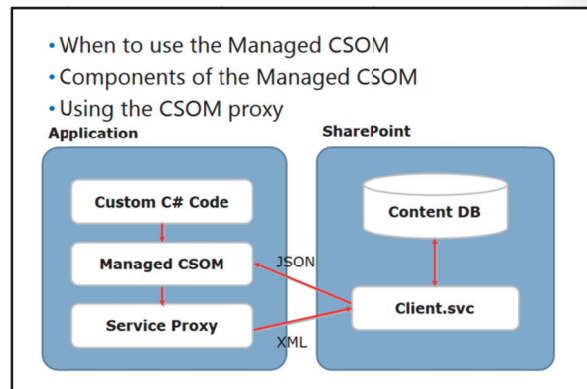
Consequently, if you know both JavaScript and C#, it is easy to write both browser-based code and managed code that works with SharePoint data.

When to use the Managed CSOM

You can use the Managed CSOM whenever you use the .NET Framework to build a client for SharePoint. For example:

- *A desktop application.* Such an application might use Windows Presentation Foundation (WPF) to implement a user interface and present SharePoint information along with data from other sources.
- *A website.* An ASP.NET web application, written in MVC or Web Forms, could use the Managed CSOM to access SharePoint data.
- *A cloud-hosted SharePoint app.* Cloud-hosted apps can include ASP.NET managed code that runs in the remote web. You can call SharePoint functions from ASP.NET code by using the Managed CSOM.

You cannot use the Managed CSOM in a SharePoint-hosted app because only client-side JavaScript code can be included in such apps.



Components of the Managed CSOM

The classes and functions that make up the Managed CSOM are stored in two files: Microsoft.SharePoint.Client.dll and Microsoft.SharePoint.ClientRuntime.dll. You can find these files in SharePoint ISAPI folder. To build an application that uses the Managed CSOM, you must add a reference to both of these DLLs in Visual Studio.

Just like the JavaScript CSOM, the Manage CSOM sends requests to the Client.svc WCF service.

Using the CSOM proxy

The Managed CSOM uses batched requests, just like the JavaScript CSOM does. Your managed code can make one or more requests by working with the CSOM proxy object. When you call the **ExecuteQuery()** method, all these batched requests are sent to the Client.svc service in a single operation. The Client.svc works with the service object model to execute the request against the SharePoint content database. Data is returned in the JSON format.

Using Client Context, Loading Objects, Executing Queries

Just like with the JavaScript CSOM, when you use the Managed CSOM, you must start by creating a client context object. The context object represents the user's current location within SharePoint and you can use it to gain access to the current site collection, site, and other objects.

The following code demonstrates how to create a context object by using the Managed CSOM in C# code. The context object is used to access the site collection and return its URL.

```
public string getSiteCollectionUrl () {
    string appWebUrl = Page.Request["SPAppWebUrl"];
    using (ClientContext context = new ClientContext(appWebUrl));
    {
        Site siteCollection = context.Site;
        context.Load(siteCollection, s=>s.Title,
            s=>s.ServerRelativeUrl);
        context.ExecuteQuery();
        return siteCollection.Url;
    }
}
```

Using the Client Context Object

```
public string getSiteCollectionUrl () {
    string appWebUrl = Page.Request["SPAppWebUrl"];
    using (ClientContext context = new ClientContext(appWebUrl));
    {
        Site siteCollection = context.Site;
        context.Load(siteCollection, s=>s.Title, s=>s.ServerRelativeUrl);
        context.ExecuteQuery();
        return siteCollection.Url;
    }
}
```

Notice the following features of the preceding code example:

- *appWebUrl*. To use the client context object, you must pass a string URL to the SharePoint site. In a cloud-hosted app, a URL for the app web is always found in the **SPAppWebUrl** request parameter. You can use this to supply the necessary URL to the client context.
- *Load()*. Because the Manage CSOM uses batched queries and the service proxy, you must use the **Load()** method to set up the object in the client context object for retrieval from the server. In the preceding example, the site collection is loaded in this way. You can set up a batch of requests by loading multiple objects before you execute a single query to retrieve them. If you want to specify the properties of the object that the query will return, pass lambda expressions to the **Load()** method. In the preceding example, lambda expressions are used to request the **Title** and **ServerRelativeUrl** properties.

- *ExecuteQuery()*. Use this method to execute the batch of queries.

Reading SharePoint Data

When you are working with SharePoint data, you must frequently begin by finding out what lists are present in a site and what items are in those lists. You can display this information in a table of data or populate the fields in an editor form. The Manage CSOM provides a range of functions that make reading data simple, for example:

- *ClientContext.Web.Lists*. This property of the SharePoint site returns a collection of all the lists and document libraries in the site.
- *ClientContext.Web.Lists.GetByTitle()*. This method on a collection of lists returns a specific list that matches the title passed.
- *List.GetItems()*. This method on a list returns a collection of items. If you use the method with no parameters, all the items in the list are returned. Alternatively, you can specify a CAML query to filter and order the information returned.

- Obtain a collection of lists in a site:

```
ListCollection allLists = currentWeb.Lists;
```

- Obtain a list by its title:

```
List suggestionsList = allLists.GetByTitle("Suggestions");
```

- Obtain items in a list by using a CAML query:

This example shows how use the **Web.Lists** property, the **Lists.GetByTitle()** method, and the **List.GetItems()** method from the Managed CSOM. The call to **GetItems()** uses a CAML query to order the results alphabetically.

Querying for Data in a SharePoint Site

```
public string investigateContent () {
    // Get the app web URL.
    string appWebUrl = Page.Request["SPAppWebUrl"];

    // Set up the client context.
    using (ClientContext context = new ClientContext(appWebUrl))
    {
        // Start by loading the site.
        currentWeb = context.Web;
        context.Load(currentWeb);

        // Now we have the SharePoint site, load the lists in the site.
        ListCollection allLists = currentWeb.Lists;
        context.Load(allLists);

        // Set up a CAML query to load items in the Suggestions list.
        string query = "<View><Query><OrderBy><FieldRef Name='Title' /></OrderBy></Query>" +
            "<ViewFields><FieldRef Name='ID' />< FieldRef Name='Title' /></ViewFields></View>";
        CamlQuery camlQuery = new CamlQuery();
        camlQuery.ViewXML = query;

        // Load the suggestions list.
        List suggestionsList = allLists.GetByTitle("Suggestions");
        context.Load(suggestionsList);

        // Get the items in the suggestions list by using the CAML query.
        ListItemCollection items = suggestionsList.GetItems(camlQuery);
        context.Load(items);
    }
}
```

```

// Now we have batched all the requests, we can execute the query.
context.ExecuteQuery();

// Tell the user about the lists.
string message = "The lists are as follows: ";
foreach (List currentList in allLists)
{
    message += currentList.Title + " ";
}

// Tell the user about the items in the suggestions list.
message += "The suggestions are as follows: ";
foreach (ListItem currentItem in items)
{
    message += currentItem.Title + " ";
}
return message;
}
}

```

Changing SharePoint Data

The Managed CSOM includes a range of classes and methods that you can call to create, update, and delete items in SharePoint lists. These include the following:

- *ListCollection.Add()*. This method creates a new list in the Lists collection for a SharePoint site.
- *ListCreationInformation*. This class enables you to set the properties of a new list. You must create a **ListCreationInformation** object and set its properties, and then pass the **ListCreationInformation** object to the **ListCollection.Add()** method.
- *List.AddItem()*. This method creates a new item in a SharePoint list.
- *ListItemCreationInformation*. This class enables you to set the properties of a new list item. Use the class as you use **ListCreationInformation** for new lists.
- *ListItem.Update()*. This method enables you to update list items. When you set properties of an existing list item, you must call the **Update()** method to save your changes.
- *ListItem.DeleteObject()*. This method remove an item from the list.

The following code shows an example class with methods you can call to create a new list, create a new item in the Suggestions list, update an existing item, and delete an item.

- Creating lists:
 - Create a ListCreationInformation object
 - Pass the object to Web.Lists.Add()
- Creating items:
 - Create a ListItemCreationInformation object
 - Pass the object to List.AddItem()
- Updating items:
 - Set fields on an item
 - Call Item.Update()
- Deleting items:
 - Call Item.DeleteObject()

Changing SharePoint Data

```
public class SharePointEditor
{
    string appWebUrl;

    public SharePointEditor()
    {
        appWebUrl = Page.Request["SPAppWebUrl"];
    }

    public void CreateList (string name, string description)
    {
        using (ClientContext context = new ClientContext(appWebUrl)
        {
            // Start by creating a new list creation information object.
            ListCreationInformation listInfo = new ListCreationInformation();
            listInfo.Title = name;
            listInfo.Description = description;
            listInfo.TemplateType = (int)ListTemplateType.GenericList;

            // Create the new list item and execute the query.
            List newList = context.Web.Lists.Add(listInfo);
            context.ExecuteQuery();
        }
        }

    public void CreateItem (string title)
    {
        using (ClientContext context = new ClientContext(appWebUrl)
        {
            // Get the suggestions list.
            List suggestionsList = context.Web.Lists.GetByTitle("Suggestions");
            context.Load(suggestionsList);

            // Create the List Item Creation Info object.
            ListItemCreationInformation itemInfo = new ListItemCreationInformation();

            // Create the new item and set its properties.
            ListItem newItem = suggestionsList.AddItem(itemInfo);
            newItem["Title"] = title;

            // You must call the Update method before you execute the query.
            newItem.Update();
            context.ExecuteQuery();
        }
        }

    public void UpdateItem (int id, string title)
    {
        using (ClientContext context = new ClientContext(appWebUrl)
        {
            // Get the suggestions list.
            List suggestionsList = context.Web.Lists.GetByTitle("Suggestions");
            context.Load(suggestionsList);

            // Get the item and set its properties.
            ListItem item = suggestionsList.GetItemById(id);
            item["Title"] = title;

            // You must call the Update method before you execute the query.
            item.Update();
            context.ExecuteQuery();
        }
        }
    }
}
```

```

}

public void DeleteItem (int id)
{
    using (ClientContext context = new ClientContext(appWebUrl)
    {
        // Get the suggestions list.
        List suggestionsList = context.Web.Lists.GetByTitle("Suggestions");
        context.Load(suggestionsList);

        // Get the item and delete it.
        ListItem item = suggestionsList.GetItemById(id);
        item.DeleteObject();

        // Remember to execute the query.
        context.ExecuteQuery();
    }
}
}
}
}

```

Handling Server-Side Errors

In the Managed CSOM, server-side errors can be handled in the same way as in the JavaScript CSOM: by using an **ExceptionHandlingScope** object and calling **StartTry()**, **StartCatch()**, and **StartFinally()** methods. However, C# supports using() {} code blocks, which you can use to ensure that the scope and its **try**, **catch**, and **finally** objects are disposed correctly before you call **ExecuteQuery()**.

The following code shows how to create an object that uses an exception handling scope to implement try/catch/finally code on the server.

- Use an exception handling scope for server-side errors as in the JavaScript CSOM
- Use using() {} code blocks to ensure correct disposal of objects
- Make sure there is only one call to ExecuteQuery() after the exception handling scope is disposed of

Using an Exception Handling Scope to Resolve Server-Side Errors

```

public class ErrorScope ()
{
    public void scope () {

        // Get the App Web URL so that you can get the client context.
        string appWebUrl = Page.Request["SPAppWebUrl"];
        using (ClientContext context = new ClientContext(appWebUrl)
        {

            // Create and start the exception handling scope.
            ExceptionHandlingScope xScope = new ExceptionHandlingScope(context);
            using (xScope.StartScope())
            {

                // Define the try block.
                using (xScope.StartTry())
                {
                    // Place code that may generate exceptions here.
                }

                // Define the catch block.
                using (xScope.StartCatch())
            }
        }
    }
}

```

```
        {  
            // Place code that handles exceptions here.  
        }  
  
        using (xScope.StartFinally())  
        {  
            // Place code that should execute regardless of errors here.  
        }  
    }  
}  
context.ExecuteQuery();  
}  
}
```

Lab A: Using the Client-Side Object Model for Managed Code

Scenario

The field sales team at Contoso regularly submits expense claims for mileage. Because of the fluctuating price of fuel, variable tax rates, and other factors, some salespeople find it difficult to calculate their expense claims accurately. To help the sales team, you will implement an app that calculates mileage expenses. To ensure that the app always uses the latest formula and to remove the processing burden from client browsers, you decide to create an autohosted app. The app will prompt users for the required information, calculate the claimable expenses, and then submit the claim on behalf of the salesperson.

Objectives

After completing this lab, you will be able to:

- Add code to a cloud-hosted SharePoint app.
- Use the Managed CSOM for to create and submit list items.

Lab Setup

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-08
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

The Mileage Recorder app is an autohosted SharePoint app in which the remote web is built with the ASP.NET MVC 4 programming model. In this lab, the infrastructure necessary to integrate the MVC remote web with SharePoint is already in place. You will complete a list in the app web and code in the remote web that uses the Managed CSOM to access SharePoint.

Exercise 1: Create the Mileage Claim List

Scenario

The Mileage Recorder app requires a SharePoint list to store mileage claims. In this exercise, you will create site columns, a content type, and the list itself. You will create these objects in the SharePoint app web.

The main tasks for this exercise are as follows:

1. Add Site Columns
2. Add the Mileage Claim Content Type
3. Add the Claims List

► Task 1: Add Site Columns

- Start the virtual machine, and log on with the following credentials:
- Virtual Machine: **20488B-LON-SP-08**
- User name: **CONTOSO\Administrator**
- Password: **Pa\$\$w0rd**
- Open the following Visual Studio solution:
- E:\Labfiles\LabA\Starter\MileageRecorder\MileageRecorder.sln

- Create a new site column named **Destination**. Use the following information:
 - Required: **TRUE**
- Create a new site column named **ReasonForTrip**. Use the following information:
 - Type: **Note**
 - Required: **TRUE**
- Create a new site column named **Miles**. Use the following information:
 - Type: **Integer**
 - Required: **TRUE**
- Create a new site column named **EngineSize**. Use the following information:
 - Type: **Integer**
 - Required: **TRUE**
- Create a new site column named **Amount**. Use the following information:
 - Type: **Currency**
 - Required: **TRUE**
- Save your changes
- ▶ **Task 2: Add the Mileage Claim Content Type**
 - Add a new content type named **MileageClaim** to the MileageRecorder SharePoint app web.
 - Add the following columns to the **MileageClaim** content type:
 - Destination
 - ReasonForTrip
 - Miles
 - EngineSize
 - Amount
 - Set the description of the **MileageClaim** content type to **This content type defines a Mileage Claim for the Mileage Recorder app.**
 - Save your changes.
- ▶ **Task 3: Add the Claims List**
 - Add a new list, named **Claims**, to the MileageRecorder SharePoint project.
 - In the **Claims** list, remove the **Item** and **Folder** content types, and then add the **MileageClaim** content type.
 - Remove the **Title** column from the **Claims** list.
 - Set the **Description** for the **Claims** list to **This list stores mileage claims.**
 - Save your changes.

Results: A cloud-hosted SharePoint app with a list for storing mileage claims configured in the app web.

Exercise 2: Add Mileage Claim Creation Code

Scenario

Now that you have created the Claims lists in the SharePoint app web, you can write code in the remote web that creates mileage claims. Your code will use the Managed CSOM to communicate with SharePoint.

The main tasks for this exercise are as follows:

1. Store URLs for Later Use
2. Code the Amount Calculation
3. Complete the Mileage Claim Create Action

► Task 1: Store URLs for Later Use

- In the **MileageRecorderRemoteWeb** project, in the **Controllers** folder, open the **HomeController.cs** file for editing.
- In the **Index** action method, store the value of the **SPAppWebUrl** query string parameter in a **Session** variable named **SPAppWebUrl**, if that query string parameter is not null.
- Also store the value of the **SPHostUrl** query string parameter in a **Session** variable named **SPHostUrl**, if that query string parameter is not null.
- Save your changes.

► Task 2: Code the Amount Calculation

- In the **Models** folder, open the **MileageClaim** class and add a new method named **calculateAmount**. Use the following information:
 - Parameters: none.
 - Return Type: void
- In the **calculateAmount** method, create the following **double** variables:
 - **result**, value null.
 - **lowMileageRate**, value 0.6.
 - **highMileageRate**, value 0.5.
- If the **EngineSize** property of the **MileageClaim** object is less than 1000, set the value of **lowMileageRate** to **0.3**, and then value of **highMileageRate** to **0.25**.
- If the **Miles** property of the **MileageClaim** object is less than 100, set the value of **result** to the **Miles** property multiplied by **lowMileageRate**. Otherwise, set the value of **result** to the **Miles** property multiplied by **highMileageRate**.
- Set the **Amount** property of the **mileageClaim** object to **result**.
- Save your changes.

► Task 3: Complete the Mileage Claim Create Action

- In the **MileageClaimController** code file, locate the following code:

```
//Insert mileage claim creation code here
```

- Call the **calculateAmount()** method on the **claim** object.
- Get the value of the **Session["SPAppWebUrl"]** property and store it in a new **string** variable named **appWebUrl**.

- In a **using** code block, create a new **ClientContext** object named **context**. Pass the **appWebUrl** to the **ClientContext** constructor.
- Use the **context** object to get the **Claims** list and store it in a variable named **claimsList**. Then load the **claimsList** object.
- Create a new **ListItemCreationInformation** object named **creationInfo**.
- Use the **creationInfo** object to add a new item to **claimsList**. Store the new item in a **ListItem** object named **newClaim**.
- Use the properties of the MVC model object **claim** to set the following properties of the SharePoint item **newClaim**:
 - Destination
 - ReasonForTrip
 - Miles
 - EngineSize
 - Amount
- Call the **Update** method for the **newClaim** item, and then execute the CSOM query.
- Save your changes.

Results: A cloud-hosted app that can create items in a SharePoint list.

Exercise 3: Display Mileage Claims on the App Start Page

Scenario

In the exercise, you will add the code that reads all the existing claims from the Claims list and displays those claims on the app's start page. You will also test the completed app.

The main tasks for this exercise are as follows:

1. Complete the Mileage Claim Index Action
2. Test the Mileage Recorder App

► Task 1: Complete the Mileage Claim Index Action

- In the **Index** action method for the **MileageClaimController**, locate the following line of code:

```
// Obtain all the mileage claims here
```

- Create a new enumerable list of **MileageClaim** objects named **claimsToDisplay**.
- Get the value of the **Session["SPAppWebUrl"]** property and store it in a new **string** variable named **appWebUrl**.
- In a **using** code block, create a new **ClientContext** object named **context**. Pass the **appWebUrl** to the **ClientContext** constructor.
- Use the **context** object to get the **Claims** list and store it in a variable named **claimsList**. Then load the **claimsList** object.
- Create a new **CamlQuery** object named **camlQuery**.

- Use the **camlQuery** object to get a collection of all the list items in the **claimsList**. Store the collection in a variable named **allClaims**.
- Load the **allClaims** object and then execute the CSOM query.
- Create a **foreach** loop that traverses all the list items in **allClaims**.
- Within the **foreach** loop, create a new **MileageClaim** object named **currentClaim**.
- Use the properties of the **sharepointClaim** object to set the following properties of the MVC object **currentClaim**:
 - Destination
 - ReasonForTrip
 - Miles
 - EngineSize
 - Amount
- Add the **currentClaim** object to the **claimsToDisplay** list.
- Pass the **claimsToDisplay** list to **Index** partial view that the action method returns.
- Save your changes.

► Task 2: Test the Mileage Recorder App

- Start the Mileage Recorder app in debugging mode and trust it.
- Make a new mileage claim. Use the following information:
- Destination: **Las Vegas**
- Reason for Trip: **Attending SharePoint Conference**
- Miles: **400**
- Engine Size: **2000**.



Note: The new claim is created and displayed with an automatically calculated value for **Amount**.



Note: If time permits, create several other claims to check that the Amount value is calculated as you expect.

- Stop debugging and close Visual Studio.

Results: A completed autohosted SharePoint app.

Question: In Exercise 2, how did you ensure that all pages in the remote web could locate the app web in SharePoint to read or write data?

Lesson 2

Using the Client-Side Object Model for JavaScript

The SharePoint JavaScript CSOM is a script library that you can use to ease the development of code that accesses SharePoint data. You can use the JavaScript CSOM from any JavaScript host application to read SharePoint items, documents, or properties and perform create, update, and delete operations. Because most web browsers can execute JavaScript, this is an excellent library to use when you want to access SharePoint data from a SharePoint hosted app.

Lesson Objectives

After completing this lesson, you will be able to:

- Identify scenarios in which you can use the JavaScript CSOM.
- Use the client context object to access information about the SharePoint site.
- Load objects in the current context and run queries.
- Read SharePoint data and present data to users.
- Create, update, and delete SharePoint items and documents.
- Handle any errors that arise from JavaScript CSOM operations.

Overview of the CSOM for JavaScript

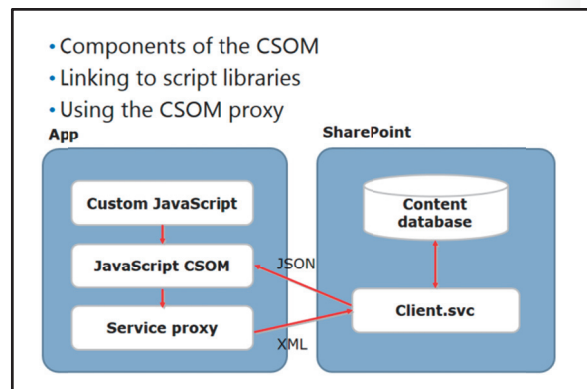
In earlier versions of SharePoint, most custom code ran within SharePoint server-side processes and therefore was able to call classes and methods in the server-side object model. In SharePoint 2010, a new client-side object model (CSOM) was introduced, which enabled developers to call core SharePoint functionality from browsers, Silverlight applications, and other clients. The CSOM executed requests by sending them to a Windows Communication Foundation (WCF) endpoint. In SharePoint 2013, because the new app model emphasizes client-side code, the CSOM has been greatly expanded: you can now use it to access both SharePoint core functionality and enterprise search, business connectivity services, managed metadata, social networking, enterprise content management, web content management, record management, and so on.

In fact, there are two CSOMs in SharePoint 2013: in this lesson, you will see the JavaScript CSOM. In the third lesson of this module, you will see the Managed CSOM, which can be used from .NET Framework clients, such as desktop applications.

Components of the CSOM

The classes and functions that make up the JavaScript CSOM are stored in two files: `sp.js` and `sp.runtime.js`. You can find these files in the SharePoint LAYOUTS folder. In SharePoint 2013, the functions in these JavaScript files call a WCF service named `Client.svc` that runs on all SharePoint web front-end servers. This service is found in the SharePoint ISAPI folder. The `Client.svc` service is also available in any SharePoint site by using the alias `/_api`. For example, if your site URL is:

`http://intranet.contoso.com`



Then the Client.svc service can be found at:

```
http://intranet.contoso.com/_api
```

The Client.svc service uses the server-side object model to communicate with the SharePoint content database.

Although you can use the JavaScript CSOM without any other script libraries, Visual Studio includes the jQuery library in all app projects by default because it eases development of server-side code. For example, by using the `$(document).ready()` jQuery function, you can ensure that no CSOM functions run before the Document Object Model (DOM) is fully loaded.

Linking to script libraries

In a page within a SharePoint app, use `<script>` tags to link to all the JavaScript libraries you need. You can either include these files in the app or link to their network locations.

Visual Studio includes the following `<script>` tags in app projects. These link to `sp.js` and `sp.runtime.js` stored on the SharePoint server.

Linking to the JavaScript CSOM

```
<script type="text/javascript" src="/_layouts/15/sp.runtime.js"></script>
<script type="text/javascript" src="/_layouts/15/sp.js"></script>
```

Alternatively, you can copy the `sp.runtime.js` and `sp.js` files into the `/Scripts` folder in your app and modify the link to that folder. However, in most cases, because the app must connect to SharePoint to access content, it is logical to download the CSOM from SharePoint as well.

Visual Studio includes the following `<script>` tags in app projects to link to jQuery.

Linking to jQuery

```
<script type="text/javascript" src="../Scripts/jquery-1.7.1.min.js"></script>
```

Notice that jQuery is included in the app `Scripts` folder by default. You might change this link in the following circumstances:

- *You want to use a later version of jQuery.* If you want to use a more up-to-date version of jQuery, obtain the script library from <http://jquery.com> and add it to the `Scripts` folder in your app. You must update the version number in the `<script>` tag.
- *You want to use jQuery from a Content Delivery Network (CDN).* A CDN is a set of high-performance servers distributed around the world. When you request a file from a CDN, a server physically close to your location responds. This can accelerate the performance of your application, but it requires an Internet connection. If you link to jQuery in a CDN, you can remove the jQuery file from your app's `Scripts` folder. Two possible CDNs that you can use for jQuery are:
 - <http://code.jquery.com>. This CDN is managed by the jQuery Foundation.
 - <http://ajax.aspnetcdn.com/ajax/jquery>. This CDM is managed by Microsoft.

In addition to links to script libraries, remember that you must link to your own JavaScript files, such as the `app.js` file that apps use by default.

Using the CSOM proxy

When you make one or more requests to SharePoint by using the JavaScript CSOM, you work with a JavaScript object that acts as a proxy for the Client.svc WCF service. These requests are grouped into a batch and sent in a single operation to the WCF service in the form of XML. The Client.svc works with the service object model to execute the request against the SharePoint content database. When there is data to return, such as a list of items in a list, the data is sent in the JavaScript Object Notation (JSON). You can

work with this form of data very easily in JavaScript to present it to users. For example, it is easy to loop through collections.

Using the Client Context Object

When you access SharePoint through the JavaScript object model, you must start by creating a client context object, much as you would when you write server-side SharePoint code. The context object represents the user's current location within SharePoint, and you can use it to gain access to the current site collection, site, and other objects.

The following code demonstrates how to create a context object by using the JavaScript CSOM. The context object is used to access the site collection and return its URL.

```
getSiteCollection = function () {
    context = new SP.ClientContext.get_current();
    siteCollection = context.get_site();
    context.load(siteCollection);
    context.executeQueryAsync(onSuccess, onFailure);
},
onSuccess = function () {
    alert("URL: " + siteCollection.get_url());
},
onFailure = function(sender, args) {
    alert("Could not obtain the site collection URL.");
}
```

Using the Client Context Object

```
"use strict";

var Contoso = window.Contoso || {};

Contoso.SiteCollection = function () {
    var siteCollection,
        context,

        getSiteCollection = function () {
            context = new SP.ClientContext.get_current();
            siteCollection = context.get_site();
            context.load(siteCollection);
            context.executeQueryAsync(onSuccess, onFailure);
        },

        onSuccess = function () {
            alert("Site Collection URL: " + siteCollection.get_url());
        },

        onFailure = function(sender, args) {
            alert("Could not obtain the site collection URL");
        }

    return {
        execute: getSiteCollection
    }
}();

$(document).ready(function () {
    Contoso.SiteCollection.execute();
});
```

In the preceding code, notice that the query is executed asynchronously and the data returned is processed in the separate **onSuccess()** callback function. This pattern ensures that long-running queries do not halt a process and freeze the browser.

There are a range of other methods, called in the same way as you call **get_site()** in the preceding example, that enable you to access other information from the client context object. These include:

- *get_web()*. This method returns the current SharePoint site (SPWeb). This may be the top-level site in a site collection, or a subsite. In a SharePoint-hosted app, the current web is always the app web.
- *get_web().get_currentUser()*. This method, which you must access through the current SharePoint site, returns the identity of the user that made the request. You can use the user to access profile properties or permissions.
- *get_web().get_siteGroups()*. This method returns any SharePoint groups that have been defined for the current site.

Loading Objects and Running Queries

The JavaScript library uses a proxy class to interact with the SharePoint Client.svc service and execute batches of queries. This architecture means that you must take two steps to perform a data operation in SharePoint:

- *Load an object.* When you load an object, you set it up in the client context object for retrieval from the server. You can load multiple objects to batch them together for retrieval in a single XML query. You can use two methods to load objects:
 - *Load()*. This method specifies an object or collection to retrieve, such as a SharePoint site or the items in a list. In addition to the object to retrieve, you can specify the properties of the object to retrieve. Do this by passing the names of the properties as string parameters to the **Load()** method. If you do not specify any property names, all the properties of the object are loaded.
 - *LoadQuery()*. This method specifies information to retrieve by defining a Language Integrated Query (LINQ) request.

- Loading objects
 - The context.load() method
 - The context.loadQuery() method
- Executing operations
 - Asynchronous
 - Synchronous

Execute the operation. When you call the **executeQueryAsync()** method, the batched XML requests are sent to the Client.svc service. Because this is an asynchronous operation, the browser does not halt while it waits for a response, and you must define callback functions that run when a response is received from the server.

The following code shows how to load an object into the client context and how to execute the operation asynchronously and display the results.

Loading an Object and Running a Query

```
"use strict";

var Contoso = window.Contoso || {};

Contoso.SiteCollection = function () {
    var siteCollection,
        context,

        getSiteCollection = function () {
            context = new SP.ClientContext.get_current();
            siteCollection = context.get_site();
            context.load(siteCollection, "Title", "ServerRelativeUrl");
            context.executeQueryAsync(onSuccess, onFailure);
        },

        onSuccess = function () {
            alert("Site Collection URL: " + siteCollection.get_url());
        },

        onFailure = function(sender, args) {
            alert("Could not obtain the site collection URL");
        }

    return {
        execute: getSiteCollection
    }
}();

$(document).ready(function () {
    Contoso.SiteCollection.execute();
});
```

Demonstration: How to Use load and loadQuery

In this demonstration, you will see how to:

- Use the **load** method from the JavaScript CSOM to load items.
- Loop through results by using an enumerator.
- Use the **loadQuery** method from the JavaScript CSOM to load items.
- Loop through results by using the **forEach** method.

Demonstration Steps

- Connect to the **20488B-LON-SP-08** virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Start **File Explorer**.
- Navigate to **E:\Democode\Starter\LoadItemsDemo**.
- Double-click **LoadItemsDemo.sln**.
- If the **How do you want to open this type of file (.sln)?** dialog box appears, click **Visual Studio 2012**. In Solution Explorer, expand the **Scripts** folder, and then double-click **App.js**.

- Locate the following code:

```
var getItemsWithLoad = function () {  
};
```

- Place the cursor within the located function.
- Type the following code:

```
var discussionList = context.get_web().get_lists().getByTitle('Discussion');  
context.load(discussionList);
```

- Press Enter.
- Type the following code:

```
var items = discussionList.getItems('');
```

- Press Enter.
- Type the following code:

```
context.load(items);
```

- Press Enter.
- Type the following code:

```
context.executeQueryAsync(function () {  
});
```

- Place the cursor within the anonymous function you just created.
- Type the following code:

```
var enumerator = items.getEnumerator();
```

- Press Enter.
- Type the following code:

```
while (enumerator.moveNext()) {  
}
```

- Place the cursor within the **while** loop you just created.
- Type the following code:

```
var item = enumerator.get_current();
```

- Press Enter.
- Type the following code:

```
alert("Discussion Item: " + item.get_item("Title"));
```

- Locate the following code:

```
var getItemsWithLoadQuery = function () {  
};
```

- Place the cursor within the located function.

- Type the following code:

```
var discussionList = context.get_web().get_lists().getByTitle('Discussion');
context.load(discussionList);
```

- Press Enter.
- Type the following code:

```
var items = discussionList.getItems('');
```

- Press Enter.
- Type the following code:

```
var results = context.loadQuery(items);
```

- Press Enter.
- Type the following code:

```
context.executeQueryAsync(function () {
});
```

- Place the cursor within the anonymous function you just created.
- Type the following code:

```
results.forEach(function (item) {
})
```

- Place the cursor within the **forEach** loop you just created.
- Type the following code:

```
alert("Discussion Item: " + item.get_item("Title"));
```

- On the **DEBUG** menu, click **Start Debugging**.
- If the **Windows Security** dialog box appears, in the **Username** box, type **Administrator**.
- In the **Password** box, type **Pa\$\$w0rd**.
- Click **Get Items with Load**.
- Click **OK** for each result.
- Click **Get Items with LoadQuery**.
- Click **OK** for each result.
- Close Internet Explorer.
- Close Visual Studio.

Reading SharePoint Data

The JavaScript CSOM makes it easy to obtain the lists and libraries in a SharePoint site, the items in a list, and a specific item in a list. These methods enable you to display data to the user, for example by using jQuery to update a web page element. You can also use these methods when you want to display current values for the user to edit.

You can use the following methods to read data from SharePoint:

- *get_lists()*. This method is available from a SharePoint site object and returns all the lists in the site.
- *get_lists().getByTitle()*: You can use the **getByTitle** method on any collection of lists to obtain a list with a specific title.
- *getItems()*. You can use the **getItems** method on any list to return all the items in the list. Alternatively, by passing a Collaborative Application Markup Language (CAML) query to *getItems*, you can filter and order the items returned.
- *getItemById()*. If you already have the ID of the relevant item, you can use **getItemById** to return the item.

- Reading all the lists in a site
- Getting a list by title
- Getting items in a list by using a CAML query
- Getting an item by ID

This example shows how use the **get_lists()** method, the **get_lists().getByTitle()** method, and the **getItems()** method from the JavaScript CSOM. The call to *getItems()* uses a CAML query to order the results alphabetically.

Querying for Data in a SharePoint Site

```
"use strict";

var Contoso = window.Contoso || {};

Contoso.ReadData = function () {
    var site,
        alllists,
        suggestionsList,
        items,
        context,

        getData = function () {
            context = new SP.ClientContext.get_current();
            site = context.get_web();
            context.load(site);

            // Now we have the SharePoint site, load the lists in the site.
            alllists = site.get_lists();
            context.load(alllists);

            // Set up a CAML query to load items in the Suggestions list.
            var query = "<View><Query><OrderBy><FieldRef Name='Title' /></OrderBy></Query>" +
                "<ViewFields><FieldRef Name='ID' />< FieldRef Name='Title' /></ViewFields></View>";
            var camlQuery = new SP.CamlQuery();
            camlQuery.set_viewXml(query);

            // Load the suggestions list.
            suggestionsList = site.get_lists().getByTitle("Suggestions");
            context.load(suggestionsList);
```

```

        // Get the items in the suggestions list by using the CAML query.
        items = suggestionsList.getItems(camlQuery);
        context.load(items);

        context.executeQueryAsync(onSuccess, onFailure);
    },

    onSuccess = function () {
        // Tell the user about the lists.
        var message = "The lists are as follows: ";
        var listEnumerator = allLists.getEnumerator();
        while (listEnumerator.moveNext()) {
            message += listEnumerator.get_current().get_title() + " ";
        }
        alert(message);

        // Tell the user about the items in the suggestions list.
        message = "The suggestions are as follows: ";
        var itemEnumerator = items.getEnumerator();
        while (itemEnumerator.moveNext()) {
            message += itemEnumerator.get_current().get_item('Title') + " ";
        }
        alert(message);
    },

    onFailure = function(sender, args) {
        alert("Could not obtain the data in the current site");
    };

    return {
        execute: getData
    };
}());

$(document).ready(function () {
    Contoso.ReadData.execute();
});

```

In the preceding example, notice that an enumerator object is used to loop through the lists and the items.

Changing SharePoint Data

When you create new items, update existing items, or delete items in a SharePoint list by using the JavaScript CSOM, you must still use a client context object, and load and execute a query just as you would for methods that read data. Use the following approaches to edit SharePoint items:

- *To create a new list.* Get the lists collection for the site, and then call the **add()** method on that lists collection. You must create a new **ListCreationInformation** object and pass it to the **add()** method. You can use the **ListCreationInformation** object to set properties such as a title for the list.

- Creating a new list in a site
- Creating a new item in a list
 - Using `SP.ListItemCreationInformation`
- Updating an existing item in a list
- Deleting an item from a list

- *To create a new item in a list.* Get a specific list, and then call the **addItem()** method on that list. You must create a new **ListItemCreationInformation** object and pass it to the **addItem()** method. You can set properties on the item by calling the **item.set_item()** method.
- *To update an existing item in a list.* Get a single item in a list by using the **getItemById()** method on the list. After you have the item, use the **set_item()** method to update its fields, and then call **update()** to save changes before you call **context.executeQueryAsync()**.
- *To delete an existing item in a list.* Get a single item in a list by using the **getItemById()** method on the list. After you have the item, call **deleteObject()** before you call **context.executeQueryAsync()**.

The following example defines an object in the Contoso namespace with methods for creating, updating and deleting items in a list named Suggestions. You could call these methods, for example, when a user clicks a submit input control.

Editing Items in SharePoint Lists

```

"use strict";

var Contoso = window.Contoso || {};

// This is an object to store the items in the Suggestions list.
Contoso.Suggestions;

Contoso.SuggestionsList = function () {

    createSuggestion = function (subject, feedback) {
        // Get the client context and suggestions list.
        var context = new SP.ClientContext.get_current();
        var list = context.get_web().get_lists().getByTitle("Suggestions");
        context.load(list);

        // Create the new item and set fields.
        var listItemCreationInfo = new SP.ListItemCreationInformation();
        var newItem = list.addItem(listItemCreationInfo);
        newItem.set_item("Subject", subject);
        newItem.set_item("Feedback", feedback);

        // Remember to call update before executing the query.
        newItem.update();
        context.executeQueryAsync(onSuccess, onError);
    },

    updateSuggestion = function (id, subject, feedback) {
        // Get the client context and suggestions list.
        var context = new SP.ClientContext.get_current();
        var list = context.get_web().get_lists().getByTitle("Suggestions");
        context.load(list);

        var itemToUpdate = list.getItemById(id);
        itemToUpdate.set_item("Subject", subject);
        itemToUpdate.set_item("Feedback", feedback);

        // Remember to call update before executing the query.
        newItem.update();
        context.executeQueryAsync(onSuccess, onError);
    },

    removeItem = function (id) {
        // Get the client context and suggestions list.
        var context = new SP.ClientContext.get_current();
        var list = context.get_web().get_lists().getByTitle("Suggestions");
        context.load(list);

        // Get the item from its ID.

```

```

    var itemToDelete = list.getItemById(id);

    // Delete it before executing the query.
    itemToDelete.deleteObject();
    context.executeQueryAsync(onSuccess, onError);
  },

  onSuccess = function () {
    alert("The operation completed successfully");
  },

  onError = function (sender, args) {
    alert("The operation caused an unexpected error");
  };

  return {
    create: createSuggestion,
    update: updateSuggestion,
    delete: removeSuggestion
  };
} ();

```

Handling Server-Side Errors

In JavaScript, it is easy to handle errors that arise on the client by using the try/catch/finally construction. When you run a query on the server, you specify the name of two functions in the call to **executeQueryAsync**: The first function is called if the query succeeds, and the second function is called if the query fails. This failure function is a good place to handle problems that arise on the server. You have already seen failure callback functions in previous code examples.

However, the failure callback function is only called when the server responds to the client. The failure callback function executes on the client and has limited access to server information. Remember that, when you use the CSOM, you can batch many queries into a single call to the Client.svc service. If an operation causes an error, the whole batch fails and you have no way to ensure that subsequent operations complete. You also cannot instruct the server how to respond to errors on the client.

To solve these issues, CSOM includes a mechanism with which you can send error-handling instructions to the server along with your batched query. The mechanism is implemented by the **ExceptionHandlingScope** object. This object enables you to define a try/catch/finally construction that applies while the query executes on the server.

- By default, a server-side error causes the whole batch of operations to fail.
- To avoid this, you can use an exception handling scope to define server-side try/catch/finally blocks

```

var e = new SP.ExceptionHandlingScope(context);
var s = e.startScope();
var t = e.startTry();
// This is the try block.
t.dispose();
var c = e.startCatch();
// This is the catch block.
c.dispose();

```

The following code shows how to create an object that uses an exception handling scope to implement try/catch/finally code on the server.

Using an Exception Handling Scope to Resolve Server-Side Errors

```

"using strict";

var Contoso = window.Contoso || {};

Contoso.ErrorScope = function() {
  var site,
      scope = function() {
        var context = new SP.ClientContext.get_current();

        // Create and start the exception handling scope.
        var e = new SP.ExceptionHandlingScope(context);
        var s = e.startScope();

        // Define the try block.
        var t = e.startTry();

        // Place code that may generate exceptions here.

        t.dispose(); // This call to dispose marks the end of the try block.

        // Define the catch block.
        var c = e.startCatch();

        // Place code that handles exceptions here.

        c.dispose(); //This call to dispose marks the end of the catch block

        // Define the finally block.
        var f = e.startFinally();

        // Place code that should execute regardless of errors here.

        f.dispose(); // This call to dispose marks the end of the finally block.

        // You must also dispose of the exception handling scope before you execute the
query.
        s.dispose();
        context.executeQueryAsync(onSuccess, onFailure);
      },
      onSuccess = function () {
        alert("Query was successful");
      },
      onFailure = function () {
        alert("Query failed");
      }

  return {
    execute: scope
  }
} ();

$(document).ready(function () {
  Contoso.ErrorScope.execute();
});

```

Lesson 3

Using the REST API with JavaScript

As you have seen, the JavaScript CSOM provides a broad range of client-side objects and methods that you can use to access and manipulate data stored in SharePoint. Alternatively, because the Client.svc service is RESTful, you can call it directly by making GET, POST, MERGE, PUT, and DELETE HTTP requests to simple URLs in the SharePoint farm. Furthermore, jQuery includes several functions that assist you in formulating RESTful queries and processing the results. In this lesson, you will see how to use the REST API directly.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand when to use the REST API to access SharePoint from a SharePoint app.
- Read SharePoint data by issuing GET requests to the REST API.
- Change SharePoint data by issuing POST, MERGE, PUT, and DELETE requests to the REST API.
- Handle errors that arise during REST API operations.

Overview of the REST API

The JavaScript CSOM that you saw in the previous lesson is a set of JavaScript functions that sends calls to the Client.svc Windows Communication Foundation (WCF) service. This web service can also be used directly by formulating and sending your own calls instead of relying on the CSOM library. It is easy to create and send such calls because the Client.svc library is a RESTful web service. RESTful web services have the following characteristics:

- *Logical URLs.* The REST service itself has a URL. By specifying a URI within the service's URL, you can define the object or collection of objects to work with. For example, in SharePoint, the Client.svc is available in the /_api subfolder of any site. To select a list named "Discussion," you could use the URI /_api/web/lists/getbytitle('Discussion').
- *Support for HTTP verbs.* To specify the operation you want to perform in a RESTful service, you use HTTP verbs. For example, to read information, you use the GET verb. To remove an item, you use the DELETE verb. By using a combination of a URL with an HTTP verb, you specify the object or objects to select and the operation to perform on those objects.
- *Support for OData.* RESTful services use the Open Data Protocol (OData) to create queries for data. For example, you can use an OData operator such as \$select in a URL to filter the collection of objects that your operation will act on. The OData protocol also specifies that RESTful services should return data in the Atom Publishing Protocol (AtomPub) or JavaScript Object Notation (JSON) formats.

One advantage of RESTful services is that you can easily explore them with a web browser. This is because a RESTful service uses the HTTP protocol, with user-friendly URLs that you can easily enter manually, and returns XML data that the browser can display. This is extremely helpful during development because you can use a web browser such as Internet Explorer to query data and examine the structure of the response.

RESTful services:

- Use logical URLs to specify objects
- Use HTTP verbs to specify operations
- Use OData to exchange data

The SharePoint Client.svc service is a REST API

```
$getJSON(
  "http://intranet.contoso.com/_api/web",
  function (data) {
    alert("The SharePoint site is called: " + data.d.Title);
  }
)
```


For example, you can obtain the properties of a SharePoint website by entering the following URL in the Internet Explorer address bar:

`http://intranet.contoso.com/_api/web`

Another advantage for JavaScript developers is that the jQuery library has helpful functions that you can use to formulate REST calls quickly. Because you can request JSON data, it is also very easy to process the results.

The following code show how to use the jQuery **getJSON()** function to obtain information about the current SharePoint site from the REST API.

Issuing a REST API Request with jQuery

```
$(document).ready( function () {
    $.getJSON(
        "http://intranet.contoso.com/_api/web",
        function (data) {
            alert('The SharePoint site is called: ' + data.d.Title);
        }
    );
});
```

SharePoint REST API URLs

Before you begin programming against the SharePoint REST API, it is essential to understand how REST API URLs are constructed. With this understanding, you can easily formulate the URLs to call any REST API method and pass parameters. The following examples work with a SharePoint site at the following URL:

`http://intranet.contoso.com/`

Therefore, all REST API operations must start with the following URL, which is the location of the Client.svc service:

`http://intranet.contoso.com/_api/`

- Address of the REST API
- URLs for common SharePoint objects
- Using OData operators

```
http://intranet.contoso.com
/_api/web/lists/getbytitle("MyList")/items
?$select=ID,Title
&$order=Title
&$filter=startswith(Title,"A")
```

Common SharePoint objects

The following table shows the URLs you must use to access SharePoint site collections, sites, lists, and other objects.

URL	Notes
<code>http://intranet.contoso.com/_api/site</code>	This returns the site collection object and its properties. Note that this URL does not return the SharePoint site (SPWeb).
<code>http://intranet.contoso.com/_api/site/owner</code>	This returns the SP.User object for the user who created the site collection.
<code>http://intranet.contoso.com/_api/site/url</code>	This returns the full URL for

MCT USE ONLY - STUDENT USE PROHIBITED

URL	Notes
	the site collection, including the fully qualified domain name, port number, and path.
<code>http://intranet.contoso.com/_api/web</code>	This returns the SharePoint site object and its properties.
<code>http://intranet.contoso.com/_api/web/title</code>	This returns the title of the SharePoint site
<code>http://intranet.contoso.com/_api/web/currentUser</code>	This returns the SP.User object for the current user.
<code>http://intranet.contoso.com/_api/web/lists</code>	This returns a collection of all the lists and document libraries in the current SharePoint site. Lists are only returned if the current user has at least read access to them.
<code>http://intranet.contoso.com/_api/web/lists/getbytitle("MyList")</code>	This returns the list with the title "MyList."
<code>http://intranet.contoso.com/_api/web/lists/getbytitle("MyList")/items</code>	This returns all the items in the "MyList" list.

Controlling the objects returned


Many of the URLs listed in the previous table return collections of objects. When you have large content databases, lists can contain thousands of items and sites can contain many lists. Therefore, it is essential to be able to control the collection of objects that a REST API call returns. You can do this by using OData query operators. The operators you can use include the following:

- *\$select*. Use the \$select operator to specify the fields that the query will return. If you do not use the \$select operator, all the fields are returned except any fields that are likely to be very large, such as attached files.
- *\$order*. Use the \$order operator to specify one or more fields by which to order the results.
- *\$filter*. Use the \$filter operator to set conditions. Only items that satisfy the conditions will be returned. For example, you could use the \$filter operator to return only items whose Title starts with an "A".
- *\$top*. Use the \$top operator to specify a number of objects to return.
- *\$skip*. Use the \$skip operator to specify a number of objects to omit from the results. \$top and \$skip are used to implement paging. For example, to show the second page of results, when there are ten results on each page, you would specify \$top=10 and \$skip=10.

The following table shows some REST API URLs that use the OData operators to control the objects returned:

URL	Notes
<code>/_api/web/lists/getbytitle("MyList")/items?\$select=ID,Title</code>	This returns all the items in the MyList list. However, only the ID and Title fields

URL	Notes
	are returned for each item.
<code>/_api/web/lists/getbytitle("MyList")/items?\$select=ID,Title&\$order=Title</code>	This returns the same items and fields as the previous query but items are ordered by title.
<code>/_api/web/lists/getbytitle("MyList")/items?\$filter=startswith(Title, "A")</code>	This returns only items in the MyList list that start with the letter A.

 **Note:** Try exploring your SharePoint REST API by typing URLs similar to those in the preceding tables into the Internet Explorer address bar. Substitute the URL of your own SharePoint site. By examining the AtomPub XML returned in each case, you can become proficient in formulating REST API URLs. These skills will be helpful when you write JavaScript code that call the REST API.

Reading Data

If you are writing any client-side code that reads data from SharePoint, you must call the REST API from JavaScript and use the HTTP GET verb. For example, you can issue such calls when you write a SharePoint hosted app, which can only execute client-side code. Three things can help you to formulate and issue such requests:

- *The `_spPageContextInfo.webServerRelativeUrl` property.* It is bad practice to hard-code the URL of your SharePoint site into JavaScript files because, when you deploy an app or when uses install it from a catalog, the site URL changes. Instead, you look up the server-relative URL for the current site. The `_spPageContextInfo.webServerRelativeUrl` property enables you to look up this value.
- *The jQuery `$.getJSON()` function.* jQuery includes two functions that help you to formulate and issue REST requests. The `getJSON()` function is simple and quick to use.
- *The jQuery `$.ajax()` function.* The `ajax()` function is more complex than the `getJSON()` function, but you have more control over the request.

- Using the `_spPageContextInfo.webServerRelativeUrl` property
- The jQuery `getJSON()` function
- The jQuery `ajax()` function

In the following example, the `_spPageContextInfo.webServerRelativeUrl` property is used to formulate two REST API URLs. The first URL is used with the `getJSON()` function to find out about the current website. The second URL is used with the `ajax()` function to find out about the current user.

Reading Data from the REST API

```

$(document).ready( function() {
    // Formulate the URL to get the current website.
    var websiteRequestURL = _spPageContextInfo.webServerRelativeUrl + "/_api/web"
    // Formulate the URL to get the current user.
    var userRequestURL = _spPageContextInfo.webServerRelativeUrl + "/_api/web/currentuser"

    // Use getJSON() to find out about the current website.
    $.getJSON(websiteRequestURL, function (data) {
        alert("The current web site has the title " + data.d.Title);
    });
    // Use ajax() to find out about the current user.
    $.ajax({
        url: userRequestURL,
        type: "GET",
        headers: {
            "accept": "application/json;odata=verbose"
        },
        success: function (data) {
            alert("The current user has the name " + data.d.Title);
        },
        error: function (err) {
            alert("There was an error obtaining the user name");
        }
    });
});

```

Creating and Updating Data

When you want to create a new item, update an existing item, or delete an item through the REST API, you specify the location of the item by using URLs just as you do when you read information from an item. However, you must use different HTTP verbs:

- *POST*. You can use POST when you want to create a new item.
- *PUT*. You can use PUT when you edit an existing item. If you use PUT, you must specify all the properties of the item—both those properties that have changed and those properties that have remained the same.
- *PATCH*. You can use PATCH when you edit an existing item. If you use PATCH, you can specify only those properties that have changed.
- *MERGE*. You can use MERGE when you edit an existing item. MERGE is similar to PATCH, but PATCH is used more frequently by convention.
- *DELETE*. You can use DELETE when remove an item.

- Creating new items
 - Formulate a URL to the parent list in the REST API
 - Use the POST verb
- Updating existing items
 - Formulate a URL to the item itself
 - Use the PATCH, MERGE, or PUT verbs
- Deleting items
 - Formulate a URL to the item itself
 - Use the DELETE verb
- Always pass a form digest

To issue REST calls with these verbs from client-side code, you cannot use the jQuery **getJSON()** function, because this function only sends the GET verb. Instead, you must use the **ajax()** function.

All data modification operations require a form digest. A form digest is a hash of page content that proves the request comes from the original page sent from the server. This increases security by preventing any request not associated with a page sent from SharePoint. To send the form digest with your REST request, get the contents of the page element with ID "`__REQUESTDIGEST`" and send it in the `X-RequestDigest` header.

The following example shows how to create a **SuggestionsList** object with methods that create, update, and delete suggestions. You can call these methods, for example, when users click buttons on the page.

```
"use strict";

var Contoso = window.Contoso || {};

Contoso.SuggestionsList = function () {

    create = function (subject, feedback) {

        // To create an item, formulate a URL to the parent list.
        var listUrl = _spPageContextInfo.webServerRelativeUrl +
            "/_api/web/lists/getByTitle('Suggestions')/items";

        // Store the form digest.
        var formDigest = $('#__REQUESTDIGEST').val();

        $.ajax({
            url: listUrl,
            type: "POST",
            data: JSON.stringify({
                '__metadata': { 'type': 'SP.Data.SuggestionsListItem' },
                'Subject': subject,
                'Feedback': feedback
            }),
            headers: {
                "accept": "application/json;odata=verbose",
                "X-RequestDigest": formDigest
            },

            success: function () {
                alert("Successfully created the suggestion");
            },

            error: function (err) {
                alert("Could not create the suggestion");
            }
        });
    },

    update = function (id, subject, feedback) {
        // To update an item, formulate a URL to the item itself.
        var listUrl = _spPageContextInfo.webServerRelativeUrl +
            "/_api/web/lists/getByTitle('Suggestions')/getItemById('" + id + "')";

        // Store the form digest.
        var formDigest = $('#__REQUESTDIGEST').val();

        $.ajax({
            url: listUrl,
            type: "POST",
            data: JSON.stringify({
                '__metadata': { 'type': 'SP.Data.SuggestionsListItem' },
                'Subject': subject,
                'Feedback': feedback
            }),
            headers: {
                "accept": "application/json;odata=verbose",
```

```
        "X-RequestDigest": formDigest,
        "IF-MATCH": "**",
        "X-Http-Method": "PATCH"
    },
    success: function () {
        alert("Successfully updated the suggestion");
    },
    error: function (err) {
        alert("Could not update the suggestion");
    }
},

delete = function (id) {

    // To delete an item, formulate a URL to the item itself.
    var listUrl = _spPageContextInfo.webServerRelativeUrl +
        "/_api/web/lists/getByTitle('Suggestions')/getItemById('" + id + "')";

    // Store the form digest.
    var formDigest = $('#_REQUESTDIGEST').val();

    $.ajax({
        url: listUrl,
        type: "DELETE",
        headers: {
            "accept": "application/json;odata=verbose",
            "X-RequestDigest": formDigest,
            "IF-MATCH": "**"
        },
        success: function () {
            alert("Successfully updated the suggestion");
        },
        error: function (err) {
            alert("Could not update the suggestion");
        }
    }

    return {
        createSuggestion: create,
        updateSuggestion: update,
        deleteSuggestion: delete
    }
} ();
```

Lab B: Using the REST API with JavaScript

Scenario

The management team at Contoso has asked that you extend the Site Suggestions app to include more sophisticated functionality. In particular, they want users to be able to vote on suggestions made by others. Votes can be positive or negative, and the net votes should be displayed with each item.

Objectives

After completing this lab, you will be able to:

- Create list relationships in a SharePoint app.
- Add voting functionality to a SharePoint app.
- Filter list items in a SharePoint app.

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-08
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Creating List Relationships

Scenario

To add voting functionality to your app, you must create a list in the app web that will store votes. You must also link each vote to the suggestion it relates to. In this exercise, you will create the site columns, content type, and list instance to support voting.

The main tasks for this exercise are as follows:

1. Add Site Columns
2. Add the Vote Content Type
3. Add the Votes List

► Task 1: Add Site Columns

- Start the virtual machine, and log on with the following credentials:
 - Virtual Machine: **20488B-LON-SP-08**
 - User name: **CONTOSO\Administrator**
 - Password: **Pa\$\$w0rd**
- Open the following Visual Studio solution:
 - E:\Labfiles\LabB\Starter\SiteSuggestionsApp.sln.
- Create a new site column named **Positive**. Use the following information:
 - Display Name: **Positive?**
 - Type: **Boolean**
 - Required: **TRUE**
- Create a new site column named **SuggestionLookup**. Use the following information:
 - Display Name: **Suggestion Lookup**

- Type: **Lookup**
- List: **Lists/Suggestions**
- Show Field: **Subject**
- Required: **TRUE**
- Save your changes.
- ▶ **Task 2: Add the Vote Content Type**
- Add a new content type named **Vote** to the site suggestions app.
- Add the following columns to the **Vote** content type:
 - Positive?
 - Suggestions Lookup
- Set the description of the **Vote** content type to **This content type defines a Vote for the Site Suggestions app**.
- Save your changes.
- ▶ **Task 3: Add the Votes List**
- Add a new list named **Votes** to the Site Suggestions app.
- In the **Votes** list, remove the **Item** and **Folder** content types, and then add the **Vote** content type.
- Remove the **Title** column from the **Votes** list.
- Set the **Description** for the **Votes** list to **This list stores votes linked to the corresponding suggestions**.
- Save your changes.

Results: A SharePoint-hosted app with a lookup site column that links items in the Suggestions list with items in the Votes list.

Exercise 2: Add Vote Recording

Scenario

In this exercise, you will enable users to register a positive or negative vote for each suggestion. The code you will add uses the jQuery ajax() function to call the REST API.

The main tasks for this exercise are as follows:

1. Add the recordVote Function
2. Call the recordVote Function

▶ Task 1: Add the recordVote Function

- In the **App.js** script, add a new variable named **recordVote** that equals an anonymous function that accepts a single parameter named **positive**.
- In the new anonymous function, store the REST path to the **Votes** list in a new variable named **votesListURL**.
- Use jQuery to store the value of the page element with ID **__REQUESTDIGEST** in a new variable named **formDigest**.

- Create a new call to the jQuery **ajax()** function.
 - Within the **ajax()** function, set the **url** property to **votesListURL** and the **type** property to **"POST"**.
 - For the **data** property, use the **JSON.stringify()** function to pass an object with the following properties:
 - Positive: **positive**
 - SuggestionLookupId: **currentSuggestion.get_item('ID')**
 - **__metadata**: an object with type **SP.Data.VotesListItem**
 - For the **headers** property, pass an anonymous object with the following properties:
 - **accept**: **application/json;odata=verbose**
 - **content-type**: **application/json;odata=verbose**
 - **X-RequestDigest**: **formDigest**
 - For the **success** property, pass an anonymous function that displays an alert with the message **Thank you for your vote**.
 - For the **error** property, pass an anonymous function that displays an alert with the message **Could not register your vote. Please try again later**.
- **Task 2: Call the recordVote Function**
- Open the **Default.aspx** webpage.
 - In the **onclick** event for the **Like** link, call the **Contoso.SuggestionsApp.record_vote()** function and pass **true** for the **positive** parameter.
 - In the **onclick** event for the **Dislike** link, call the **Contoso.SuggestionsApp.record_vote()** function and pass **false** for the **positive** parameter.
 - Save your changes.

Results: A simple SharePoint hosted app in which users can add positive or negative votes to each site suggestion.

Exercise 3: Display Votes for Each Suggestion

Scenario

Now that users can register positive and negative votes for each suggestion, you must add code that displays the net votes with the suggestion that they relate to. The net votes value is the number of positive votes for a suggestion minus the number of negative votes.

The main tasks for this exercise are as follows:

1. Add the voteCount Function
2. Call the voteCount Function
3. Test the Suggestions App

► Task 1: Add the voteCount Function

- In the **App.js** script, add a new variable named **voteCount** that equals an anonymous function that takes no parameters.
- In the new anonymous function, create a new variable named **netVotes** and set its value to **zero**.

- Create a new variable named **votesListURL** and set its value to **_spPageContextInfo.webServerRelativeUrl**.
- To the value of the **votesListURL** variable, append the REST API path to the items in the Votes list.
- To the value of the **votesListURL** variable, append a query string that filters votes. The filter should return only those votes whose **SuggestionLookup** value equals **currentSuggestion.get_item('ID')**.
- Create a new call to the jQuery **ajax()** function.
- Within the **ajax()** function, set the **url** property to **votesListURL** and the **type** property to **"GET"**.
- For the **headers** property, pass an anonymous object with the following property:
 - accept: **application/json;odata=verbose**
- For the **success** property, pass an anonymous function that accepts a single parameter named **data**.
- In the new function, use the jQuery **each()** function to loop through the **data.d.results** collection. For each result, execute an anonymous function that accepts an index named **i** and a result named **result**.
- In the new function, if the **result.Positive** property is **true**, increase **netVotes** by one. Otherwise decrease **netVotes** by one.
- After you loop through all the results, set the content of the page element with ID **votes-count** to **netVotes**.
- For the **error** property of the **ajax()** function, pass an anonymous function that accepts a parameter named **err**.
- In the **error** function, alert the user that the votes could not be counted.
- Save your changes.
- ▶ **Task 2: Call the voteCount Function**
 - In the **onDisplaySuggestionSuccess** function, immediately before the call to the **fadeOut()** function, call the **voteCount()** function with no parameters.
 - Save your changes.
- ▶ **Task 3: Test the Suggestions App**
 - Start the Suggestions app in debugging mode. You are prompted to log on, use the following credentials:
 - Username: **Administrator**
 - Password: **Pa\$\$w0rd**
 - Create a new Suggestion with the following properties:
 - Subject: **A good idea**
 - Feedback: **This is to test positive votes.**
 - Create a new Suggestion with the following properties:
 - Subject: **An average idea**
 - Feedback: **This is to test net votes.**
 - Create a new Suggestion with the following properties:
 - Subject: **A bad idea**
 - Feedback: **This is to test negative votes.**

- Register two positive votes for the suggestion titled **A good idea**.
- Register one positive vote and one negative vote for the suggestion titled **An average idea**.
- Register two negative votes for the suggestion titled, **A bad idea**.
- Click each suggestion in the list and check that the net vote for each is as expected.
- Stop debugging and close Visual Studio.

Results: A SharePoint-hosted app that uses a filtered REST API call to obtain and process the votes for each suggestion.

Question: In the code that retrieves and displays votes, how did you ensure only votes for the current suggestion are retrieved from SharePoint?

Question: In Exercise 2, you passed a form digest with REST request. Why was this form digest unnecessary in the REST request you formulated in Exercise 3?

Module Review and Takeaways

In this module, you have seen three technologies that you can use in apps for SharePoint and other packages to read and write information to and from SharePoint. The technologies are:

- *The .NET Managed CSOM.* You can use this object model from any .NET application. This model is often used in cloud-hosted applications built with ASP.NET to access SharePoint.
- *The JavaScript CSOM.* You can use this object model from any JavaScript interpreter. JavaScript code is most often executed within a browser, such as Internet Explorer.
- *The REST API.* You can call the REST API from any client that can call a web service.

Common Issues and Troubleshooting Tips

Common Issue	Troubleshooting Tip
A REST API request that modifies a SharePoint item generates an error.	

Review Question(s)

Question: You are writing a SharePoint-hosted app and you want to access the items in a SharePoint list. Which of the following technologies can you use to write your code:

The JavaScript CSOM

The REST API

The .NET Managed CSOM

Module 9

Developing Remote-hosted Apps for SharePoint

Contents:

Module Overview	9-1
Lesson 1: Overview of Remote-Hosted Apps	9-2
Lesson 2: Configuring Remote-Hosted Apps	9-9
Lab A: Configuring a Provider-Hosted SharePoint App	9-15
Lesson 3: Developing Remote-Hosted Apps	9-19
Lab B: Developing a Provider-Hosted SharePoint App	9-25
Module Review and Takeaways	9-28

Module Overview

When you develop a SharePoint-hosted app, all webpages and other resources are stored in the app web within the SharePoint content database, which means you don't have to set up a web server or configure a connection and authentication mechanism with SharePoint. When you develop a cloud hosted app, webpages and other resources are stored in a remote web, which is a website on a web server outside SharePoint. Therefore, you must set up the remote web server and make sure that code in the remote web can connect and authenticate with SharePoint. Do not let this extra configuration discourage you from creating cloud-hosted apps because there are many things you can build in a cloud-hosted app that are not possible in a SharePoint-hosted app. You will see this enhanced flexibility that cloud-hosted apps can achieve in this module.

Objectives

After completing this module, you will be able to:

- Describe how remote-hosted apps work and how to configure the permissions and cross-domain calls that they may require.
- Configure apps for hosting on Windows Azure or remote servers.
- Develop apps for hosting on Windows Azure or remote servers.

Lesson 1

Overview of Remote-Hosted Apps

When you build a remote-hosted app, you must create a remote website and enable it to communicate securely with SharePoint for access to lists, libraries, search, and other facilities. You must also make sure that, when multiple customers use your app, they cannot see each other's sensitive data. These requirements are solved in different ways, depending on whether you decide to create an auto-hosted app or a provider-hosted app. In this lesson, you will see how auto-hosted apps and provider-hosted apps solve these requirements and the advantages and disadvantages of each approach.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand how remote-hosted apps differ from SharePoint-hosted apps.
- Describe how apps of different types authenticate with a SharePoint server.
- Design a provider-hosted app architecture for a set of functional requirements.
- Design an auto-hosted app architecture for a set of functional requirements.
- Choose the most appropriate app architecture for a scenario.

Introducing Remote-Hosted Apps

SharePoint-hosted, auto-hosted, and provider-hosted apps take different approaches to two fundamental problems when they access SharePoint. These problems are:

- *Authentication.* How to ensure that SharePoint can identify the user and the app that a request comes from. By identifying the user and app, SharePoint can determine whether to allow or deny the request and ensure data security.
- *Isolation.* How to ensure that different companies cannot access each other's sensitive data when they use the same app. Any app that deals with personal, business-critical, legal, copyrighted, or other sensitive information must isolate the tenant companies that purchase the app.

By understanding the different app models and how they solve these problems, you can help ensure that you select the right model for each scenario.

SharePoint-hosted apps

As you have already seen, SharePoint-hosted apps are based on an app web, which is a SharePoint website that stores all webpages, lists, libraries, and other resources for the app. The app web handles the two problems automatically without any configuration or code by the developer:

- *Authentication.* In order to access the app web, a user must have already satisfied whatever authentication requirements are in place in SharePoint. For example, if you have disabled anonymous access to SharePoint, the user must have provided correct credentials before she saw the app start page. You can determine the current user from the JavaScript CSOM client context object or by using the REST API.

- **SharePoint app:**
 - Pages, lists, and other resources in app web
 - Authentication provided by SharePoint
 - Isolation provided by farm or tenancy
- **Auto-hosted app:**
 - Can only be installed in Office 365
 - Authentication provided by Windows Azure ACS
 - Isolation provided by auto-provisioning SQL Database
- **Provider-hosted app:**
 - App provider must maintain the remote web
 - Authentication provided by an S2S trust
 - Isolation must be built by the provider

- *Isolation.* In an on-premises farm, there is only one tenant. In this case, the boundaries of the SharePoint farm automatically isolate one tenant of your app from another. In Office 365, there is a single SharePoint farm for many tenants, but each tenant company has a single tenancy and cannot access SharePoint sites in another other tenancy. This isolation applies to app webs in the same way it applies to another other SharePoint site.

Auto-hosted apps

Auto-hosted apps are remote-hosted apps in which the remote web is a website within Windows Azure. If the app requires a database, and most do, the database is created in Windows Azure SQL Database. This is the cloud version of Microsoft SQL Server. An auto-hosted app uses facilities in Windows Azure to authenticate requests and isolate tenants:

- *Authentication.* Auto-hosted apps use the Open Authentication (OAuth) protocol to exchange security tokens and ensure authentication is trustworthy. Auto-hosted apps use Windows Azure Access Control Service (ACS) to authenticate users.
- *Isolation.* Auto-hosted apps use a remote website in Windows Azure and a database in Windows Azure SQL Database. Windows Azure can provision website and database automatically when a customer installs your app. In this way, isolation is automatic because each tenant has an isolated database.

Because auto-hosted apps use Windows Azure facilities in these ways, you can only install them in Office 365 SharePoint tenancies. They are not available for installation in on-premises SharePoint farms.

Provider-hosted apps

In a provider-hosted app, the remote web can be a website hosted on any web server outside the SharePoint farm. The web server can be Windows Azure, an IIS web server on your own premises or at an Internet Service Provider (ISP), or an Apache server or any other HTTP server. Similarly, you can use a database in Microsoft SQL Server, Windows Azure SQL Database, Oracle, MySQL, or any other database server. Therefore, provider-hosted apps are more flexible than SharePoint-hosted or auto-hosted apps, but you must solve the authentication and isolation issues yourself:

- *Authentication.* Provider-hosted apps use Server to Server (S2S) authentication to enable SharePoint to identify users and apps authenticated in the remote web. You must ensure that SharePoint trusts the remote web server to authenticate users. You create this trust relationship by creating an X.509 certificate that can be used to protect the exchange of security tokens.
- *Isolation.* When you design and build a provider-hosted app, you must take responsibility for ensuring that tenants are isolated. For example, you can design a database in which each record is stored with the tenancy ID. By providing the tenancy ID with every query, you can filter data so that tenants see only their own data.

Authentication Mechanisms

When you create a farm solution or sandboxed solution, SharePoint can authenticate the user account associated with each request and use that account to allow or deny access to resources. The SharePoint app model improves this design because SharePoint can authenticate both the user and the app that initiated a request. Therefore, you can assign permissions both to user accounts and apps. There are three mechanisms by which this authentication is performed.

- Internal authentication:
 - SharePoint pages and sites
 - SharePoint-hosted apps
 - Remote-hosted apps that use the cross-domain library
- External authentication:
 - With OAuth and Windows Azure ACS
 - Auto-hosted Apps
 - With an S2S Trust
 - Provider-hosted apps

Internal authentication

In internal authentication, SharePoint collects credentials from users, verifies them, and creates a signed token, which is returned to the browser. For subsequent requests, the browser includes this token so that SharePoint can verify that the request is authenticated. The token is in the Security Assertion Markup Language (SAML) format. The authentication may be completed by using Windows Integrated authentication or ASP.NET Forms authentication.

Internal authentication is used when you request and receive SharePoint pages from a site collection or site outside any app. It is also used when you request resources inside a SharePoint-hosted app web. Because the app exists in a dedicated application domain, SharePoint can identify both the user account and the app associated with the request.

It is also possible to use internal authentication from a remote-hosted app by using the cross-domain library. This is a SharePoint 2013 JavaScript library that you would use in a remote-hosted app to access SharePoint information from the browser, without reloading the entire page from the remote web.

In summary, internal authentication is used by SharePoint sites, SharePoint-hosted apps, and remote-hosted apps that use the cross-domain JavaScript library.

External authentication using OAuth

In external authentication, an external web server (usually the web server that hosts the remote web) collects and verifies credentials from the user. To use external authentication, you must make sure that SharePoint trusts the external web server to identify users.

One way to ensure this trust is to use OAuth and the Windows Azure ACS. In this case, you must make sure that ACS can identify user accounts and has a security principal for the application. When a request is sent to SharePoint from the remote web server, ACS embeds the app ID and the user ID in an access token, which is included with the request. In this way, SharePoint can allow or deny access for user account or app identities.

External authentication through OAuth is available from auto-hosted apps because these are deployed in Office 365 and Windows Azure. The necessary trust relationship between SharePoint and ACS is present by default. You only have to create an ACS security principal for the app.

External authentication using S2S

When you do not have Windows Azure and Office 365, you must configure a trust relationship between the remote web server and SharePoint yourself. This situation usually applies to provider-hosted apps, which often run on web servers outside both the SharePoint farm and Windows Azure.

In such situations, you can establish the necessary trust by configuring Server-to-Server (S2S) authentication. You create an X.509 certificate with a private and public key owned by the remote web server. When the remote web server formulates a request to SharePoint, it includes an access token signed with the private key. If SharePoint can verify the request with the public key, it proves that the trusted remote web server authenticated the user. Both the user ID and the app ID are usually included in the access token, although in some scenarios, you may include just the app ID.

Provider-Hosted Apps

A provider-hosted app is the most versatile app architecture because you can host the remote web in Windows Azure or elsewhere, use client-side and server-side code, access SharePoint data and external data, and choose from a wide range of technologies from which to build your app. However, to enable this functionality, you have more responsibility—you must ensure the app is always available, set up the remote web, configure a trust relationship, and isolate tenants.

Building provider-hosted apps

In a provider-hosted app, the remote web is a website, and you can use any website technology to build and host it. Often, provider-hosted apps are built with Microsoft technologies such as ASP.NET and hosted on IIS or Windows Azure. However, you can also choose to use PHP or Ruby on Rails to build server-side code and host the remote web on Apache or another web server.

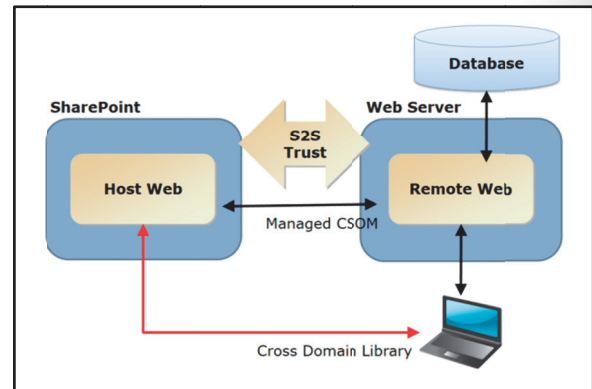
Most provider-hosted apps use a database to store information as well as accessing SharePoint content. This may be a Microsoft database like SQL Server or Windows Azure SQL Database. However, if your team has skills in Oracle, MySQL, or another database, you can use those instead.

Ensuring isolation

In a provider-hosted app, there is no automatic mechanism to ensure isolation. This does not matter if you are building an app for a single company that will never be used by anyone else. In such a case, both the remote web and the app database are used by a single company. However, more commonly, you want to sell the app to multiple customers, perhaps by publishing the app in the Office Store. If there is any possibility that more than one company will use your app, you should design it to ensure isolation.

In a provider-hosted app, all customer companies access the same remote web, which is hosted on an Internet-facing web server. If the remote web works with a single database, you must ensure that every database record that has sensitive data has a column that identifies the tenant. Any code that queries for sensitive data must filter results based on the current tenant ID. Such filtering prevents each tenant from seeing data outside their own organization.

Alternatively, you can ensure isolation by using separate databases for each tenant. In this case, you must consider how a new database will be created when a new tenant purchases and installs your app. Ideally, you should automate this process.



Setting up the remote web

In a provider-hosted app, you or your website administrators are responsible for configuring and maintaining the remote web. You should consider the following factors when you design the infrastructure that will support this web:

- *Demand.* How many users do you expect to log on simultaneously at peak times? How much load does an average user generate? Does your web server or server farm support such a load?
- *Scalability.* Can your architecture respond to an expected or unexpected increase in demand?
- *Cost.* What is the Total Cost of Ownership (TCO) of the web server or web farm that supports your requirements?
- *Availability.* Is your web server or farm resilient to hardware failures, power cuts, and other disasters? Will you commit to a minimum percentage up-time in your service level agreement?

All the considerations are common to SharePoint app remote webs and less specialized websites, so website administrators are likely to be experienced in making these decisions. If you use a cloud-based web server such as Windows Azure, much of this responsibility is taken by the cloud solution provider instead of by your web team.

Installation

Remember that customers may find your app by installing it from the Office Store. Ideally, this process should be automated and not require customers to call your administrators or wait for a response from your staff. Although all tenants may use the same remote web when they access your app, they usually need new database records, dedicated database tables, or even a separate database to support their data.

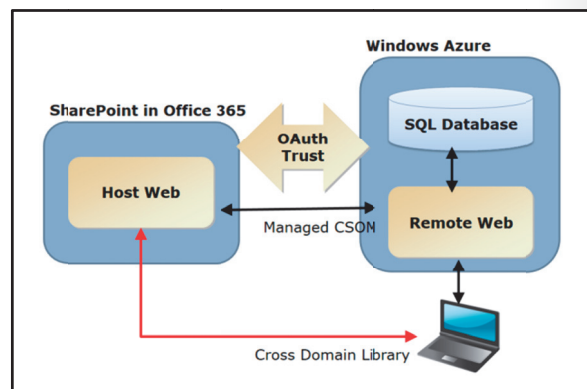
Security

Provider-hosted apps most often use external authentication through an S2S trust relationship to authenticate with SharePoint. To configure this trust relationship, you must install an X.509 certificate from a trusted certificate authority in your remote web. This certificate enables SharePoint to verify the access token the remote web provides and identify the app and the user that initiated the request.

Alternatively, you may choose to send all SharePoint calls from JavaScript code that runs in the browser. This can be useful, for example, when firewalls prevent direct calls from the remote web server to SharePoint. It also helps to make the remote web responsive because each SharePoint call does not require the webpage to reload. To enable such client-side SharePoint calls, you can use the cross-domain library. If you use this library, SharePoint uses internal authentication to verify the user identity and the app identity.

Auto-hosted Apps

An auto-hosted app is a SharePoint app that uses Windows Azure to host the remote web and Windows Azure SQL Database to host the app database. Auto-hosted apps are easier to develop and publish because tenancy isolation and installation are automatic. Customers can find your app in the Office Store, install it, and start using it without any involvement from you or your team of administrators. You need not build this automatic provisioning yourself—instead, Office 365 and Windows Azure work together to set up each tenancy.



Building auto-hosted apps

Auto-hosted apps for SharePoint only work with Office 365 SharePoint tenancies: customers cannot install them in an on-premises SharePoint farm. You must ensure all stakeholders are aware of this limitation if you decide to develop an auto-hosted app because it limits the possible market for your app. Although Office 365 is gaining popularity, there are many companies that will never migrate to a cloud version of SharePoint and maintain their on-premises farm.

You are also limited in the technologies you choose to build and host the app. You can use the following technologies:

- *ASP.NET*. Server-side code in the remote web must be managed .NET Framework code in an ASP.NET web application. You can choose to build the remote web by using ASP.NET Web Forms site or ASP.NET MVC.
- *Windows Azure*. You must host the remote web in a Windows Azure website. When a new customer installs your app, Office 365 and Windows Azure create a new, dedicated instance of the remote website.
- *SQL Database*. You use SQL Database to store any data outside SharePoint. When a new customer installs your app, Office 365 and Windows Azure create a new, dedicated instance of the database exclusively for that customer.

Installation, isolation, and scaling

When a customer purchases and installs an auto-hosted app, Office 365 and Windows Azure work together to create a new instance of the remote web and the app database. These objects are created in the customer's Windows Azure account. This automatic provisioning has the following advantages for the developer:

- *Installation is automatic*. You do not need to take any actions to support a new customer. All required actions are completed by Office 365 and Windows Azure.
- *The remote web is the customer's responsibility*. You do not need to guarantee availability or back up the remote web and its database. Instead, the customer owns these objects, and Windows Azure ensures availability.
- *Isolation is automatic*. You do not need to write extra code to ensure each tenant can only see their own data. Instead, each tenant has an isolated database so there is no possibility of a security breach.
- *Scaling is automatic*. The app provider does not need to add extra capacity to the remote web servers as the customer base grows. Instead, new remote webs are created for each customer and their users. Customers can scale their own websites and databases in Windows Azure to support large numbers of users.

As you can see, the auto-hosted architecture removes many responsibilities from the app developers and providers company. This can drastically reduce the development costs for auto-hosted apps compared to provider-hosted apps.

Security

Auto-hosted apps usually use external authentication with the OAuth protocol. This mechanism is convenient because the necessary trust relationship between SharePoint in Office 365 and ACS is in place by default. Again, the responsibility for the trust between remote web and SharePoint is taken by Windows Azure and not by the app provider.

As for provider-hosted apps, you can choose to perform internal authentication in an auto-hosted app by using the cross-domain library. This library can only be used from client-side JavaScript code.

Discussion - Choosing an App-Hosting Model

The following scenarios describe some requirements for apps for SharePoint. In each case, discuss with the instructor and your fellow students which app-hosting model you would choose to implement the required functionality.

A shared knowledge base

Your organization produces a range of precision-engineered machine components. You want to help your customers solve technical issues by publishing a knowledge base of technical articles, "how to" articles, and troubleshooting tips.

Because SharePoint is widely used in your industry, you want to publish the knowledge base as a SharePoint app. You want all customers to be able to contribute their own technical knowledge and to view articles published by other customers and your own technical staff.

A photo library

Your organization produces apps for SharePoint that encourage creativity among SharePoint users in each tenancy. You want to build an app that customers can install to help users share photos. Each photo will be stored with technical information such as the camera used, aperture, and exposure time. Users will be able to add comments to each other's photos, and the most popular photos will be displayed on the app start page.

A Customer Relationship Management app

Your organization has a successful stand-alone Customer Relationship Management (CRM) application that does not require SharePoint. However, because many of your customers use SharePoint, you would like to build a new version as a SharePoint app. You will use the same database schema as your stand-alone application. Your customers use both Office 365 and SharePoint on-premises farms.

Which app-hosting model would you use in the following scenarios?

- A shared knowledge base
- A photo library
- A Customer Relationship Management app

Lesson 2

Configuring Remote-Hosted Apps

Now that you understand how remote-hosted apps communicate with SharePoint and how provider-hosted apps differ from auto-hosted apps, you can learn how to configure security, authentication, permissions, and other settings for your application based on the hosting model you use. In this lesson, you will see how to configure settings in the app manifest file, as well as other infrastructure requirements, such as certificates.

Lesson Objectives

After completing this lesson, you will be able to:

- Configure authentication for auto-hosted apps in the app manifest and Windows Azure ACS.
- Configure an STS trust relationship for a provider-hosted app.
- Register an app principal and configure permission requests for remote-hosted apps.

Configuring Auto-Hosted App Authentication

SharePoint-hosted apps use internal authentication to identify both the app and the current user to SharePoint so that permissions can be checked. Remote-hosted apps use external authentication, unless you use the cross-domain library to call SharePoint from client-side JavaScript code. For auto-hosted apps, the OAuth protocol and ACS authenticates apps. For provider-hosted apps, you usually configure an S2S trust between the remote web server and SharePoint.

SharePoint-hosted apps require little configuration for authentication to work, but when you write remote-hosted apps, you must configure the app manifest and Web.config files correctly. Your precise configuration depends on the authentication mechanism the app uses.

- Understanding app principals and app identifiers
- App manifest requirements:

```
<AppPrincipal>
  <AutoDeployedWebApplication />
</AppPrincipal>
```

- Web.config requirements

```
<configuration>
  <appSettings>
    <add key="ClientId" value="Your-GUID-Here" />
    <add key="ClientSecret" value="Your-SecretHere" />
  </appSettings>
</configuration>
```

Understanding app principals and app identifiers

In SharePoint 2013, you can manage both user permissions and app permissions for accessing SharePoint resources. This is a major advantage that the app model has over farm solutions and sandboxed solutions. To enforce these permissions, SharePoint must be able to identify both users and apps. Therefore, just like users have user accounts, apps have app principals. App principals include the following information:

- *Client ID*. This is a GUID that uniquely identifies the app. The client ID is often called the app ID.
- *Client Secret*. This is a symmetric encryption key. It is used to encrypt tokens as they are sent from browser to the remote web and SharePoint.
- *Title*. This is the name of the app principal.
- *App Host Domain*. This is the DNS domain in which the remote web is located.

When you install a SharePoint app, you must register an app principal within the SharePoint tenancy. This creates an app identifier, which is a unique string that includes the app client ID and the tenancy ID. When SharePoint assigns a permission to your app, it stores the permission with the app identifier. This means that each app permission in SharePoint is specific to one installation of one app.

In a SharePoint-hosted app or an auto-hosted app, SharePoint automatically creates an app principal and an app identifier every time a user installs your app. Although you must understand app principals for these types of apps, you do not need to create them yourself.

App manifest requirements

In an auto-hosted app, you must inform SharePoint that app principals should be created automatically by using the **AutoDeployedWebApplication** element in the app.manifest file.

The following XML shows how to configure the app.manifest file.

An App Manifest for an Auto-hosted App

```
<AppPrincipal>
  <AutoDeployedWebApplication />
</AppPrincipal>
```

Web.config requirements

Auto-hosted apps also require two settings to be present in the Web.config file in the root folder of the remote web. These are the client ID and the client secret. The values in the Web.config file must match the corresponding values in the app principal.

The following XML shows how to configure these two values in Web.config:

Web.config Settings for an Auto-hosted App

```
<configuration>
  <appSettings>
    <add key="ClientId" value="12345678-1234-1234-1234-123456789012" />
    <add key="ClientSecret" value="fjewk4fcnekrngs129fcnder7xwer21FEGcegD47D2VFf=" />
  </appSettings>
</configuration>
```

Configuring Provider-Hosted App Authentication

Provider-hosted apps are more difficult to configure than auto-hosted apps because you must register app principals manually and ensure that the necessary trust relationship is in place between the remote web and the SharePoint server.

Registering app principals

To register an app principal for a provider-hosted app, use the AppRegNew.aspx page in the host web. This new SharePoint 2013 page requests the following information:

- *App ID*. This is the client ID that your app will use to identify itself.
- *App Secret*. This is the client secret that your app will use to encrypt tokens.
- *Title*. This is a user-friendly title for the app.

- Registering app principals
- App manifest requirements
- Configuring an S2S trust
- Web.config requirements

- *App Domain*. This is the DNS domain where the remote web is hosted.
- *Redirect URL*. This optional value enables apps to request permissions.

You must have administrative permissions in your SharePoint tenancy to register an app principal in this way.

Alternatively, you can use Windows PowerShell to register an app principal, as the following code illustrates. This code uses the **Register-AppPrincipal** cmdlet.

Registering an App Principal by Using PowerShell

```
$appDisplayName = "My First App"
$clientID = "66142967-91BB-4C50-9B70-06375D5240BC"
$targetWeb = Get-SPSite "http://intranet.contoso.com"
$authRealm = Get-SPAuthenticationRealm -ServiceContext $targetWeb
$appIdentifier = $clientID + "@" + $authRealm

Write-Host "Creating the new app principal registration..."
Register-SPAppPrincipal -NameIdentifier $appIdentifier -Site $targetWeb.RootWeb
-DisplayName $appDisplayName
```

App manifest requirements

In a provider-hosted app, you do not use the **AutoDeployedWebApplication** element in the app manifest file. Instead, you must use the **RemoteWebApplication** element and include the client ID.

The following XML shows how to configure the app.manifest file for a provider-hosted app.


An App Manifest for a Provider-hosted App

```
<AppPrincipal>
  <RemoteWebApplication ClientId="12345678-1234-1234-1234-123456789012" />
</AppPrincipal>
```

Configuring an S2S trust

Provider-hosted apps that don't use Windows Azure ACS and OAuth must have a trust relationship between SharePoint and the remote web server for authentication to work. To create such a trust, you must complete the following configuration tasks:

- *Obtain an X.509 Certificate*. An X.509 certificate contains a public and private key pair and is digitally signed by a Certificate Authority (CA). For your production environment, you should purchase an X.509 certificate from an established CS. Such a certificate establishes your app's identity to SharePoint. For development purposes, you can create your own X.509 certificate by using the command line `makecert.exe` and `certmgr.exe` tools.
- *Install the X.509 Certificate on SharePoint*. You must register the certificate as a trusted security token issuer. You can create this registration by using the `New-SPTrustedSecurityTokenIssuer` Windows PowerShell cmdlet.
- *Make private key available on the remote web server*. To complete this step, you must export the private key from the certificate and install it on the remote server.

 **Note:** You will see how to perform each of these steps in detail in Lab A: Configuring a Provider-Hosted SharePoint App.

Web.config requirements

The Web.config file for your provider-hosted app must include extra properties that enable the app to use the S2S trust.

The following XML shows how to configure a provider-hosted app to use an S2S trust.

Configuring a Provider-hosted App

```
<configuration>
  <appSettings>
    <add key="ClientId" value="12345678-1234-1234-1234-123456789012" />
    <add key="ClientSigningCertificatePath"
value="C:\Certificates\appwebserver.contoso.com.pfx" />
    <add key="ClientSigningCertificatePassword" value="Pa$$w0rd" />
    <add key="IssuerId" value="12345678-1234-1234-1234-123456789012" />
  </appSettings>
</configuration>
```

Requesting App Permissions

As you have learned, SharePoint 2013 can authenticate both the user and the app when a request for SharePoint content is made. Using these identities, SharePoint can check the permissions on a requested resource and determine whether the user has the appropriate permissions and whether the app has the appropriate permissions. The request will be permitted only if both the user and the app have at least the required level of access.

Any app that creates an app web, such as a SharePoint-hosted app, has full access to all the resources in that app web. However, to access resources in the host web or elsewhere in SharePoint, an app must request those permissions during installation. When you create an app, you must configure the permissions it will request when a user installs it.

```
<AppPermissionRequests>
  <AppPermissionRequest Right="Read"
    Scope="http://sharepoint/content/sitecollection/web" />
  <AppPermissionRequest Right="Write"
    Scope="http://sharepoint/content/sitecollection/web/lists" />
  <AppPermissionRequest Right="Read"
    Scope="http://sharepoint/content/tenant" />
  <AppPermissionRequest Right="QueryAsUserIgnoreAppPrincipal"
    Scope="http://sharepoint/search" />
  <AppPermissionRequest Right="Write"
    Scope="http://sharepoint/social/microfeed" />
</AppPermissionRequests>
```

App permission requests

SharePoint assigns permissions to an app during app installation. You configure the permissions that your app will request in the app manifest file during development. During installation, the user who installs your app is shown a dialog box that explains the permissions that the app requires to run. If the user decides to click the **Trust It** button, the requested permissions are assigned to the app. Otherwise, the installation halts.

To request app permissions, you must include **AppPermissionRequest** elements in the app manifest file for your app. These elements must be included in a top level element named **AppPermissionRequests**. Each **AppPermissionRequest** element includes two attributes:

- **Scope.** The Scope attribute determines the object that the app requires access to. For example:
 - *Scope="http://sharepoint/content/sitecollection/web/list"*. This Scope attribute requests a permission on all lists within the host web.
- **Right.** The Right attribute determines the level of access requested on the object in the Scope attribute. For example:
 - *Right="Read"*. This Right attribute requests read-only access to all objects.

- *Right="Write"*. This Right attribute requests permission to modify items.
- *Right="FullControl"*. This Right attribute requests full access to all properties and data for object.



Note: An app permission assigns access to a type of object. For example, you can request read access to all the lists in the host web. Contrast this with user permissions, which can grant read access to a single specific list, such as the Suggestions list.

The value of the Scope attribute has the form of a URL but does not identify a resource by its URL. Instead, it has three parts, each of which can contain a limited number of values:

- *Product*. To access SharePoint resources, use the "http://sharepoint" value. You can also request access to resources in other products, such as Exchange Server or Lync Server.
- *Permission Provider*. The "content" value requests access to items in the SharePoint content database. SharePoint includes other permission providers, such as "search" and "social."
- *Target Object Type*. The "sitecollection" value requests access to properties of the SharePoint site collection. The "sitecollection/web" value requests access to the host web. The "sitecollection/web/list" value requests access to all the lists within the host web.



Additional Reading: For more information about app permission request scopes, see *Plan app permissions management in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=307096>.

The following XML code shows how to request permissions in the app manifest file. Several examples are included to help you understand the Scope attribute.

Requesting Permissions

```
<AppPermissionRequests>

  <!-- This element request read access to the properties of the host web -->
  <AppPermissionRequest
    Right="Read"
    Scope="http://sharepoint/content/sitecollection/web" />

  <!-- This element requests write access to all lists in the host web -->
  <AppPermissionRequest
    Right="Write"
    Scope="http://sharepoint/content/sitecollection/web/lists" />

  <!-- This element request read access to the properties of the tenancy -->
  <AppPermissionRequest
    Right="Read"
    Scope="http://sharepoint/content/tenant" />

  <!-- This element request search access without authenticating as the app -->
  <AppPermissionRequest
    Right="QueryAsUserIgnoreAppPrincipal"
    Scope="http://sharepoint/search" />

  <!-- This element request write access to a user's social networking microfeed -->
  <AppPermissionRequest
    Right="Write"
    Scope="http://sharepoint/social/microfeed" />

</AppPermissionRequests>
```

If the user who installs the app does not have at least the level of permission requested by the app, the user cannot complete the installation. For example, only the site administrator can install an app that requests full control over a SharePoint site. For this reason, you should only include permission requests in your app manifest file for operations that your app really needs to function correctly.

The permissions assigned to an app during installation are stored with the app identifier. Remember that the app identifier is a string that includes both the tenancy ID and the app client ID. This means that each permission is specific to one instance of your app. That permission does not apply to other installations of your app, whether those installations are in the same tenancy, in other tenancies in the same SharePoint system, or in other SharePoint farms.

Lab A: Configuring a Provider-Hosted SharePoint App

Scenario

The finance team at Contoso stores sales ledgers and purchase ledgers for different regions in separate lists on their site. Invoices are issued and paid in local currency, so each ledger entry is stored with a region. Entries in the Regions list store the currency and exchange rate for each region. The chief financial officer wants to view and compare sales ledger and purchase ledger balances for each region in one place. Your task is to implement this functionality in a provider-hosted app. In this lab, you will configure trust relationships and configure the app settings. In the next lab, you will develop the functionality of the app.

Objectives

After completing this app, you will be able to:

- Configure SharePoint to trust a provider-hosted app.
- Configure settings for a provider-hosted app.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-09
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Configuring An S2S Trust Relationship

Scenario

For your provider-hosted app to communicate with SharePoint and access resources, you must:

- Create an X.509 Certificate for the trust relationship and export the private key in an encrypted file.
- Register the certificate with IIS.
- Create a trusted token issuer in the SharePoint farm.
- Create an app principal for the app.

In this exercise, you will complete these tasks by adding key lines of code to Windows PowerShell scripts. You will then execute the scripts in order to configure the trust relationship

The main tasks for this exercise are as follows:

1. Create and Register an X.509 Certificate
2. Register a Trusted Security Token Issuer
3. Register an App Principal

► Task 1: Create and Register an X.509 Certificate

- Open the following script in Windows PowerShell ISE:
- E:\Labfiles\LabA\PowerShell\Starter\CertificateCreator.ps1
- Locate the "Create a new folder here" comment and add a command below it that creates a new folder. Use the following Information:
 - Cmdlet: New-Item
 - Path: \$certFolder
 - Item Type: Directory

- Use the **Force** parameter
- Confirm: False
- Pipe the output to the **Out-Null** cmdlet
- Locate the "Create the new certificate here" comment and in the line below it, modify the command that creates a new certificate to use the **localMachine** certificate store location.



Note: For help with the makecert.exe options, open a command prompt and type these two commands:

```
c:\ProgramFiles\Microsoft Office Servers\15.0\Tools\makecert -?
c:\ProgramFiles\Microsoft Office Servers\15.0\Tools\makecert -!
```

- Locate the "Register the certificate here" comment and in the line below it, modify the command that creates a new certificate to use the **localMachine** system store location.



Note: For help with the certmgr.exe options, open a command prompt and type this command:

```
c:\ProgramFiles\Microsoft Office Servers\15.0\Tools\certmgr -?
```

- Locate the "Export the private key to a variable here" comment and in the line below it, modify the command that exports the private key to use the string "**Password**" as the password.
- Save your changes
- Run the CertificateCreator.ps1 Windows PowerShell script.
- Examine the contents of the C:\Certificates folder to check that the results of the script.

► Task 2: Register a Trusted Security Token Issuer

- In Windows PowerShell ISE, open the following script:
 - E:\Labfiles\LabA\PowerShell\Starter\TrustedTokenIssuerCreator.ps1
- Locate the "Create the new trusted token issuer here" comment and in the line below it, modify the command that creates a trusted token issuer in SharePoint to use the **\$publicCertificate** certificate.
- Save your changes.
- Run the TrustedTokenIssuerCreator.ps1 PowerShell Script. Examine the details of the token issuer that PowerShell returns.

► Task 3: Register an App Principal

- Open the following script:
 - E:\Labfiles\LabA\PowerShell\Starter\AppPrincipalCreator.ps1
- Locate the "Create the new registration here" comment and in the line below it, modify the command that creates a new app principal to use the **\$targetWeb.RootWebsite**.
- Save your changes
- Run the AppPrincipalCreator.ps1 script.
- Close all open windows.

Results: A working trust relationship that an app can use to authenticate with SharePoint and access resources.

Exercise 2: Creating a Provider-hosted App

Scenario

The Windows PowerShell scripts you completed and executed have created all the required elements for the S2S trust to function. In this exercise, you will create and configure a provider-hosted app to use the trust relationship.

The main tasks for this exercise are as follows:

1. Configure the App Manifest File
2. Configure Web.config
3. Test the App

► Task 1: Configure the App Manifest File

- Open Visual Studio 2012 and open the `E:\Labfiles\LabA\App\Starter\ContosoLedgersApp\ContosoLedgersApp.sln` solution.
- In the **AppManifest.xml** file for the Contoso Ledgers app, configure the **ClientId** for the remote web to be `66142967-91BB-4C50-9B70-06375D5240BC`
- Insert an **AppPermissionRequests** element within the **App** top-level element.
- Add an **AppPermissionRequest** element that requests read access to all lists and libraries in the host web.
- Save your changes.

► Task 2: Configure Web.config

- In the Web.config file for the ContosoLedgersAppWeb project, set the following value for the ClientId app setting:
 - `66142967-91BB-4C50-9B70-06375D5240BC`



Note: Notice that the certificate path, password and issuer ID have been configured by the SharePoint app template wizard when you created the project.

- Save your changes.

► Task 3: Test the App

- Start the app in debugging mode, trust the app, and then make a note of the numbers of sales and purchases that the app displays.
- Browse to the **<http://dev.contoso.com>** site and note the number of items in the **Sales Ledger** list.
- Note the number of items in the **Purchase Ledger** list. Did the Contoso Ledgers app report the correct totals of the Sales Ledger and Purchase Ledger lists?
- Stop debugging and close Visual Studio.

Results: A simple provider-hosted app that uses an S2S trust to access SharePoint and displays item totals from lists in the host web.

Question: The certificate you created and used in this lab to configure the S2S trust relationship is not suitable for a completed provider-hosted app that is published to the Office Store. Why is this?

Question: In the default code for the Contoso Ledgers app, what method on the **TokenHelper** class is used to obtain the client context object?

Lesson 3

Developing Remote-Hosted Apps

After choosing an architecture and then configuring security, permissions, and a host web server for your new remote-hosted SharePoint app, you must write code to implement your functional requirements. Much of this code will use the JavaScript CSOM, the managed CSOM, or the REST API to access SharePoint data as you saw in the last module. In this lesson, you will learn how to choose a suitable technology to build your remote web, how to supply access tokens with SharePoint requests, and how to use the SharePoint Chrome Control to inherit a SharePoint look and feel in your app web.

Lesson Objectives

After completing this lesson, you will be able to:

- Choose a server-side technology, and SharePoint API, and a cross-domain call API with which to build a remote web.
- Write managed code to create and store access tokens to validate SharePoint requests.
- Use the Chrome Control to ensure that your remote-hosted app blends with the look and feel of the host web.

Choosing Technologies

When you build a remote-hosted SharePoint app, you have a wide range of technology choices. The ones you use depend on your company's development standards, your preferred development methodology, and the skills possessed by your team of developers.

All of the technologies discussed in this topic can be used in both provider-hosted and auto-hosted apps.

Server-side technologies

Because the remote web is simply a specialized form of website, you can build it by using any web technology. For example, you can create it using static HTML pages, PHP, Ruby on Rails, and so on. Because Windows Azure can host PHP and Node.js sites, you can even choose non-Microsoft technologies when you build auto-hosted apps, which must leverage ACS and can be provisioned on Office 365.

However, because SharePoint is a Microsoft technology, many providers will choose the SharePoint server-side code execution technology to build the remote web. This technology is called ASP.NET. When you choose ASP.NET, you can write server-side code in Visual Studio, Visual C#, or any other .NET Framework language. You can also choose from the following three programming models:

- *Web Pages*. This is a simple programming model in which pages include both HTML elements and server-side code within code blocks. A view engine executes the server-side code at run time and inserts the results into the HTML page. In this way, data from SharePoint, a database, or another source can be formatted and inserted into a webpage, which is sent to the browser. A Web Pages site often uses the Razor view engine but other view engines are also available. Choose Web Pages if your remote web server-side code is simple and your team does not have Web Forms or MVC skills.

- Server-side technologies:
 - Non-Microsoft technologies
 - ASP.NET
 - Web Pages
 - Web Forms
 - MVC
- Server-side calls to SharePoint:
 - Managed CSOM
 - REST API
- Client-side calls to SharePoint:
 - JavaScript CSOM
 - REST API

- *Web Forms*. This more sophisticated programming model enables developers to drag and drop controls such as buttons, text boxes, links, and data grids onto a webpage to build up a user interface. Each control has a range of event handlers in which developers write code to respond to user actions. Many web development teams have good Web Forms skills. Choose Web Forms if you already know the technology or if you want to build a remote web in a way similar to building a desktop application. SharePoint itself is built using Web Forms and the SharePoint app project templates in Visual Studio use Web Forms.
- *MVC*. In this programming model, you build a web application by define three types of object: Models, Views, and Controllers. Models are classes that model the objects that your app handles. For example, in a SharePoint app, you might define a model that corresponds to a particular SharePoint content type. Views define the user interface for the app. Views are HTML webpages that include managed code blocks that execute on the server, much like webpages. Controllers are classes that respond when the user takes an action such as clicking a link or requesting a particular URL. Controllers are central to MVC apps because they retrieve a Model object from a database or SharePoint and pass it to a View for display. MVC ensures good separation of concerns by dividing server-side code into Models, Views, and Controllers, which helps when you build large and complex apps. MVC is also compatible with unit testing and test-driven development. Choose MVC if you want to create a highly complex app, if you want to test your app thoroughly throughout development, or if your team already has MVC skills. MVC also gives the developer very precise control over the HTML that is rendered.

Server-side calls to SharePoint

When you build a remote-hosted app that accesses SharePoint content, you must first choose whether to make those calls from server-side code, which runs on the remote web server, or client-side code, which runs on the browser.

Server-side code has the following advantages:

- If the remote web server is physically close to the SharePoint web front-end servers, server-side code can access SharePoint content quickly.
- Server-side code can integrate data from SharePoint and another source, such as a database before pages are sent to the browser.
- Server-side code is compiled before execution. For this reason, it can perform complex operations more rapidly than JavaScript code on the client. Bear in mind, however, that when there are many simultaneous users all requesting complex operations, the server's computational resources may be stretched and delays can occur.
- Compiled code cannot be viewed or copied easily. This helps to protect your technology by preventing competitors from seeing your source code.

You can choose to call SharePoint from server-side code by using two APIs:

- The Managed CSOM
- The REST API

See Module 8 for details of how to use these APIs.

Client-side calls to SharePoint

If you choose to call SharePoint from client-side code, the remote web server sends an HTML page to the browser that includes JavaScript code inline or in a separate script file. The JavaScript code can use the SharePoint JavaScript CSOM script library to access SharePoint content and display it.

Use client-side calls if you want to move processing load off the remote web server. Often, server-side code runs faster than client-side script because server-side code is compiled, and server processing and memory resources are faster. However, when there are many simultaneous users making complex requests, they must contend for fast but limited server resources. Client-side execution does not cause this problem because each browser process requests locally and does not place load on the web server after the page is delivered.

You can choose to call SharePoint from client-side code by using two APIs:

- The JavaScript CSOM
- The REST API

See Module 8 for details of how to use these APIs.

Another scenario in which you should choose to use client-side code to access SharePoint is when the remote web server is separated from SharePoint by a firewall. Because REST and the Managed CSOM send SharePoint requests through port 80 and the HTTP protocol, most firewalls pass these requests. However, occasionally, a very strict firewall policy prevents SharePoint and the remote web server communicating directly. Making calls to SharePoint from the client can circumvent these restrictions.

Many real-world apps use a combination of client-side and server-side calls to SharePoint. For example, you can use a Managed CSOM call on the server to build a page to display in the browser. Later, you may want to update some of the data displayed without rebuilding the entire page. You can use a client-side JavaScript call to refresh a small portion of the page data from SharePoint. This approach helps to improve the responsiveness of the page.

Coding Security Requirements

You have seen how SharePoint uses different authentication mechanisms to identify the app and the user associated with a request. After SharePoint performs this authentication, it issues an access token to the user's browser. You must make sure that this access token is included with all subsequent requests to assure SharePoint that it has already performed authentication. Because the token is signed, it cannot be incepted by any malicious user and used to bypass authentication. In this topic, you will learn about the different types of token and how to handle them in your code.

- Token types:
 - Context tokens
 - Access tokens
 - Refresh tokens
- Using the TokenHelper class:
 - Accessing the context token
 - Obtaining an access token and including it in subsequent requests

Token types

To pass a token correctly, you must understand the different kind of tokens that SharePoint uses. These include the following.

- *Context tokens.* Context tokens are only used in authentication through OAuth, not through S2S trusts. Context tokens are issued by ACS and include user identity, the tenancy identity, and the URL of the host web.
- *Access tokens.* An access token is issued when a client is authenticated. This is the token that you must include with requests to prove that your client is authenticated. You pass this token in the HTTP header. Visual Studio SharePoint app project templates include a special class, named **TokenHelper**, to assist you in coding access tokens.

- *Refresh tokens.* An access token expires 12 hours after it is issued. To obtain a new access token, the app must re-authenticate, and this can take time. You can make this process more efficient by using refresh tokens because refresh tokens last for six months. You can pass a refresh token to obtain a new access token without a complete authentication operation. You can store refresh tokens, for example in the database of a remote web.

The TokenHelper class

The **TokenHelper** class helps you obtain tokens and include them in the HTTP headers you send with CSOM and REST API SharePoint calls. You can use the following methods on the **TokenHelper** class:

- *TokenHelper.GetContextTokenFromRequest.* If the authentication mechanism is OAuth, the context token from ACS is included in the **Page.Request** object. You can use the **GetContextTokenFromRequest** method to isolate the context token and store it.
- *TokenHelper.GetClientContextWithContextToken.* Pass the context token to this method to obtain an access token. The method returns a client context object. Whenever you call **ExecuteQuery** on that client context, the access token is included by default.
- *TokenHelper.GetS2SAccessTokenWithWindowsIdentity.* If the authentication mechanism is S2S, there is no context token from ACS. Instead, you can use the **GetS2SAccessTokenWithWindowsidentity** method to obtain the access token from the user identity determine by the remote web server.

The following ASP.NET code uses the TokenHelper class to obtain a context token and an access token for an app authenticated by OAuth.

OAuth TokenHelper Examples

```
// Store the context token. This is possible because authentication used OAuth.
var contextToken = TokenHelper.GetContextTokenFromRequest(Page.Request);

// Store the host web URL, which is in the SPHostUrl request parameter.
var hostURL = Page.Request["SPHostUrl"];

// Get the client context object by passing the context token.
// Queries through this client context object automatically include the access token.
using (var context = TokenHelper.GetClientContextWithContextToken(hostURL, contextToken,
Request.Url.Authority)) {

    // Get the object you want by using Managed CSOM.
    context.Load(context.Web);
    context.ExecuteQuery;

}
```

The Chrome Control

When you build a SharePoint-hosted app, all pages and other resources are served by SharePoint from within the app web. For this reason, SharePoint-hosted apps automatically inherit the look and feel of the host web and present custom functionality within the familiar SharePoint environment.

This inheritance is not automatic when you create a remote-hosted app because the remote web is served from outside the SharePoint environment. There is no requirement to ensure that your remote web looks like the host web, but a similar look and feel reassures users; placing familiar links and controls where users expect them can ease the adoption of your app.

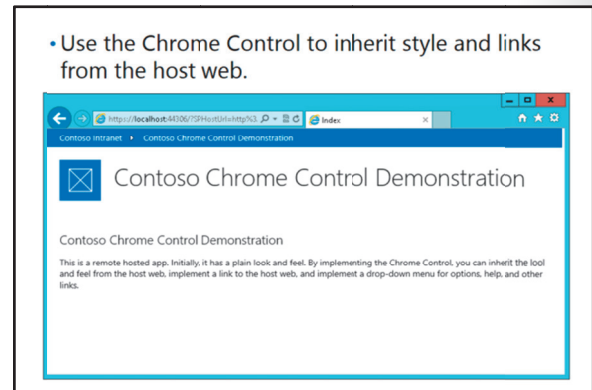
Remember that, although there is a default SharePoint look and feel, many companies customize how SharePoint looks. For example, they may implement custom style sheets, master pages, and branding. Therefore, you cannot hardcode the default SharePoint style into your app and expect that it will look like all SharePoint host webs.

The Chrome Control is a solution to this challenge. The Chrome Control obtains the style sheet and other header elements from the host web and applies them to a remote web at run time. By using the Chrome Control, you ensure that the look of your remote web automatically blends with other sites in and SharePoint system where your app is installed. In addition, the Chrome Control provides a link to the host web, which is a user interface requirement for all apps. Finally, the Chrome Control can present a drop-down list like the one in a SharePoint site that includes the site settings link. You can populate this list with any configuration or options links your app requires.

To use the Chrome Control, copy the `sp.ui.controls.js` JavaScript library into your remote web project. You must then ensure that each page in the remote web includes a **script** element that links to this library. In an ASP.NET remote web, you can use a master page or template view to propagate this link to every page.

You can use the Chrome Control by writing JavaScript code that calls the JavaScript library. Alternatively, you can use the Chrome Control declaratively by setting options on a **div** control.

- Use the Chrome Control to inherit style and links from the host web.



The following code shows how to use the Chrome Control declaratively

Setting Chrome Control Options in HTML

```
<div id="chrome-control-div"
  data-ms-control="SP.UI.Controls.Navigation"
  data-ms-options='{
    "appIconUrl": "../Images/AppIcon.png",
    "appTitle": "Contoso Remote App",
    "appHelpPageUrl": "../Help/HelpStart.aspx",
    "settingsLinks": [
      {
        "linkUrl": "../Options.aspx",
        "displayName": "Edit Options"
      },
      {
        "linkUrl": "../Admin.aspx",
        "displayName": "Configure App"
      }
    ]
  }'
>
</div>
```

Lab B: Developing a Provider-Hosted SharePoint App

Scenario

Now that you have created a server-to-server trust relationship and configured your app settings, you can start to add functionality to your app. The app must sum ledger balances for each region, convert all balances into U.S. dollars, and present a summary of the information. You want to ensure that the app inherits the look and feel of the host web and so you have decided to implement the Chrome Control.

Objectives

After completing this lab, you will be able to:

- Retrieve and manipulate SharePoint list data from remote-hosted apps.
- Use the Chrome Control to style a remote-hosted app.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-09
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Working with SharePoint Data

Scenario

You have created a very simple provider-hosted app with a functioning S2S trust relationship. The app displays two-item counts from SharePoint lists in the host web. In this exercise, you will extend this simple app to display sales totals and purchase totals in both local currencies and U.S. dollars. This summary will be helpful to company executives. The app will calculate totals according to the contents of three lists in the host web:

- *Regions*. This list stores currency regions in which Contoso operates. For each item, the list stores the name of the region, the name of the local currency, and the exchange rate against U.S. dollars.
- *Sales Ledger*. This list stores individual sales items for every region. Sales staff record details that include the customer and the amount for the sale in their local currency. They also choose a region from the Regions list.
- *Purchase Ledger*. This list stores individual purchases for every region. Buyers record the supplier name and the amount in their local currency. They also choose a region.

The app will use the amounts and the exchange rates from the Regions list to total sales and purchases in U.S. dollars.

The main tasks for this exercise are as follows:

1. Build the User Interface
2. Obtain Regions from SharePoint
3. Display Data in the User Interface

► Task 1: Build the User Interface

- Open the following Visual Studio solution for editing:
 - E:\Labfiles\LabB\Starter\ContosoLedgersApp\ContosoLedgersApp.sln
- Open the **Default.aspx** ASP.NET page for editing.
- Copy the entire contents of the following file to the clipboard:

- `E:\Labfiles\LabB\Markup\SalesAndPurchaseTables.txt
- In Visual Studio, in the **Default.aspx** page, paste the copied markup to replace the comment:
 - <!-- Display Sales and Purchase Summary Tables Here -->
- Start the app in debugging mode and trust the app.



Note: The app displays item totals and the header rows for Sales and Purchase summary tables.

- Stop debugging.

► Task 2: Obtain Regions from SharePoint

- In the code-behind for the **Default.aspx** page, after the "Get the three relevant lists" comment, get the list titled **Regions** and store it in a new object named **regionsList**. Then load the list.
- After the "Get the items collections" command, get all the items in the **regionsList** object and store them in a new object named **regionItems**. Then load the **regionItems** object.
- Save your changes.

► Task 3: Display Data in the User Interface

- In the **Default.aspx Page_Load** method, immediately before the end of the **using** block, create a loop that traverses all the **ListItem** objects in the **regionItems** collection.
- In the loop, pass the current **ListItem** object and the **saleItems** collection to the **createRow** method to create a new table row. Store the row in a **TableRow** object named **saleRow**.
- Add the **saleRow** object to the **Rows** collection of the **SalesTable** control.
- Pass the current **ListItem** object and the **purchaseItems** collection to the **createRow** method to create a new table row. Store the row in a **TableRow** object named **purchaseRow**.
- Add the **purchaseRow** object to the **Rows** collection of the **PurchasesTable** control.
- Start the app in debugging mode. The app displays summary sales and purchases data.
- Stop debugging.

Results: A provider-hosted app that displays a summary of global sales and purchases.

Exercise 2: Using the Chrome Control

Scenario

You have built an application that displays and summarizes data from three lists in the host web. The app uses an S2S trust relationship to authenticate with SharePoint. However the app currently has a plain look and feel that does not blend with the host web. In this exercise, you will fix this problem by implementing the Chrome Control in your app.

The main tasks for this exercise are as follows:

1. Set Up Chrome Control Scripts
2. Render the Chrome Control
3. Test the App

► Task 1: Set Up Chrome Control Scripts

- Add the following file to the **Scripts** folder:
 - C:\Program Files\Common Files\microsoft shared\web server extensions\15\TEMPLATE\LAYOUTS\sp.ui.controls.js
- In the **Default.aspx** page, insert a `<script>` element that links to the `sp.ui.control.js` file you just added to the project. Ensure the new `<script>` element is before the link to **App.js** and after the link to **jQuery**.
- At the start of the `<body>` element, insert a new `<div>` element with the ID **chrome-control-placeholder**.
- Save your changes.

► Task 2: Render the Chrome Control

- In the **App.js** script, replace the call to the **alert** function with code that creates a new object named **options**. Set the following properties for **options**:
 - `applconUrl: ../Images/Aplcon.png`
 - `appTitle: Contoso Ledgers App`
- Create a new **SP.UI.Controls.Navigation** object named **navControl**. Use the following information:
 - Chrome control div: **chrome-control-placeholder**
 - Chrome control options: **options**
- Set the **navControl** object to be visible.
- When the document has loaded, ensure that the **Contoso.ChromeControl.render()** method is called.
- Save your changes.

► Task 3: Test the App

- Start the app in debugging mode. If the app displays data without SharePoint formatting, use the Internet Explorer developer tools to clear the browser cache and then refresh the page.



Note: The remote app displays with SharePoint formatting applied.



Note: If time permits, you can test the app further by adding new regions, sales and purchases to the lists in the host web. Check that new regions are displayed as new rows in the tables and that the exchange rate you set for each region is used to calculate the totals in U.S. dollars.


- Stop debugging and close Visual Studio.


Results: A complete Contoso Ledgers app that summarizes global sales and purchasing data for Contoso executives.

Question: The Contoso Ledgers app you created accessed data in the host web and displayed it to the user. If you created the necessary lists in the app web, what changes could you make to your code?

Module Review and Takeaways

In this module, you have learned how to create and build remote-hosted apps that access SharePoint data. You saw how auto-hosted app differ from provider-hosted apps, how apps authenticate with SharePoint, and how you can ensure that each tenant who uses your app is isolated from all the other tenants to ensure security. You also saw how to create security access tokens and how to blend your app with SharePoint by using the Chrome Control.

 **Best Practice:** Use the Chrome Control whenever you want your app to blend with the host web's look and feel. You cannot be sure that the host web will always use the familiar SharePoint look and feel, so if you hard code the usual SharePoint design into your app, the app may not look like the customer's host web. The Chrome Control avoids this problem by downloading the style sheet and headers elements from the host web at run time.

 **Best Practice:** If you want to use an S2S trust relationship with a provider-hosted app, purchase an X.509 certificate from a globally-trusted certificate authority. If you use a certificate you created yourself, customers see error messages and warnings when they install your app that discourage them from proceeding with the installation.

Review Question(s)

Question: You have noticed that many of the Office 365 customers in your industry want extra customer relationship management functionality in their SharePoint online tenancy. You decide to build an app to satisfy these specialized requirements. What hosting model is easiest for you to build?

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or False: The Chrome Control can only be used with provider-hosted apps.	

Test Your Knowledge

Question	
<p>You are building a provider-hosted app, which you want to publish to the Office Store. You want customers with on-premises SharePoint farms to be able to use your app and you do not want to use a Windows Azure account. How should you configure authentication?</p>	
<p>Select the correct answer.</p>	
<input type="checkbox"/>	You do not need to configure authentication because each user's SharePoint account can be used to access all resources.
<input type="checkbox"/>	Use OAuth authentication with an X.509 certificate to create the trust relationship.
<input type="checkbox"/>	Use S2S authentication with an X.509 certificate to create the trust relationship.
<input type="checkbox"/>	Use an isolated SQL Database for authentication with an X.509 certificate to create the trust relationship
<input type="checkbox"/>	Use OAuth authentication with the ACS service, which has a default trust relationship with Office 365

Module 10

Publishing and Distributing Apps

Contents:

Module Overview	10-1
Lesson 1: Understanding the App Management Architecture	10-2
Lesson 2: Understanding App Packages	10-5
Lesson 3: Publishing Apps	10-9
Lab A: Publishing an App to a Corporate Catalog	10-13
Lesson 4: Installing, Updating, and Uninstalling Apps	10-16
Lab B: Installing, Updating, and Uninstalling Apps	10-20
Module Review and Takeaways	10-23

Module Overview

An important design objective for the SharePoint 2013 app infrastructure is to enable users to easily locate, purchase, and install apps. Another design objective is to make it easy for vendors to package and publish apps both within a SharePoint on-premises farm and in Microsoft Office 365. Now that you can create, code, and debug apps, you need to learn about the infrastructure within a SharePoint farm that addresses these objectives. In this module, you will also learn how to package a completed app, publish it, and manage installation and versions.

Objectives

After completing this module, you will be able to:

- Explain how SharePoint manages app publishing and distribution.
- Describe the contents of an app package.
- Publish apps to a corporate catalog or the Office Marketplace.
- Install, update, and uninstall apps.

Lesson 1

Understanding the App Management Architecture

If you want to publish an app for the users of an on-premises SharePoint farm, you can place the app in the app catalog. If you want to publish the app for all SharePoint users in any on-premises farm and in hosted SharePoint deployments, including Office 365, you can place the app in the Office Store. In this lesson, you will learn about these app publishing locations and the service applications that support them.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the App Management service application and the Subscription Settings service application and how they enable app publishing and maintenance.
- Choose a publishing location for a scenario.

Service Applications for App Management

There are two service applications that must be installed and running in your SharePoint system to support the app infrastructure:

- *The App Management Service.* This service application has a database that stores configuration data for the apps that are installed in your farm. For each app, the App Management Service stores the app domain, app principal, app identifier, licenses, and other configuration information.
- *The Site Subscription Settings Service.* This service application manages tenancies. For each tenancy, the Site Subscription Service stores configuration data such as the tenancy ID, which is used as part of each app identity. Because apps are always run within the scope of a single tenancy, this service application is essential for the app infrastructure.

```

• The App Management Service
  · Create by using Central Administration or Windows PowerShell

• The Site Subscription Settings Service
  · Create by using Windows PowerShell

$ApplicationPool = New-SPServiceApplicationPool -Name "Site Subscription Settings Pool" -Account CONTOSO\Administrator
$SiteSubService = New-SPSubscriptionSettingsServiceApplication -Name "Subscription Settings Service" -DatabaseName "SubscriptionSettingsServiceDatabase" -applicationpool $applicationPool
$AppProxy = New-SPSubscriptionSettingsServiceApplicationProxy -ServiceApplication $SiteSubService

```

If you have subscribed to the Office 365 cloud platform, these service applications are already in place and running. You do not need to configure or modify them because such tasks are the responsibility of Microsoft staff.

However, if you administer an on-premises farm, these service applications may not be present, and you must create them. To create the App Management service, you can use Central Administration:

From the Central Administration home page, under **Application Management**, click **Manage Service Applications**.

Click **New**, and then click **App Management Service**.

Configure the name, database, failover server, application pool, and service proxy settings and then click **OK**.

To create the Site Subscription Settings Service, you must use Windows PowerShell. The following code shows how to create a new service in its own application pool and with a new service proxy.

Configuring the Site Subscription Settings Service

```
# Create a new application pool
$ApplicationPool = New-SPServiceApplicationPool -Name "Site Subscription Settings Pool" -Account
CONTOSO\Administrator

# Create a new Site Subscription Settings service application
$SiteSubService = New-SPSubscriptionSettingsServiceApplication -Name "Subscription Settings
Service" -Databasename "SubscriptionSettingsServiceDatabase" -applicationpool $ApplicationPool

# Create a new service application proxy
$ServiceApplicationProxy = New-SPSubscriptionSettingsServiceApplicationProxy -ServiceApplication
$SiteSubService
```

Publishing Locations

There are two possible locations where you can publish an app to make it available for SharePoint users to install: the Office Store and the app catalog. Before you choose a publishing location, you must understand which users can see your app in each location and which users can install your app.

After you complete an app's development, you create a single file called an app package for deployment. For SharePoint-hosted and auto-hosted apps, this package contains all the files and resources the app uses. For provider-hosted apps, you must set up a remote web yourself, but an app package is still required for SharePoint resources, such as the app manifest file. In all cases, you publish your app by uploading the app package to the Office Store or to an app catalog.

App packages are published in:

- The Office Store
 - For global access
- An app catalog
 - For access from the same web application

The Office Store

The Office Store is a global list of apps that Office customers can use to find, purchase, and install extensions to the Office suite. You can find three types of app in the Office Store:

- *Apps for SharePoint.* Apps are the SharePoint-hosted, auto-hosted, or provider-hosted extensions to SharePoint functionality that you have seen in this course.
- *Office apps.* Office apps are extensions to the desktop and web-based Office applications. For example, an Office app might present a new task pane in Word, a new mail edit pane in Outlook, or a new content box in Excel.
- *Windows Azure Catalog apps.* Windows Azure Catalog apps are Software as a Service (SaaS) applications built on Azure technologies such as Windows Azure services and SQL Database.

When you publish an app in the Office Store, users all over the world in both on-premises and cloud-based SharePoint systems can obtain your app. Publish your app in the Office Store if you want it to be globally available.

The app catalog

An app catalog is a specialized document library within a single SharePoint web application. The app catalog stores only SharePoint apps. Apps in the app catalog are visible only to users within the same web application.

If your SharePoint tenancy is within Office 365, a single app catalog is automatically created for all your users when you subscribe to the service. Place app packages in this catalog to publish them to all users in your organization.

In an on-premises farm, there is no default app catalog created at installation. Instead, you must create one by using Central Administration or Windows PowerShell. If you have multiple web applications, you can create one app catalog for each web application.

Users can browse and install apps from the app catalog. They can also request apps from the Office Store. This helps to enable users without app installation permissions to obtain the apps they need with the help of SharePoint administrators. SharePoint users who cannot install an app from the Office Store can add a request for that app to the app catalog. An administrator, with the required permission, can later assess the request and, if it is approved, install the app from the Office Store.

You will see how to create an app catalog later in this module.

Lesson 2

Understanding App Packages

App packages are single files that contain compressed resources for apps. You can explore these resources by changing the file name extension from .app to .zip and then using File Explorer in the same way as you would explore a .zip file. The contents of a package vary, depending of the hosting model you chose and the app design. In this lesson, you will learn about the format and contents of app packages.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the format of an app package file.
- Describe the typical content in a SharePoint-hosted app package.
- Describe the typical content in an auto-hosted app package and the typical content in a provider hosted app package.
- Explore the contents of an app package.

App Package Format

The app package is a single file with an .app file name extension that you can create by using the Publishing Wizard in Visual Studio. The package file is based on the Open Package Convention (OPC) format, which you may be familiar with because it is also used for Office documents such as .docx and .xlsx files.

The package file contains the following files:

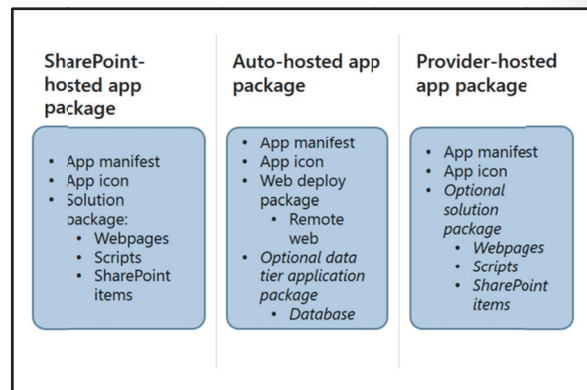
- *AppManifest.xml*. The app manifest file includes configuration settings such as the app start page, other entry points, and permission requests. Every SharePoint app package must include an app manifest file.
- *AppIcon.png*. The app icon file is optional but highly recommended for all apps. SharePoint displays this icon for the app in the host web, and it can contain the app logo and branding. Include the path to the app icon in the app manifest.
- *The app web solution package*. If your app includes resources to install in the app web, the app package will include a SharePoint solution file. For example, in a SharePoint hosted app, you may want to create a new list, a new content type, and a range of new site columns in the app web. The webpages, script files, and style sheets must also be deployed to the app web. In a remote-hosted app, you can also choose to create resources in the app web. Visual Studio zips such resources into a standard SharePoint solution file with a .wsp file name extension. This solution file is itself zipped into the app package. SharePoint-hosted apps always have app web solution files in the app package because they always install features in the app web.
- *Host web features*. If you include other entry points to your app, such as app parts and UI custom actions, the app package must install these as features in the host web. In the app package, each app part or custom action is represented by a feature XML file and an elements XML file.

Common app package contents

- AppManifest.xml
 - Required for all apps
- AppIcon.png
 - Recommended for all apps
- App web solution
 - Required for SharePoint-hosted apps
 - Optional for remote-hosted apps
- Host web features
 - Optional for all apps

Packages for Remote-Hosted Apps

An app package for a SharePoint-hosted app always contains an app manifest file and usually an app icon. The resources that the app installs into the app web are encapsulated in a solution file and zipped into the app package. These resources include the webpages and style sheets that make up the user interface for the SharePoint-hosted app and the JavaScript files that implement the functionality. There may also be lists, libraries, content types, site columns, and other SharePoint items in the solution file. The app package for a SharePoint-hosted app contains all the files and resource that make up the app.



The contents of the app file for remote-hosted apps are different, depending on the app model you have selected.

Packages for auto-hosted apps

Like packages for a SharePoint-hosted app, packages for auto-hosted apps contain everything that is required to deploy a new instance of the app. Remember that a deployed auto-hosted app can include the following components:

- *Remote web.* This is a website automatically deployed to Windows Azure during app installation. All the webpages and code that make up the app user interface and functionality are stored in the remote web. A remote web is required for an auto-hosted app.
- *App web.* Optionally, you can include an app web in an auto-hosted app. This enables you to create SharePoint lists, libraries, and other items that are dedicated to the app.
- *Database.* Optionally, you can include a database to store structured data for your app. The database will be automatically deployed as a new Windows Azure SQL Database when a user installs your app.

To package the remote web, Visual Studio creates a .zip file called a web deploy package. This format is used by Visual Studio to deploy any website to Windows Azure. The web deploy package has a .web.zip file name extension and is included in the app package.

To package the app web, if you are using one, Visual Studio creates a SharePoint solution file, with a .wsp file name extension, just like the app web for a SharePoint-hosted app. The solution file is included in the app package.

To package the database, Visual Studio uses a special file format called a data tier application package. This is an XML file zipped with a .dacpac file name extension. The .dacpac file includes XML data to define all the tables, columns, views, indexes, stored procedures, and other database objects required.

Packages for provider-hosted apps

App package files for provider-hosted apps are different than auto-hosted and SharePoint-hosted apps because they do not include all the files, items, and resources necessary to install the application. Remember that, in a provider-hosted app, the provider is responsible for deploying and maintaining the remote web and any database the remote web uses. When users install your app, they configure SharePoint to connect to the start page in the remote web when users click on the app icon. Therefore, you can consider that an app package for a provider-hosted app contains the information needed to make that connection.

Provider-hosted app packages must include the app manifest. This defines the start page, any other entry points, permission requests, and other properties of the install app. You should also include an app icon file to be presented in the host web.

As for auto-hosted apps, you can choose to create an app web in a provider-hosted app. This app web can contain lists, libraries, and so on, that are dedicated to the app and will be deleted if the app is uninstalled. The app web will be encapsulated as a SharePoint solution file and zipped into the app package.

Demonstration: Exploring an App Package

In this demonstration, you will see how to:

- Explore the contents of a SharePoint app package by renaming the .app file.
- Explore the contents of a solution file within an app package by renaming the .wsp file

The app package explored in this demonstration is for a SharePoint-hosted app.

Demonstration Steps

- Start the **20488B-LON-SP-10** virtual machine, if it is not already running.
- Log on as **CONTOSO\Administrator**, with the password **Pa\$\$w0rd**.
- Open File Explorer, and browse to **E:\Democode**.
- In File Explorer, right-click **SiteSuggestionsApp.app**, and then click **Rename**.
- Select the file name extension **.app**, type **.zip**, and then press Enter.
- In the **Rename** dialog box, click **Yes**.
- Right-click **SiteSuggestionsApp.zip**, and then click **Extract All**.
- In the **Extract Compressed (Zipped) Folders** dialog box, click **Extract**.
- Double-click **AppIcon.png**.



Note: Windows Photo Viewer displays the app icon file.

- Switch to File Explorer.
- Right-click **AppManifest.xml**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.



Note: Visual Studio displays the contents of the app manifest.

- Switch to File Explorer.
- Right-click **SiteSuggestionsApp.wsp**, and then click **Rename**.
- Select the file name extension **.wsp**, type **.cab**, and then press Enter.
- In the **Rename** dialog box, click **Yes**.
- Double-click **SiteSuggestionsApp.cab**.
- Right-click **Feature.xml**, and then click **Extract**.
- In the **Select a Destination** dialog box, browse to **E:\Democode**, and then click **Extract**.

- In File Explorer, browse to the folder **E:\Democode**.
- Right-click **Feature.xml**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- Close Visual Studio.
- Close File Explorer.
- Close Windows Photo Viewer.

Lesson 3

Publishing Apps

The Visual Studio Publishing Wizard helps you to create app packages, which you can place in the Office Store or an app catalog. From these locations, users can install your app in their own host webs. In this lesson, you will see how to complete pages in the Publishing Wizard and how to set up Office Store seller accounts and app catalogs.

Lesson Objectives

After completing this lesson, you will be able to:

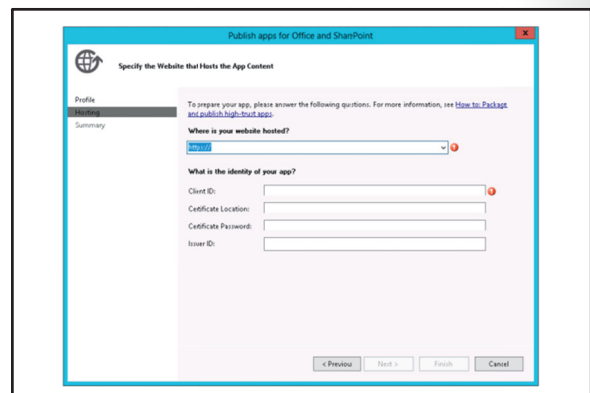
- Use the Visual Studio Publishing Wizard to package and publish apps.
- Publish an app to the Office Store and configure licenses.
- Set up an app catalog in a SharePoint tenancy and use it to distribute apps.

The Publishing Wizard

The Publishing Wizard for apps creates app packages. You can start the Publishing Wizard by clicking **Publish** on the Visual Studio **BUILD** menu, or by right-clicking the app web project and then clicking **Publish**. The pages that you must complete in the Publishing Wizard depend on the hosting model that your app uses:

- *For SharePoint-hosted and auto-hosted apps.* No configuration is required in the Publishing Wizard, which displays only a summary page. You can choose to open the folder where the wizard places the app package after it is completed, and then click **Finish**.
- *For provider-hosted apps.* The app package must include all the information required for SharePoint to connect to the remote web and trust the remote web server for authentication. The wizard displays the following pages:
 - *The Profile page.* Each time you use the Publishing Wizard, settings are stored in a publishing profile, which is saved as part of the Visual Studio remote web project. On the Profile page of the Publishing Wizard, you can choose to create a new profile or use a profile you created on a previous occasion. Publishing profiles make it easy to recall settings that you used for previous versions of the app.
 - *The Hosting page.* On the Hosting page, you configure connection and trust details for the app. You must provide a URL where the app start page can be found. In addition, to set up an S2S trust relationship, you must provide the client ID, private certificate path, certificate password, and issuer ID. These values will be stored in the Web.config file for the app.
 - *The Summary page.* On the Summary page, review your settings. If everything is correct, click **Finish**. Visual Studio creates the app package.

Visual Studio places the app package in the **Bin** folder within the project hierarchy. Within this folder, the path depends on the build configuration for the project and the version number in the app manifest file.




For example, if you select the **Release** build configuration and the app manifest file version number is **1.0.0.0**, the Publishing Wizard creates the app package in **/bin/Release/app.package/1.0.0.0**.

Publishing to the Office Store

The Office Store is a global catalog of app for SharePoint, Office desktop applications, Office web applications and Windows Azure. A user with a subscription to any of these products can use the Office Store to locate, purchase, and install apps to extend functionality. For developers, the Office Store provides a global marketplace in which you can easily promote, sell, and license your app.

SharePoint users who have the necessary permissions can purchase and install an app from the Office Store into their SharePoint tenancy, whether that tenancy is in Office 365 or in an on-premises SharePoint farm. If you do not have the appropriate permission, you can create an app request in your local app catalog. Your tenancy administrators can see this request and may choose to install the app for you.

- The Office Store is a global catalog of apps for SharePoint, Office, and Azure
- A dashboard seller account is required
- Upload your apps and provide:
 - A title
 - A version number
 - A description
 - A category
 - A logo
 - Screen shots
- Microsoft tests and approves the quality of all apps

 **Additional Reading:** For more information about apps for SharePoint in the Office Store, see *Office Store* at <http://go.microsoft.com/fwlink/?LinkID=307097>.

Dashboard seller account

Before you can publish an app in the Office Store, you must create a dashboard seller account. You can create this account by visiting <https://sellerdashboard.microsoft.com/Registration>. You must log on to the site by using a Windows Live account. Any account that you use to access Outlook.com mail, SkyDrive, or Windows Azure can be used for this purpose. After you log on, create the seller account by providing personal and company details, such as contact information and the company website.

The information you provide must be verified and approved by Microsoft before you can begin publishing apps. This process can take two or three days.

In the dashboard seller account, you can provide Microsoft with the details of a bank or PayPal account. If you publish paid-for apps, Microsoft collects payments when users install your apps and automatically credits this revenue to your account. Note that paid-for apps are not the only way to generate revenue from your app. For example, you might consider a free app that includes advertising to generate revenue.

Uploading app packages


After your dashboard seller account is approved, you can upload app packages to be published in the Office Store. For each app, you must provide the following information:

- *Title*. This is the title that the app is displayed under in the Office Store. Usually, this title will match the title you configured in the app manifest file.
- *Version number*. This number is used to help users identify when an upgrade is available. Again, this number usually matches the version number in the app manifest.
- *Description*. Use this field to describe your app to users and convince them to install it.

- **Category.** Categories help users to locate apps by browsing through apps with similar purposes. For example, current categories in the Office Store include Communication, Social, Project Management, and Public-Facing Web Sites.
- **Logo.** You should include a distinctive logo for your app that reflects its functionality and your brand.
- **Screen shot.** You must provide at least one screen shot. It is advisable to provide a screen shot for each function of your app so that potential customers can see what it does.
- **Licensing fee.** If you choose to publish a paid-for app, you must configure a fee for each user or for each group of users. You can also specify a free trial period.


App approval

Microsoft is committed to helping ensure that the Office Store contains only apps of high quality. To achieve this, there is a rigorous approval process for each app. After you upload your app, it may take several days for it to be approved.

 **Additional Reading:** For information about app validation policies, see *Validation policies for the apps submitted to the Office Store (Version 1.4)* at <http://go.microsoft.com/fwlink/?LinkID=307098>.

App Validation Policies

<http://go.microsoft.com/fwlink/?LinkID=307098>

 **Note:** This link is to the most up-to-date validation policies at the time of writing (version 1.4). These policies may change at any time. Make sure you consult the most recent validation policies as you prepare your app for publication.

After your app is approved, it is fully published and appears in the Office Store. Customers can purchase, download, and install the app.

Publishing to an App Catalog

If you publish an app to the Office Store, it becomes available to all SharePoint tenancies, including both Office 365 and on-premises SharePoint farms. Sometimes you might prefer to publish your app to a smaller range of users. For example:

- You have written an app specifically for the users of your own company, or for the users of a single company that hired you.
- You have written an app for a team of sales people to sell to customers who already use SharePoint.
- You have written an app for a SharePoint consultancy. The consultancy will include the app with the on-premises farms they install at customer sites as one of their unique selling points.

- You can use an app catalog to publish an app to the users of a single tenancy or web application
- Use Central Administration to create a new app catalog site
- Upload app package files to the app catalog and complete the metadata to publish apps

For such situations, you can publish the app in an app catalog. An app catalog is a specialized SharePoint site in which you place the app package files from Visual Studio. In an on-premises farm, you can create one app catalog for each web application. Users can install the apps in the app catalog within any host web in that web application. In Office 365, there is one app catalog for all the site collections and sites in your tenancy.

Creating an app catalog

In an on-premises farm, there is no app catalog created by default. You must create an app catalog for each web application. You can create the app catalog by following these steps:

1. In the Central Administration home page, under **Apps**, click **Manage app catalog**.
2. Choose **Create a new app catalog site**, and then click **OK**.
3. Enter a **Title** and a **Description** for the app catalog.
4. Choose a **Web site address** for the app catalog.
5. Choose a **Primary site collection administrator** for the app catalog. These users have full control over the app catalog contents and properties.
6. Under **End Users**, choose a set of users or groups who have read access to the app catalog. These users can install apps from the catalog.
7. Choose a **Quota template** for the site and then click **OK**.

Only administrators of the web application can create an app catalog site.

Publishing apps in an app catalog

There are two document libraries in an app catalog:

- *Apps for SharePoint*. Upload your SharePoint app package, which you created in Visual Studio, to this document library. SharePoint reads some properties for the app from the app manifest file, but you must manually provide some other metadata.
- *App for Office*. Upload Office desktop apps to this document library.



Note: Because there is one app catalog for each web application, you can manage the visibility of apps by installing them in different app catalogs. Each user can only see the apps that are installed in the app catalog in the web application that they are currently using.

This technique is only possible in on-premises SharePoint farm because, in an Office 365 tenancy, there is only one app catalog for all the users in your tenancy.

Because there is one app catalog for each web application, you can manage the visibility of apps by installing them in different app catalogs.

Lab A: Publishing an App to a Corporate Catalog

Scenario

The Site Suggestions app that you developed earlier in this course is now ready to be distributed on the Contoso intranet. To help ensure that everything goes smoothly, you want to run through the publishing process. In this lab, you will create a new app catalog on your development machine and publish the Site Suggestions app to it.

Objectives

After completing this lab, you will be able to:

- Create a new app catalog for a SharePoint on-premises web application.
- Use the Visual Studio Publishing Wizard to create an app package.
- Publish the app package in an app catalog.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-10
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

Exercise 1: Creating an App Catalog

Scenario

Before you can publish an app in your SharePoint farm, you must create an app catalog in which to publish it. In this exercise, you will create the catalog.

The main tasks for this exercise are as follows:

1. Create an App Catalog

► Task 1: Create an App Catalog

- In Central Administration, navigate to the **Manage App Catalog** page.
- Create a new app catalog site. Use the following information:
 - Title: Contoso Intranet App Catalog
 - Description: Publish apps here for use on the Contoso Intranet
 - Website address: <http://london/sites/appcatalog>
 - Primary Site Collection Administrator: CONTOSO\Administrator
- Browse to the new app catalog. Verify that there are no apps published.
- Close Internet Explorer.

Results: A SharePoint farm with an app catalog set up in the default web application.

Exercise 2: Creating an App Package

Scenario

In this exercise, you will create a new package for the Site Suggestions app.

The main tasks for this exercise are as follows:

1. Prepare the App for Publishing
2. Package the App

► Task 1: Prepare the App for Publishing

- Open the following Visual Studio solution:
 - E:\Labfiles\LabA\Starter\SiteSuggestionsApp.sln.
- In the AppManifest.xml file, set the version number for the app to 1.0.0.1.
- Save your changes.
- Configure the solution to use the **Release|Any CPU** configuration.
- Build the Site Suggestions app solution.

► Task 2: Package the App

- Note the contents of the following folder: E:\Labfiles\LabA\Starter\SiteSuggestionsApp\bin\Release
- Package the **SiteSuggestionsApp** project. Do not select **Open the output folder after successful packaging**.
- Re-examine the contents of the **Release** folder and its subfolders.

Results: An app package that can be used to publish the Site Suggestions app in a SharePoint app catalog.

Exercise 3: Publishing an App Package

Scenario

Now that you have an app catalog and an app package, you can publish the package in the app catalog. When this process is complete, users can see and install your app in their host webs.

The main tasks for this exercise are as follows:

1. Publish the App
2. Test the App Catalog

► Task 1: Publish the App

- In Internet Explorer, browse to the app catalog site you created in Exercise 1.
- Publish the **SiteSuggestionsApp.app** package you created in the last exercise. Use the following information:
 - Short description: This app collects feedback on SharePoint sites for users.
 - Long description: Users can submit new ideas for each site and vote on other people's ideas
 - Category: Site Administration
 - Publisher Name: Contoso

- Featured: True

► **Task 2: Test the App Catalog**

- View the site contents for the **<http://dev.contoso.com>** SharePoint site.
- View the list of apps that you can add to the site from your organization.
- Close all applications.

Results: A published SharePoint hosted app in a corporate app catalog.

Question: Where does the app catalog look for the version number of the app you upload?

Question: A developer, working on a new app for the marketing department, has shown a user the latest features of the app, which is approaching completion. The user wants to install the app in the marketing site, but when he clicks **add an app**, he cannot find the app in any of the lists. What should you tell the user?

Lesson 4

Installing, Updating, and Uninstalling Apps

After an app is published to the Office Store or to an app catalog, users with appropriate permissions can browse and install it in their host webs. There are two other import events to consider in the life cycle of an app: update and uninstall. If you have a new version of your app, you upload it to the Office Store or app catalog. Users are informed of the new version and can choose to update their instance of the app. If a user chooses to uninstall your app, the removal is clean, and no app code or custom objects remain in the SharePoint host web. In this lesson, you will learn how to manage the app life cycle and its events.

Lesson Objectives

After completing this lesson, you will be able to:

- Install apps from the Office Store or an app catalog.
- Manage version numbers and upgrades for an app.
- Uninstall an app by using the SharePoint web interface or a Windows PowerShell script.

App Installation

To install a new app, users click the **Add an app** link in the **Site Contents** page for their host web. This displays the **Apps you can add** page. The list of apps includes a range of default apps, such as a new document library, tasks list, or wiki page library. If the current web application has an app catalog, apps in the **App for SharePoint** document library will be included in the list of **Apps you can add**. To install the app, click its entry in the list and give the new instance a name. SharePoint installs the app and configures a shortcut in the **Site Contents** page.

- Installation procedures
- Installation permissions
- Agreement to permission requests
- Installation with Windows PowerShell:


```
$web = "http://intranet.contoso.com/"
$apppackage = "C:\Apps\MyApp.app"
$appsource = ([microsoft.sharepoint.administration.spappsource]
::CorporateCatalog)
$spapp = Import-SPAppPackage -Path $apppackage -Site $web
           -Source $appsource -Confirm: $false
Install-SPApp -Web $web -Identity $spapp -Confirm: $false
```

The **Apps you can add** page also includes a link to the **SharePoint Store**. This list includes all the SharePoint apps from the Office Store in a list that you can browse. For each app in the store, you can see details such as the description and screen shots the vendor included when publishing the app to the store. You can also see ratings and reviews from other users of each app. To install a chosen app from the store, click its **Add It** button.

Installation permissions

To install an app, a user requires the following permissions:

- *Read access to the app catalog.* If the app is in the app catalog, the user must have read permission in the catalog to be able to browse for the app.
- *Full control access to the host web.* A user must be an administrator in the site where the app will be installed. This is the site that will become the host web.
- *Equal or greater permission than those requested by the app.* Each app requests the permissions it requires according to the requests you have configured in its app manifest file. For example, the app can request write access to the tenancy, read access to managed metadata, or full control access to the social microfeed. The installing user must have permissions that equal or exceed those requested by the app for the installation to succeed.

Agreement to permission requests

When a user installs your app from any source, the user must agree to the permissions you requested in the app manifest file. A page is displayed with the title "Do you trust *App Name*?" On this page, a sentence appears for each permission request you included. For example, consider the following request:

```
<AppPermissionRequest Scope="http://sharepoint/content/sitecollection" Right="Read" />
```

If you included this permission request in your app manifest file, the install page includes the sentence "Let it read items in this site collection."

The installing user should check all the requested permissions thoroughly. If the user agrees with the permissions, he or she can click **Trust It**. SharePoint will install the app.

You can use the **Install-SPApp** command to install an app from the app catalog. In the following example, an app package is both uploaded to the app catalog and then installed.

Using Windows PowerShell to Register and Install an App

```
# Set up required strings
$web = "http://intranet.contoso.com/"
$apppackage = "C:\Apps\MyApp.app"
$appsource = ([microsoft.sharepoint.administration.spappsource]::CorporateCatalog)

# Import the app to the app catalog
$spapp = Import-SPAppPackage -Path $apppackage -Site $web -Source $appsource -Confirm: $false

#Install the app
Install-SPApp -Web $web -Identity $spapp -Confirm: $false
```

Updating an App

Whether you choose to publish your app in the Office Store or in an app catalog, it is likely that you will want to make improvements after the first version is published. For example, you may want to implement a different user interface based on user feedback or build extra functionality or features.

The app manifest file includes a version number for the app. When you publish a new version of the app, you should increment this number. SharePoint will alert users to the new version the next time they use the app. The user can choose to apply the upgrade or choose to continue using the original version.

- Upgrades are based on the version number in the app manifest
- Upgrades are voluntary
- Upgrades are simplified because SharePoint-hosted apps and auto-hosted apps are self-contained
- For provider-hosted apps, make sure that the remote web supports both old and new versions

Upgrading SharePoint-hosted and auto-hosted apps

Upgrades for SharePoint-hosted and auto-hosted apps are simple because each tenancy is automatically isolated from all others.

In a SharePoint-hosted app, all resources are stored in the app web. When a new version becomes available, if the user installs the new version, upgrades are applied and the user sees the new functionality. Alternatively, if the user chooses to ignore the upgrade, they can continue working with the original version of the app. Because all resources are in a separate app web, there is no difficulty with supporting old and new versions at the same time.

The same is also true of auto-hosted apps. Because the remote web and the database are hosted in the tenant's Windows Azure account, they can choose to apply the upgrade or continue using the old version. Again, you need not take any action to support both versions.

Upgrading provider-hosted apps

Upgrades can be more complex with provider-hosted apps. Remember that the provider is responsible for the remote web, which is a website on IIS, Windows Azure, or some other web server. The app manifest file provides a link to the app start page and, optionally, any other app entry points.

To implement an upgrade, you must deploy a new version of the remote web. Then publish a new version of the app package, with an incremented version number, in the Office Store or app catalog.

The complexity arises because users are not forced to accept the upgrade; instead, they can choose to continue using the old version. For this reason, you must make sure that the remote web can distinguish versions of the app and present the appropriate user interface for each. The old version must be supported indefinitely or at least until you can be sure that all users have upgraded. If you do not handle this complexity correctly, users can be left with an out-of-date version of your app that does not work and cannot be upgraded.

One simple way to support multiple versions is to use separate remote webs. For example, if the version 1 remote web is at <http://apps.contoso.com/myappv1>, you could publish the upgraded remote web at <http://apps.contoso.com/myappv2>. The upgraded app manifest file includes the second URL for the start page and a higher version number. As long as you maintain both remote webs, you can support both versions.

Uninstalling Apps

Removing an app from a host web is a straightforward operation. In the **Site Contents** page, click the ellipsis next to the app title, and then click **Remove**. For SharePoint-hosted apps, and any remote-hosted app that includes an app web, SharePoint deletes the app web and all the app entry points. Shortcuts are removed from the host web and app permissions no longer apply.

When a user removes a remote-hosted app, no changes are made to the remote web and its database. Instead, shortcuts are removed from the host web. Permissions no longer apply because the app instance, with its app identifier, is deleted.

You can use the **Uninstall-SPAppInstance** command to remove an app from a host web.

- When you remove an app, the app web is deleted
 - Shortcuts and the app identifier are deleted, so permission no longer apply
 - Provider-hosted content is not deleted
 - You can use Windows PowerShell to remove apps:
- ```
$Web = "http://intranet.contoso.com/"
$appname = "Contoso Ledgers App"
$appInstance = Get-SPAppInstance -Web $Web |
 Where-Object {$_.Title -eq $appname}
Uninstall-SPAppInstance -Identity $appInstance -Confirm:$false
```

## Using Windows PowerShell to Uninstall an App

```
Set up required strings
$web = "http://intranet.contoso.com/"
$appname = "Contoso Ledgers App"

#Get the app instance
$appInstance = Get-SPAppInstance -Web $web | Where-Object {$_.Title -eq $appname}

#Uninstall the app
Uninstall-SPAppInstance -Identity $appInstance -Confirm:$false
```

## Lab B: Installing, Updating, and Uninstalling Apps

### Scenario

Before you release the Site Suggestions app, you want to make sure that the install and uninstall processes work as expected. In this lab, you will install the app on a local SharePoint site. You will then update the app and make sure that the experience for end users is seamless. Finally, you will uninstall the app and verify that it is removed cleanly.

### Objectives

When you have completed this lab, you will be able to:

- Install an app from the corporate catalog.
- Deploy an update to an installed app.
- Uninstall an app.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-10
- Username: CONTOSO\Administrator
- Password: Pa\$\$w0rd

### Exercise 1: Installing an App

#### Scenario

In this exercise, you will install the Site Suggestions app from the app catalog. You will also verify that the app works as expected and installs the resources you expect in the SharePoint app web.

The main tasks for this exercise are as follows:

1. Install the Site Suggestions App from the App Catalog
2. Test the App

#### ► Task 1: Install the Site Suggestions App from the App Catalog

- Use Internet Explorer to browse the **http://dev.contoso.com** SharePoint site.
- Browse the list of apps **From Your Organization**.
- Add and trust the Site Suggestions app.

#### ► Task 2: Test the App

- Browse to the Site Suggestions app start page. If you are prompted to authenticate, log on as Administrator.
- Add a new site suggestion. Use the following information:
  - Subject: Test Suggestion
  - Feedback: This is just to test this deployment.
- View the new suggestion.
- Close Internet Explorer.

**Results:** A deployed and functional SharePoint-hosted app.

## Exercise 2: Upgrading an App

### Scenario

The development team has completed the voting functionality for the Site Suggestions app. You want to deploy this new version for all users and enable them to upgrade from version 1.0.0.1 of the app.

The main tasks for this exercise are as follows:

1. Repackage Version 2
2. Update the App Catalog
3. Update a Deployed App

#### ► Task 1: Repackage Version 2

- Open the following solution:
- E:\Labfiles\LabB\Starter\SiteSuggestionsApp.sln
- In the app manifest, set the version number to 2.0.0.0
- Configure the solution to use the Release|Any CPU configuration.
- Run the Publishing Wizard a second time.

#### ► Task 2: Update the App Catalog

- In Internet Explorer, browse to the app catalog for the Contoso intranet.
- Publish the **SiteSuggestionsApp.app** package you created in the last exercise.

#### ► Task 3: Update a Deployed App

- In Internet Explorer, browse to the **Site Contents** list for the **http://dev.contoso.com** site.
- Install and trust version 2.0.0.0 of the Site Suggestions app.
- Add a new site suggestion. Use the following information:
- Subject: Testing version 2
- Feedback: Version 2 includes voting functionality.
- Vote for the **Testing version 2** suggestion. Click the suggestion again to display the net votes.

**Results:** An upgraded app deployed through an app catalog.

## Exercise 3: Removing an App

### Scenario

You are happy with the install and update process for the Site Suggestions app. In this exercise, you will remove the Site Suggestions app from the app catalog and remove the deployed instance of the Site Suggestions app.

The main tasks for this exercise are as follows:

1. Remove the App from the App Catalog
2. Remove the Deployed Instance of the Site Suggestions App

#### ► Task 1: Remove the App from the App Catalog

- In Internet Explorer, navigate to the **Apps for SharePoint** list in the app catalog.

- Delete the **SiteSuggestionsApp** from the **Apps for SharePoint** list.
  - In the **http://dev.contoso.com** site, view the list of apps you can install from your organization. **SiteSuggestionsApp** should not appear in the list.
- ▶ **Task 2: Remove the Deployed Instance of the Site Suggestions App**
- Access the **SiteSuggestionsApp** in the Contoso intranet site. Notice that the instance of the app remains deployed even though the app has been removed from the app catalog.
  - Remove the **SiteSuggestionsApp** from the Contoso intranet host web.
  - Close all open applications.


**Results:** A SharePoint site with no custom apps installed or available to install.


**Question:** The Site Suggestions app is a SharePoint-hosted app. When a user installs the app, where does the app create resources?

**Question:** When you remove an app from the app catalog, what happens to the deployed instances of that app?

## Module Review and Takeaways

When Microsoft designed the SharePoint app infrastructure, a high priority was placed on the manageability of apps. To address this priority, Microsoft has enabled app providers to publish apps in app catalogs and the Office Store. From these locations, users can easily search for and identify apps that address their own unique needs in SharePoint sites. After an app is found, a user can easily install in a host web or request that an administrator install it. Furthermore, it is easy to update to a new version of an app and remove an app that is no longer required.

 **Best Practice:** Consider publishing a finished app in the Office Store, even if the app was developed for a specific customer with unusual requirements. If you publish the app to the global SharePoint community in this way, you can generate extra return on the investment made by you or by the customer who paid for the app development.

 **Best Practice:** Take care to set version numbers in the app manifest file precisely and logically. The upgrade process relies on these version numbers. If you do not manage the version numbers correctly, users will not be prompted to install your new functionality and will continue to use an out-of-date app.

### Review Question(s)

**Question:** You have been hired by a firm of lawyers to create a specialized document management SharePoint app. The app is SharePoint-hosted and enables users to track the progress of cases through multiple courts. Should you publish the app in an app catalog or the Office Store?

Verify the correctness of the statement by placing a mark in the column to the right.

| Statement                                                                                                    | Answer |
|--------------------------------------------------------------------------------------------------------------|--------|
| True or False: To prevent any further use of an app, you can remove it from the app catalog or Office Store. |        |

### Test Your Knowledge

| Question                                                                                                             |                                                     |
|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Which of the following service applications must be in place for you to publish and deploy apps from an app catalog? |                                                     |
| Select the correct answer.                                                                                           |                                                     |
| <input type="checkbox"/>                                                                                             | The Application Discovery and Load Balancer Service |
| <input type="checkbox"/>                                                                                             | The Site Subscription Settings Service              |
| <input type="checkbox"/>                                                                                             | The Managed Metadata Service                        |
| <input type="checkbox"/>                                                                                             | The Secure Store Service                            |
| <input type="checkbox"/>                                                                                             | The Security Token Service                          |

Verify the correctness of the statement by placing a mark in the column to the right.

| Statement                                                                                                        | Answer |
|------------------------------------------------------------------------------------------------------------------|--------|
| True or False: The app package for an auto-hosted SharePoint app must include a SharePoint solution (.wsp) file. |        |



# Module 11

## Automating Business Processes

### Contents:

|                                                                                      |       |
|--------------------------------------------------------------------------------------|-------|
| Module Overview                                                                      | 11-1  |
| <b>Lesson 1:</b> Understanding Workflow in SharePoint 2013                           | 11-2  |
| <b>Lesson 2:</b> Building Workflows by using Visio 2013 and SharePoint Designer 2013 | 11-7  |
| <b>Lab A:</b> Building Workflows in Visio 2013 and SharePoint Designer 2013          | 11-16 |
| <b>Lesson 3:</b> Developing Workflows in Visual Studio 2012                          | 11-20 |
| <b>Lab B:</b> Creating Workflow Actions in Visual Studio 2012                        | 11-25 |
| Module Review and Takeaways                                                          | 11-29 |

## Module Overview

By automating business processes, you can help ensure they are completed correctly, with each step being completed by the correct person at the correct time. SharePoint 2013 supports automation by enabling you to model your business processes and create workflows that notify users of tasks, move documents through a process, and improve overall efficiency.

### Objectives

At the end of this module, you will be able to:

- Describe the architecture and capabilities of workflow in Microsoft SharePoint 2013.
- Create declarative workflows in Microsoft Visio 2013 and SharePoint Designer 2013.
- Create and deploy custom workflows by using Microsoft Visual Studio 2012.

## Lesson 1

# Understanding Workflow in SharePoint 2013

Before you can begin to develop workflows for SharePoint, you need to understand the architecture of the workflow platform and how the workflow components work together both in and outside of SharePoint.

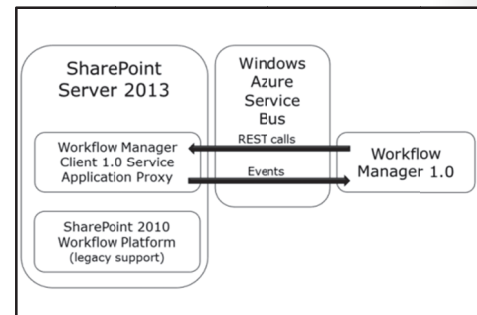
## Lesson Objectives

At the end of this lesson, you will be able to:

- Describe the SharePoint workflow platform.
- Describe how associations and subscriptions work.
- List the types of SharePoint 2013 workflow.
- List the tools available for developing SharePoint 2013 workflows.

## Introduction to the SharePoint Workflow Platform

SharePoint 2013 introduces a new workflow execution host, Workflow Manager Client 1.0, which provides advantages over the earlier workflow technologies available in SharePoint 2010. It uses Windows Workflow Foundation 4, which is built on top of the messaging functionality that Windows Communication Foundation provides.



## Workflow Manager Client 1.0

Workflow Manager is not part of SharePoint itself, and you install it by using the Web Platform Installer. After installation completes, you can use the Workflow Manager Configuration Wizard to create the databases and services you need to host and execute workflows. If you do not install Workflow Manager, you cannot use SharePoint 2013 workflows; however, you can still create and use SharePoint 2010 workflows, but this is not a recommended approach.

## Interacting with SharePoint 2010 workflows

SharePoint 2013 provides full support for SharePoint 2010 workflows. The SharePoint 2010 workflow platform is still included in SharePoint 2013 and runs as it did in SharePoint 2010. You can use it to run existing workflows that you have developed for SharePoint 2010 and you can call into it from a SharePoint 2013 workflow by using workflow interop. This can be useful if you need to reuse legacy workflows.



**Additional Reading:** For more information about workflow interop, see *Use workflow interop for SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=307099>.

Workflow Manager is a process that runs outside of SharePoint in Windows Azure; therefore, the workflows executing in it run outside of SharePoint. This makes the whole workflow process more scalable than earlier versions because workflows can be hosted on a separate computer to the SharePoint site, and

you can scale workflows independently to SharePoint. Workflow Manager is designed for high volume environments and is built for use in the cloud.

## Communicating with the Workflow Manager Client

SharePoint includes the Workflow Manager Client 1.0 Service Application Proxy, which enables SharePoint to communicate with Workflow Manager Client by using events and Representational State Transfer (REST) calls over the Windows Azure Service Bus. Workflows listen for events occurring on a SharePoint site, for example, the **itemCreated** event, and then execute the workflow actions. Workflows can also use the SharePoint REST API to call back into SharePoint. Because the two platforms, SharePoint and Workflow Manager, are separate, they can be installed on different servers to improve scalability. In this scenario, server-to-server authentication between SharePoint and the Workflow Manager uses the OAuth authentication mechanism.

## Workflow Service Manager classes

The SharePoint object model has also been updated to support the new workflow system and includes a set of new classes, collectively known as the Workflow Service Manager. These classes enable you to manage, deploy, and communicate with workflows from any external code.

The object model provides support for the following services:

- The instance management service, which manages workflow instances and their execution.
- The deployment service, which manages the deployment of workflow definitions.
- The interop service, which manages the interop bridge for supporting SharePoint 2010 workflows.
- The messaging service, which manages message queuing and transport.



**Additional Reading:** For more information about the Workflow Service Manager classes, see *System.Workflow Namespaces* at <http://go.microsoft.com/fwlink/?LinkID=307100>.

## Workflow Associations and Subscriptions

Because the workflow execution in SharePoint 2013 runs outside of the SharePoint farm, the two platforms need to communicate with each other. They do this by using the Windows Azure publication/subscribe service. This is an asynchronous messaging framework where the publishers of messages do not need to know which subscribers are consuming their messages. This system of decoupled publishing and subscribing supports greater flexibility and scalability in the workflow infrastructure.

- SharePoint and workflows communicate by using the Windows Azure publication/subscribe service
- Workflow associations:
  - Bind the definition of a workflow to a scope
  - Represent a set of subscription rules
- Workflow subscriptions:
  - Enable a workflow to interact with associations
  - Respond to events from an object
- Starting workflows:
  - Manually
  - Automatically

### Workflow associations

Workflow associations bind the definition of a workflow to a SharePoint scope with default values. This effectively represents a set of subscription rules that process messages to ensure that the correct workflow instances consume them.

The available workflow scopes are **SPList** for list workflows and **SPWeb** for site workflows.

## Workflow subscriptions

Workflow subscriptions enable workflows to interact with associations. A workflow creates a subscription on the Windows Azure Service Bus by using the **create** method. The method signature of this and the methods to instantiate a subscription are defined by the workflow author in the XAML workflow definition.

Subscriptions are also bound to a scope: **SPList** or **SPWeb**. When you create a subscription, the object type is passed in as a value so that the subscription only responds to events from the object to which they are subscribed.

## Starting workflows

Workflows can be started either manually or automatically, that is, in response to an event.

When a workflow starts in response to an event, SharePoint sends a message to the publication/subscribe service. The message contains the following information:

- The ID of the originating item context
- The event itself
- The workflow initiation parameters

This information enables the relevant subscriptions to respond to the event.

When a user manually starts a workflow, SharePoint sends a message to the publication/subscribe services. The message contains the following information:

- The association identifier
- The ID of the originating item context
- The event type
- The workflow initiation parameters

Like with automatic starts, this information enables the subscribers to respond to the start event.



**Note:** If you configure a workflow to automatically start on a repeatable event, such as the **OnItemChanged** event, it cannot start a second workflow until after the first running instance completes.

## Types of Workflow

SharePoint 2013 supports three types of workflow: list workflows, site workflows, and reusable workflows. The type that you choose to use will depend on where you want the workflow to function and how you want it to start.

### List workflows

You can use list workflows to perform specific processes on items in a particular list. You cannot move a list workflow to another list; however, you can deploy the list and workflow together to another site.

- List workflows
- Site workflows
- Reusable workflows

The key advantage to list workflows is that they can access list columns and use the information from a column within the workflow. For example, reading or updating the title of a document.

### Site workflows

You can use site workflows to perform administrative tasks on a site. They are published to a SharePoint site and are not associated with a particular content type or list; therefore, they cannot be triggered by events and must either be started by a user or programmatically.

### Reusable workflows


Reusable workflows have the flexibility that enables you to reuse them across multiple sites and lists. You can even create the workflow on one SharePoint farm, save it as a template, and then copy and deploy the solution to another farm. You can export workflows for reuse in other site collections, web applications, as well as other SharePoint farms. Because you can package the workflow and all its dependencies together, you can also deploy a workflow as a sandbox solution to a hosted SharePoint environment.

A special case of reusable workflow is the global reusable workflow. If you create your workflow at the top-level site of a site collection, you can then use that workflow across the entire site collection.

## Workflow Development

Workflows in SharePoint 2013 are fully declarative, meaning that they are defined by using XAML, which is then interpreted at run time.

You can develop SharePoint 2013 workflows in a variety of tools, the choice of which is generally determined by the author's existing skills set, their main focus, and their job requirements. For example, a business analyst is likely to use Visio 2013 to develop the high-level business focused steps in a workflow, whereas a developer might use Visual Studio 2012 to incorporate a workflow into a SharePoint app.

- 
- Visio 2013
  - SharePoint Designer
  - Visual Studio 2012

### Visio 2013

Visio 2013 is a standard Microsoft Office application that many business analysts and information workers are familiar with. It includes tools that you can use to build workflows by using flowchart-type shapes and connectors. You can then import a Visio workflow into SharePoint Designer to publish the workflow to your SharePoint site.

### SharePoint Designer

SharePoint Designer 2013 includes a new visual designer that is available to you if Visio is installed on the local computer. The visual designer works in a way similar to Visio, and it enables you to drag and drop shapes to visually design your workflows and set their properties. SharePoint Designer also still provides the text-based designer that enables you to design your workflows in a declarative manner.

You can create workflows attached to lists, libraries, or sites and publish the workflow directly to that location. You can also package workflows and then deploy them to a remote server.

## Visual Studio 2012

Visual Studio 2012 is the Microsoft tool for software development. It includes a visual designer for SharePoint workflows that is similar to the SharePoint Designer, except that Visual Studio also enables you to write code to create custom workflow actions and tasks for SharePoint solutions or apps.

You deploy workflows by packaging the solution and deploying the package as a Feature to your SharePoint site. You can also create workflows as templates to enable reuse across any list or library.

## Discussion: Choosing a Suitable Development Approach

The instructor will now lead a discussion around the following scenario. Which approach to development would you use and why?

Contoso wants to automate the business process of authorizing payments to external contractors. The process is as follows:

1. The contractor submits their invoice.
2. The contractor's manager authorizes the payment.
3. An external payments company prints a remittance advice and action the payment.
4. The payment company exposes a web service for their printing services that you need to call from your workflow.

- Review the scenario in the student workbook
- Which approach to development would you use in each scenario and why?

## Lesson 2

# Building Workflows by using Visio 2013 and SharePoint Designer 2013

Workflows are often designed by information workers who are interested in the business logic of the process and have no need to understand the technology behind the workflow. Visio 2013 enables these users to create a workflow from a business point of view by using drag and drop techniques. If Visio is installed on a computer alongside SharePoint Designer, you can use the Visio tools in the Visual Designer in SharePoint Designer to similarly create a workflow by using drag and drop. You can import Visio workflows into SharePoint Designer to add parameters to the workflow, either in the Visual Designer or in the text editor.

If the user designing the workflow is familiar with SharePoint workflows and the SharePoint Designer, they can work directly in the SharePoint Designer text-based designer or Visual Designer without having to create and import a workflow from Visio. After a workflow is complete, you can configure it and then publish it to a SharePoint site.

### Lesson Objectives

At the end of this lesson, you will be able to:

- Create workflows by using Visio.
- Create and edit workflows by using SharePoint Designer.
- Publish workflows to a live site.
- Use workflows.
- Package and deploy SharePoint Designer workflows.

### Creating Workflows by Using Visio

Visio 2013 includes a Microsoft SharePoint 2013 Workflow template in the Flowchart category that you can use to design your workflows.

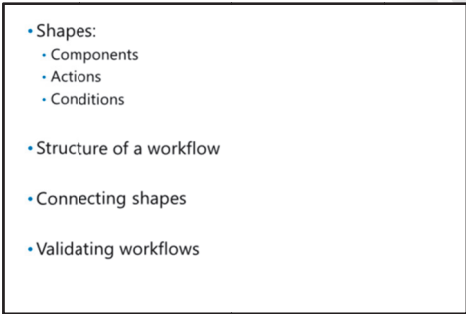
#### Visio workflow shapes

When you create a new Visio diagram based on the Workflow template, Visio creates a new workflow to which you can add components, actions, and conditions shapes that correspond to the constructs, actions, and conditions in a declarative SharePoint Designer workflow.

#### Components

The SharePoint 2013 Workflow components include:

- *Stage*. All actions in a workflow must be contained in a stage you can use to logically group a set of steps. The stage has an entry and exit point that can link to other stages in the workflow.
- *Step*. A stage can contain one or more steps that represent a series of actions in the workflow.
- *Loops*. SharePoint Designer 2013 introduces loops to workflows and they enable you to repeat one or more steps either for a set number of iterations or until a condition is true.

- 
- Shapes:
    - Components
    - Actions
    - Conditions
  - Structure of a workflow
  - Connecting shapes
  - Validating workflows

## Actions

SharePoint 2013 actions are the low-level building blocks of a workflow that determine how it behaves. They include actions to interact with a SharePoint item, to send email or assign tasks to users, to wait for an event or time duration, and to start a SharePoint 2010 workflow.



**Additional Reading:** For a complete list of the available actions, see *Action shapes* at <http://go.microsoft.com/fwlink/?LinkID=307101>.

## Conditions

The SharePoint 2013 conditions enable you to perform conditional logic in a workflow. There are conditions that you can use to determine if one value is equal to another, if a user is a member of a group, and if a SharePoint item was created or modified by a particular user.



**Additional Reading:** For a complete list of the available conditions, see *Condition shapes* at <http://go.microsoft.com/fwlink/?LinkID=307102>.

## Structure of a workflow

When you first create a workflow by using Visio, you are provided with a blank workflow that consists of one stage shape. The stage can contain other shapes and actions, such as assigning a task to a user or emailing a user. Each workflow must contain at least one stage, but you are likely to add many stages to each workflow. The only other shapes that can be added directly to the workflow, outside of a stage, are conditional shapes that you can use to route the workflow according to the results of an action in the previous stage.

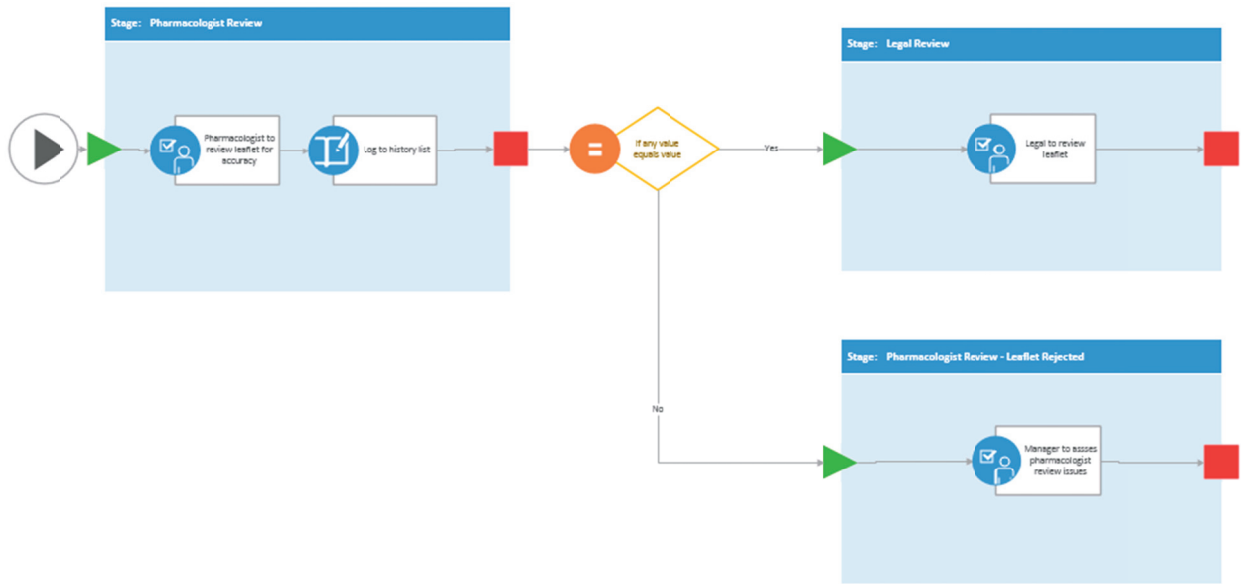
A stage can have only one route in and out of it, and they cannot be nested. You can add more than one step to a stage, though, to group your workflow actions into logical sets.

## Connecting shapes

You can use the **Connector Tool** on the **HOME** tab of the ribbon to connect shapes together to control the flow of the process. If the first shape in the pair that you are connecting has an outcome, you can right-click the connection and select **Yes** or **No** to define when the workflow should follow this route.



The following image shows a simple workflow using components, actions, and conditions in Visio.



### Validating workflows

You can use the **Check Diagram** tool on the **PROCESS** tab of the ribbon to validate the workflow against validation rules. This helps to ensure that your workflow logic is valid before you customize it in SharePoint Designer.

After you design, validate, and save your workflow in Visio, you can import it into SharePoint Designer for further customization.

### Creating and Editing Workflows in SharePoint Designer

When you open SharePoint Designer, the Navigation pane appears, which enables you to navigate your SharePoint environment. When you click the Workflows item in this pane, a list of existing workflows is displayed and you can use the **WORKFLOWS** tab on the ribbon to both create and edit list, reusable, and site workflows.

#### Creating workflows

To create a workflow, on the **WORKFLOWS** tab, in the **New** group, click the appropriate button for the type of workflow that you want to create. The information that you need to provide depends on which type of workflow you select.

- Creating workflows
- Using the Visual Designer
- Using the text-based designer
- Importing workflows

- *List workflows.* Before you create a list workflow, the list that you want to attach it to must exist. When you click **List Workflow**, you select the list, and then in the **Create List Workflow – MyList** dialog box, specify the name, and description for the workflow.
- *Reusable workflows.* When you create a reusable workflow, in the **Create Reusable Workflow** dialog box, you specify the name and description for the workflow.

MCT STUDENT USE PROHIBITED

- *Site workflows.* When you create a site workflow, in the **Create Site Workflow** dialog box, you specify the name and description for the workflow.

### Using the Visual Designer

If you have both Visio and SharePoint Designer installed on the same computer, you can use the Visual Designer in SharePoint Designer, which works similarly to the workflow template in Visio. When you first create the workflow, SharePoint Designer displays it in the text-based designer. You can switch to the Visual Designer by clicking the **View** button in the **Manage** group on the WORKFLOW ribbon.

The Visual Designer displays a Shapes pane that contains the Actions, Conditions, and Containers & Terminators categories, similar to Visio. You can drag these shapes on to the workflow surface to include them in a workflow. SharePoint Designer extends the functionality provided by Visio by enabling you to configure the shapes that you add to the workflow. When you select a shape, the SharePoint Designer **Properties** button appears in the lower-right corner of the shape and you can click items on this menu to configure the behavior of the shape. For example, for an **Assign a task** shape, you can configure the task participant, the task title, and the task description. The options available on this menu are dependent on the shape with which you are working.

### Using the text-based designer

The text-based designer also enables you to configure the shapes on a workflow, for example, to define to whom to assign a task. When you add a shape to the workflow, hyperlinks are created for any items that are configurable. When you click the link, the appropriate dialog box opens to enable you to configure the item.

For example, to configure a task, click **this user** to define the task participant and task title or to configure a transition to another stage, click **value**, and then select the appropriate stage.

### Importing workflows

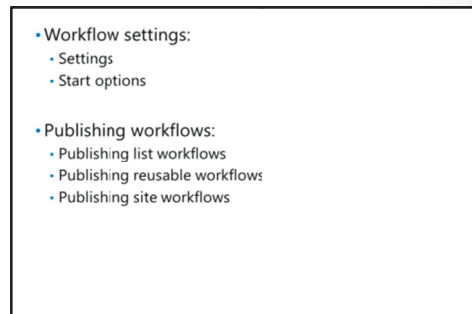
You use the **Import from Visio** button in SharePoint Designer to import a Visio workflow. This creates a new workflow in SharePoint Designer, and you need to provide a name, description, and type for the workflow. If you create a list workflow, you also need to specify the list to associate it with. By default, the workflow opens in the Visual Designer, which you can use in the same way as you use Visio to drag and drop shapes onto the workflow. You can also switch to the text-based designer if you prefer to work in that environment.

## Publishing Workflows to a Live Site

After you finish designing your workflow, you can publish it to your SharePoint site. Before publishing, though, it is good practice to save and validate your workflow. The **Check for Errors** button on the ribbon checks the current workflow for any errors that may prevent it from successfully completing. If errors are present, the text-based designer displays a message in red text alongside the relevant step. The **Save** button enables you to save a partially completed workflow without publishing it to a live site.

### Workflow settings

You can use the **Workflow Settings** page to configure how your workflow should behave.



## Settings

In the **Settings** section, you can configure the **Task List** and **History List** that the workflow will use when assigning tasks and logging messages.

By default, the workflow status automatically updates to the current stage name at each stage of the process. If you want to override this and set the status directly from your workflow (using the Set Workflow Status action), you can disable the default behavior in the **Workflow Settings**.

## Start options

You can use the start options to configure how users can start your workflow:

- *Manual start.* Use this setting to enable users to manually start the workflow from the SharePoint site.
- *Start on item creation.* Use this setting to enable the workflow to start when an item is created in the list or library.
- *Start on item change.* Use this setting to enable the workflow to start when an item changes in the list or library.



**Note:** The wording for these options varies, depending on the type of workflow that you are configuring. The three options are configurable for both list and reusable workflows, but only the manual start option is available for site workflows that cannot be automatically started.

## Publishing workflows

The method for publishing depends on the type of workflow that you have created.

### Publishing list workflows

When you create a list workflow, you associate it with a list; therefore, you can publish it directly to that list. The **Publish** button on the ribbon publishes the workflow to the associated list using the options specified in the workflow settings.

### Publishing reusable workflows

Before publishing a reusable workflow, you can use the **Associate to List** button on the ribbon to link the workflow to an existing list on your SharePoint site. When you click the button and select a list, the site opens in Internet Explorer on the **Settings > Add a Workflow** page, where you can specify the information required to publish your workflow in the chosen list.

You need to enter a name for the workflow, specify the task list and history list for the workflow to use, and the start options. That is, all the configurable options from the Workflow Settings page in SharePoint Designer. The ability to configure these options individually for each publication of a reusable workflow means tighter control over the behavior of different instances of your workflow.

If you create a reusable workflow at the top level of your site collection, you can use the **Publish Globally** button to make your workflow available to all sites in the site collection.

### Publishing site workflows

Site workflows are associated with a whole site, not an individual list or library; therefore, you can publish them directly from SharePoint Designer without having to configure any additional settings.

## Using Workflows

After you publish your workflow, you can use it from the relevant area of your SharePoint site. If it is configured to start on item creation or item change in a list or library, it will run automatically in response to the appropriate events. If it is only configured for manual start, you or your users will need to start it.

### Starting workflows automatically

When you configure a workflow that is associated with a list or library to start on item creation or item change, the workflow will start automatically when you create or change an item.

To review the status of a workflow, click the ellipsis next to an item name in the appropriate list or library, click the ellipsis on the menu, and then click **Workflows**. This displays the **Workflows** page for the particular list or library and shows all running and completed workflows. To view the details of a running or completed workflow, click the name of the workflow in the list. Here, you will find tasks created by the workflow and items added by logging to the workflow history. To update a task status, click the name of the task, and then on the ribbon, click **Edit Item** and change the settings as required.

### Starting workflows manually

When you configure a workflow to be started manually in a list or library, you can access the workflow by browsing to the workflow page for any item in that list or library. Here, you will find all workflows listed, both those configured to start automatically and manually.

To access site workflows, on the Quick Access menu, click **Site Contents**, and then click **SITE WORKFLOWS**. This displays the list of all site workflows and enables you to start a new workflow on the site. Click the name of a workflow to review and tasks or history associated with the workflow.



**Note:** If you log on to SharePoint using a system account, you will not be able to successfully run workflows and you will find they reach a suspended status during the first stage. To avoid this issue, make sure that you log on using a non-system account.

- Starting workflows automatically:
  - Create or change an item in the list or library
  - Browse to the workflows for any item in the list or library to review all in-scope workflows
  - Review the tasks or history for an individual workflow
- Starting workflows manually:
  - Browse to the workflows for a list or library
  - OR -
  - Browse to the SITE WORKFLOWS page
  - Start a new workflow or review running and completed workflows

## Demonstration: Creating Workflows in SharePoint Designer

In this demonstration, you will see how to create workflows in SharePoint Designer, how to publish workflows, and how to use workflows in SharePoint.

### Demonstration Steps

- Start the 20488B-LON-SP-11 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, type **SharePoint Designer 2013**, and then click **SharePoint Designer 2013**.
- In SharePoint Designer, click **Open Site**.
- In the **Open Site** dialog box, in the **Site name** box, type **http://team.contoso.com**, and then click **Open**.

- In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
- In the Navigation pane, click **Lists and Libraries**.
- On the ribbon, on the **LISTS AND LIBRARIES** tab, in the **New** group, click **Custom List**.
- In the **Create list or document library** dialog box, in the **Name** box, type **Demo List**, and then click **OK**.
- In the **Navigation** pane, click **Workflows**.
- On the ribbon, on the **WORKFLOWS** tab, in the **New** group, click **List Workflow**, and then click **Demo List**.
- In the **Create List Workflow - Demo List** dialog box, in the **Name** box, type **Demo Workflow**, and then click **OK**.
- Click (Start typing or use the Insert group on the ribbon.), type **Assign a task**, and then press Enter.
- Click **this user**.
- In the **Assign a Task** dialog box, in the **Participant** box, type **Dominik**, in the **Task Title** box, type **Demo task**, and then click **OK**.
- On the ribbon, on the **WORKFLOW** tab, in the **Manage** group, click **Views**, and then click **Visual Designer**.
- In the **Shapes** pane, click **Actions - SharePoint 2013 Workflow**, and then drag **Log to history list** and drop it to the right of the **Assign a task** action.
- Click the **Log to history** action that you just added, click the **SharePoint Designer Properties** button that appears in the lower-left of the action, and then click **Message**.
- In the **Log to History List Properties** dialog box, type **Task assigned**, and then click **OK**.
- On the ribbon, on the **WORKFLOW** tab, in the **Manage** group, click **Views**, and then click **Text-Based Designer**.
- Review the log action that has been added.
- On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Check for Errors**.
- In the **Microsoft SharePoint Designer** dialog box, click **OK**.
- On the ribbon, on the **WORKFLOW** tab, in the **Manage** group, click **Workflow Settings**.
- In the **Start Options** group, select the **Start workflow automatically when an item is created** check box.
- On the ribbon, on the **WORKFLOW SETTINGS** tab, in the **Save** group, click **Save**.
- On the ribbon, on the **WORKFLOW SETTINGS** tab, in the **Save** group, click **Publish**.
- On the Start screen, click **Internet Explorer**.
- In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
- In SharePoint, on the Quick Launch menu, click **Demo List**.
- Click **new item**, in the **Title** box, type **Demo Item**, and then click **Save**.
- Next to **Demo Item** in the list, click the ellipsis, and then click **Workflows**.
- Note that the Demo workflow has started.

- Under **Running Workflows**, click **Demo Workflow** and note that a task has been assigned to Dominik Dubicki.
- Under **Tasks**, click **Demo task**. If there are no items under **Tasks**, refresh the page.
- On the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
- Click **Approved** and when the Demo Workflow page loads, click **Refresh**.
- Note that the workflow is now complete and the “Task assigned” message has been logged to the **Workflow History** section.
- Close Internet Explorer.
- Close SharePoint Designer.

## Packaging and Deploying SharePoint Designer Workflows

Frequently, you will want to deploy your workflows to another SharePoint 2013 environment. For example, you may be developing your workflows in a test environment and want to move them to the production environment, or you may want to reuse your workflow functionality elsewhere in your organization. SharePoint Designer 2013 includes packaging features that enable you to copy workflows between SharePoint environments.

- **Packaging a workflow:**
  - Use the Save a Template feature in SharePoint Designer
  - Creates a WSP file in the Site Assets library
- **Exporting workflows:**
  - Use the Export File feature on the ASSETS tab of the ribbon
  - Creates a WSP file on the local computer
- **Deploying and activating workflows:**
  - Use the Upload Solution feature in the SharePoint site
  - Activate the solution, and then activate the site feature

### Packaging workflows

To package a workflow, you save the workflow as a template by using the **Save as Template** button in the **Manage** group on the ribbon in SharePoint Designer. When you save a workflow as a template, SharePoint Designer creates a Web solution package (WSP) file in the Site Assets library of your SharePoint site.

### Exporting workflows

After you create the WSP, you need to export it from the site on to the file system. To export the workflow, in SharePoint Designer, in the **Navigation** pane, click **Assets**, select a WSP, and then click **Export File**. You then specify where to save the exported file, either on the local computer or on a shared network drive.

### Deploying and activating workflows

After you export the workflow, you can copy it to your planned destination location, and then deploy it to a new site collection. To deploy the workflow, on the **Site Settings** page, under **Web Designer Galleries**, click **Solutions**. Then use the **Upload Solution** button on the ribbon to upload the solution to the site. If you are logged on as an administrator, in the **Activate Solution** dialog box, click **Activate**. If you are logged on as a non-administrator user, you must log out, log on as an administrator, select the correct solution, and then on the ribbon, click **Activate**.



**Note:** You must be logged on to the SharePoint site as an administrator to activate a workflow. If you are not, the **Activate** button will be disabled. It is easiest to log on as an administrator before deploying the workflow so that SharePoint automatically displays the **Activate Solution** dialog box.

The final step in the process is to activate the feature within the solution. On the **Site Settings** page, under **Site Actions**, click **Manage site features**, select your new feature, and then click **Activate**.

After you activate the feature, you will be able to use the workflow exactly as you did on the development site. However, if your workflow uses custom lists or libraries, the relevant list or library must also exist on the target site. If it does not, the workflow will fail.



**Note:** Remember to log on using a non-system account before trying to run your workflow.

# Lab A: Building Workflows in Visio 2013 and SharePoint Designer 2013

## Scenario

Contoso produces information leaflets for all of their pharmaceutical products. The publishing process for these information leaflets consists of several steps. The first draft is created by a pharmacologist (Paul West) and then submitted to a copy editor (Danny Levin). If changes are required, they must be incorporated by the original author (Paul West). If there are no copy edit changes, the leaflet can be published by Dominik Dubicki. After the information leaflet is published, it must be reviewed for accuracy at least once every 12 months. Your task is to develop a workflow to automate this process.

## Objectives

After completing this lab, you will be able to:

- Create workflows by using Visio.
- Edit and publish workflows by using SharePoint Designer.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-11
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

## Exercise 1: Creating Workflows by Using Visio

### Scenario

In this exercise, you will create the first draft of the workflow by using Visio. You will create a blank Visio workflow, add tasks and connectors to implement the logic, and then validate and save the workflow on the local computer.

The main tasks for this exercise are as follows:

1. Create a New Workflow in Visio 2013
2. Implement Workflow Logic
3. Validate and Save the Workflow

#### ► Task 1: Create a New Workflow in Visio 2013

- Start the 20488B-LON-SP-11 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visio and create a new file from the **Microsoft SharePoint 2013 Workflow** template and using **US Units**.

#### ► Task 2: Implement Workflow Logic

- Change the name of Stage 1 to **Write leaflet**.
- Add an Assign a task action to the Write Leaflet stage and name it Pharmacologist to write leaflet.
- Add another stage to the workflow and name it **Copy Edit**.
- Connect the output of the **Write Leaflet** task to the input of the **Copy Edit** task.
- Add an Assign a task action to the Copy Edit stage and name it Editor to review and log any changes required.



- Add an **If any value equals value** condition to the right of the **Copy Edit** stage and name it **Ready to publish?**
  - Connect the output of the **Copy Edit** stage to the **Ready to publish?** condition.
  - Add another stage to the workflow and name it **Publish**.
  - Connect the output of the condition to the input of the **Publish** stage and set the path to **Yes**.
  - Connect the output of the condition to the input of the **Write Leaflet** stage and set the path to **No**.
  - Add an Assign a task action to the Publish stage and name it Manager to publish leaflet.
  - Add another stage to the workflow and name it **Annual Review**.
  - Connect the output of the **Publish** stage to the input of the **Annual Review** stage.
  - Add an **Add time to date** action to the **Annual Review** stage.
  - Add an Assign a task action to the Annual Review stage and name it Manager to check that the leaflet is still current.
- **Task 3: Validate and Save the Workflow**
- Check the diagram against the current validation rules.
  - Save the diagram as **Leaflet** in the **E:\Labfiles\Starter** folder.
  - Close Visio.

**Results:** After completing this exercise, you should have implemented workflow logic by using Visio.

## Exercise 2: Editing Workflows by Using SharePoint Designer

### Scenario

In this exercise, you will import the workflow that you created into SharePoint Designer, edit the workflow to add detail, and then deploy the workflow. Finally, you will test the workflow to verify that it performs as expected.

The main tasks for this exercise are as follows:

1. Import a Workflow into SharePoint Designer
2. Update the Workflow by Using the Text-Based Designer
3. Publish and Test the Workflow

► **Task 1: Import a Workflow into SharePoint Designer**

- Open SharePoint Designer and connect to the **http://team.contoso.com** site as **Dominik** with the password **Pa\$\$w0rd**.
- Import the Visio 2013 **Leaflet** workflow from the E:\Labfiles\Starter folder using the following settings:
  - a. Name: Leaflet Workflow
  - b. Description: Workflow to guide product leaflets through the edit and publish processes
  - c. Workflow Type: Reusable Workflow
- Review the imported workflow.

**► Task 2: Update the Workflow by Using the Text-Based Designer**

- Switch to the Text-Based Designer view.
- Review the existing stages.
- Edit the **Write Leaflet** stage as follows:
  - Assign the task to **Paul** and title the task **Write product leaflet**.
- Edit the **Copy Edit** stage as follows:
  - a. Assign the task to **Danny** and title the task **Full copy edit of product leaflet**.
  - b. In the **If** condition, set the test to check if the outcome of the task is **Approved**.
- In the **Publish** stage, assign the task to **Dominik** and title the task **Leaflet ready for publishing**.
- In the **Annual Review** stage, change the step to 12 months from today's date, and then add a step to pause until that time.
- Assign the task to **Dominik** and title the task **Annual review due**.

**► Task 3: Publish and Test the Workflow**

- Check the workflow for errors.
- Save the workflow.
- Switch to the **Workflow Settings** view, and then associate the workflow with the **Product Leaflets** library.
- Log on to SharePoint as **Paul** with the password **Pa\$\$w0rd**.
- Name the workflow **Leaflet Workflow** and ensure that the workflow will start when a new item is added.
- Add **ProductA.docx** from the **E:\Labfiles\Starter** folder to the **Product Leaflets** library.
- Review the workflow tasks, and then edit the **Write product leaflet** task assigned to Paul to have a status of **approved**.
- Sign out of SharePoint and close the Internet Explorer window.
- Open Internet Explorer and log on to SharePoint as **Danny** with the password **Pa\$\$w0rd**.
- Review the workflow tasks and then edit the **Full copy edit of product leaflet** task assigned to Danny to have a status of **approved**.
- Sign out of SharePoint and close the Internet Explorer window.
- Open Internet Explorer and log on to SharePoint as **Dominik** with the password **Pa\$\$w0rd**.
- Review the workflow tasks and then edit the **Leaflet ready for publishing** task assigned to Dominik to have a status of **approved**. This will pause the workflow for 12 months until the annual review is due.
- Sign out of SharePoint and close the Internet Explorer window.
- Open Internet Explorer and log on to SharePoint as **Paul** with the password **Pa\$\$w0rd**.
- Add **ProductB.docx** from the **E:\Labfiles\Starter** folder to the **Product Leaflets** library.
- Review the workflow tasks, and then edit the **Write product leaflet** task assigned to Paul to have a status of **approved**.
- Sign out of SharePoint and close the Internet Explorer window.

- Open Internet Explorer and log on to SharePoint as **Danny** with the password **Pa\$\$w0rd**.
- Review the workflow tasks and then edit the **Full copy edit of product leaflet** task assigned to Danny to have a status of **rejected**.
- Sign out of SharePoint and close the Internet Explorer window.
- Open Internet Explorer and log on to SharePoint as **Paul** with the password **Pa\$\$w0rd**.
- Review the workflow tasks and verify that because Danny rejected the leaflet, the workflow has returned to the **Write leaflet** task assigned to Paul.
- Sign out of SharePoint and close the Internet Explorer window.
- Close SharePoint Designer.

**Results:** After completing this exercise, you should have created and published the Leaflet workflow by using SharePoint Designer.

## Lesson 3

# Developing Workflows in Visual Studio 2012

You may need to develop advanced workflows that include complex logic, such as calling web services or other libraries stored in your organization. You can use Visual Studio to shield other users from the complexities that this involves and present a simpler workflow action for them to use in SharePoint Designer.

You can also develop Visual Studio workflows as part of a SharePoint solution or app for SharePoint.

### Lesson Objectives

At the end of this lesson, you will be able to:

- Describe Visual Studio workflows.
- Add a workflow custom activity to a SharePoint project.
- Create workflow logic.
- Create an .actions4 file.
- Deploy and publish a workflow.

### Introduction to Visual Studio Workflows

Visual Studio workflows, like those in SharePoint Designer, are entirely declarative. This is a change from SharePoint 2010 where Visual Studio workflows could include custom code. Visual Studio workflows for SharePoint 2013 are built on Windows Workflow Foundation 4, so just like SharePoint Designer workflows, they run in Workflow Manager Client 1.0.

You use the visual designer in Visual Studio 2012 to define the structure and order of your workflow, customizing actions by defining their properties. After your workflow is complete, you package and deploy it as a SharePoint Feature to enable users to reuse the workflow wherever it is appropriate.

#### Workflow templates

There are two types of workflow project that you can create by using Visual Studio: a workflow and a workflow custom activity.

##### **Workflow template**

The Workflow template enables you to create workflows for use in your app or solution. The workflow that you create will be deployed as part of your project; therefore, you can create list or site workflows, but not reusable workflows. When creating a list workflow, you can associate the workflow with a list either when you create the workflow or at a later stage. You can also configure how the workflow should start: manually, when an item is created, or when an item changes.

##### **Workflow Custom Activity template**

The Workflow Custom Activity template enables you to create workflows that you can deploy to SharePoint for use in SharePoint Designer. You can use these to hide workflow complexity from your users

- Declarative workflows
- No custom code
- Built on Windows Workflow Foundation 4
- Packaged as Features
- Workflow templates:
  - Workflow
  - Workflow Custom Activity

and to simplify the workflow development process for them. You can enable your custom activity to accept parameters and to return data to your SharePoint Designer workflow, and the Visual Studio templates enable you to create links to configure this data that match those of the standard SharePoint actions. After you deploy a workflow custom activity to SharePoint, it will appear in the **Custom** group on the **Action** menu in SharePoint Designer. You can think of custom workflow activities as reusable building blocks that you can create for yourself or other users to include in other workflows.

## Adding a Workflow Custom Activity to a Project

You can add a workflow custom activity to any type of SharePoint 2013 project. In the **Add New Item** dialog box, in the **Office/SharePoint** template group, select **Workflow Custom Activity** to add the template to your project. Adding the workflow custom activity adds the following files to the project.

- *Feature1.feature*. The feature file specifies the scope of the Feature, for example, Web, Farm, or Site, and the location of the assemblies and files that comprise the Feature.
- *<Workflow Custom Activity name>.xaml*. The .xaml file defines the steps in your workflow. You can use the visual designer to create the workflow, but the underlying definition exists in XAML markup in this file.
- *<Workflow Custom Activity name>.action4*. The action4 file defines the conditions and actions that will display in SharePoint Designer. It also binds the fields that the user can configure in SharePoint Designer to variables used in the workflow custom activity.
- *Elements.xml*. The .xml file defines the properties for the custom action for example, whether the files can be cached at the web front end.

- Workflow Custom Activity template comprises:
  - Feature1.feature
  - *<Workflow Custom Activity name>.xaml*
  - *<Workflow Custom Activity name>.action4*
  - Elements.xml

When you create the project, you also need to provide the name of the site to use for debugging. This enables you to develop custom activity workflows on your local computer, but use a remote SharePoint site for debugging purposes.

## Creating the Workflow Logic

You create your workflow in the Activity1.xaml file by using the visual designer. When you first add the template to your project, you are presented with an empty **Sequence** control to which you can add objects from the workflow toolbox. A sequential workflow contains a series of steps that execute in the order that they are defined. You can also create state machine workflows by deleting the **Sequence** control and adding a **StateMachine** control. The steps in a state machine workflow are triggered by actions and states, and can execute asynchronously.

- Using variables, arguments, and imports:
  - Variables – store data during the workflow
  - Arguments – pass data in and out of the workflow
  - Imports – import namespaces into the workflow
- Using workflow controls:
  - Add to workflow by using the Toolbox
  - Configure by using the Properties pane

## Using variables, arguments, and imports

You can define variables, arguments, and imports for your workflow by using the pane below the visual designer.

### **Variables**

You define variables to store data during the processing of your workflow. For example, you may define a variable to store information about the user that is currently logged on to SharePoint or about the item that triggered the workflow. When you define a variable, you must specify its name, type, scope, and optionally a default value.

Workflow Manager 1.0 introduces a new variable type, named **DynamicValue**. This type enables you to store data structures in your workflows and access the data in them by using XPATH notation to navigate any hierarchy. For example, you could store Orders and OrderDetails in a **DynamicValue** variable. To access the Cost field in the first OrderDetails row, you would use the **d/Orders(0)/OrderDetails(0)/Cost** path.

### **Arguments**

You can define both in and out arguments for your workflow, to take data from the caller of the workflow and to return information to it. For example, your workflow may take an in argument of the name of a pharmaceutical product, look up additional information about this product by using a web service, and then return that information by using an out argument. When you define an argument, you must specify its name, direction, type, and optionally a default value.

### **Imports**

You can use imports to import namespaces into your workflow and use the functionality contained in them. By default, workflow imports the **System**, **System.Collections.Generic**, **System.Data**, and **System.Text** namespaces. You can import any workspace to your workflow by adding a reference to it and then adding it to the imported list.

## Using workflow controls

The workflow toolbox includes all of the controls that you can add to your workflow. The controls are categorized by their functionality. For example, the **Control Flow** group contains controls such as **DoWhile**, **If**, and **Switch<T>**, which you can use to control the flow of the workflow and the **SP – List** group contains controls such as **CheckInItem**, **CreateListItem**, and **LookupSPList**, which you can use to interact with lists and their items. Another key control is the **HttpSend** control, which enables you to call a method exposed by a web service.

When you add a control to the workflow surface, all of its configurable properties are listed in the **Properties** pane. For example, if you add a **CheckInItem** control to the surface, the properties that you can set are **Comment**, to define the comment for the check in, **ItemId**, to identify the item, and **ListId**, to identify the list or library for checking in. You may be storing the text to use for the comment in a variable, or it may have been passed to the workflow as an in argument; in which case, you use the variable or argument name as the property value. Similarly, when using the **HttpSend** control, you specify the HTTP method in the **Method** property, the URL of the web service in the **Uri** property, the headers in the **RequestHeaders** property, and a variable in which to store the return value in the **ResponseContent** property.

If you are working with **DynamicValue** variable, you can use the **GetDynamicValueProperty<T>** control to retrieve data from the variable. Use the **Source** property to define the name of the variable to work with, the **PropertyName** property to identify the data required by using the XPATH notation, and the **Result** property to define the name of the variable to store the result in.

## Creating the actions4 File

You use the .actions4 file to define the sentence that SharePoint Designer displays when you add the custom action to a workflow and to bind the settings from there to the arguments in your workflow.

### RuleDesigner element

When you create the workflow custom activity, Visual Studio provides a template .actions4 file containing an empty **RuleDesigner** element. You add **FieldBind** elements to the **RuleDesigner** element to define the arguments that SharePoint Designer will display. The **FieldBind** element includes the following attributes:

- *Field*. Use this text attribute to define the arguments to bind to in the workflow.
- *Text*. Use this text attribute to define the text to display to the user in SharePoint Designer.
- *Id*. Use this integer attribute to create a unique identifier for the element.
- *DesignerType*. Use this text attribute to define the type of control that is presented to the user in SharePoint Designer, for example, a text box or a list of users.



**Additional Reading:** For more information about the attributes of the **FieldBind** element, see *FieldBind Element (WorkflowInfo)* at <http://go.microsoft.com/fwlink/?LinkID=307103>.

The **RuleDesigner** element contains one attribute, the **Sentence** attribute, which you use to define the sentence displayed in SharePoint Designer. You can embed arguments in the sentence by using the **Id** attribute from their **FieldBind** element. SharePoint Designer then uses the **Text** attribute from the **FieldBind** element to display to the user and the **DesignerType** attribute to define the control to use for input.

The following code example shows a sample **RuleDesigner** element with two **FieldBind** child elements.

### RuleDesigner Element

```
<RuleDesigner Sentence="Get info for user %1 (output to %2)">
 <FieldBind Field="UserName" Text="User name" Id="1" DesignerType="Person" />
 <FieldBind Field="Info" Text="UserInfo" Id="2" DesignerType="TextArea" />
</RuleDesigner>
```

### Parameters element

The second top level element in the .actions4 file is the **Parameters** element, which contains one or more child **Parameter** elements. You use the **Parameter** elements to describe the input and output parameters for the workflow. The Parameter element includes the following attributes:

- *Name*. Use this text attribute to associate the **FieldBind** element with the parameter.
- *Type*. Use this string attribute to define the data type for the parameter as a partially qualified .NET Framework type.
- *Direction*. Use this text attribute to define whether the parameter is an in, out, or optional parameter.

```
<Action Name="Sample" ClassName="MyProject.Sample">
 <RuleDesigner Sentence="Get info for user %1 (output to %2)">
 <FieldBind Field="UserName" Text="User name" Id="1"
 DesignerType="Person" />
 <FieldBind Field="Info" Text="UserInfo" Id="2"
 DesignerType="TextArea" />
 </RuleDesigner>
 <Parameters>
 <Parameter Name="UserName" Type="System.String, mscorlib"
 Direction="In" DesignerType="Person" />
 <Parameter Name="Info" Type="System.String, mscorlib"
 Direction="Out" DesignerType="TextArea" />
 </Parameters>
</Action>
```

- *DesignerType*. Use this text attribute to define the type of UI element to use to edit the contents of the parameter.



**Additional Reading:** For more information about the attributes of the **Parameter** element, see *Parameter Element (WorkflowInfo)* at <http://go.microsoft.com/fwlink/?LinkID=307104>.

The following code example shows a sample **Parameters** element with two child **Parameter** elements that are associated with the **FieldBind** elements in the previous example.

### Parameters Element

```
<Parameters>
 <Parameter Name="UserName" Type="System.String, mscorlib" Direction="In"
 DesignerType="Person" />
 <Parameter Name="Info" Type="System.String, mscorlib" Direction="Out" DesignerType="TextArea"
 />
</Parameters>
```

## Deploying and Publishing a Workflow Activity

After you finish developing your workflow activity, you can deploy it to your debugging SharePoint site for testing and then publish it to your production site.

### Deploying a workflow activity

Use the **Deploy Solution** option on the **BUILD** menu in Visual Studio to publish the workflow activity to the site that you defined when creating your project.

After you publish the project, you need to clear the SharePoint Designer cache so that it can download the new workflow definitions. To do this, you need to delete the files in the following folders:

- %APPDATA%\Microsoft\Web Server Extensions\Cache
- %USERPROFILE%\AppData\Local\Microsoft\WebsiteCache

Next time you open SharePoint Designer, it will download your workflow definition, which will then be available in the **Custom** group in the **Actions** list.

### Publishing a workflow

After you test your workflow activity on your debugging site, you may want to publish it to a production site. Use the **Publish** option on the **BUILD** menu in Visual Studio to publish your workflow activity to an alternative site. This action displays the **Publish** dialog box, where you can specify the site URL where you want the workflow publishing. After you publish the workflow activity, the SharePoint site will open and you can activate the solution on the site.

If you are not currently connected to the production site, you can publish the workflow to the file system. This generates a Web Solution Package (.wsp file) that you can copy to your production server and then upload and activate on your SharePoint site.

- Deploying a workflow activity:
  - Deploy to the site defined when you created the project
  - Clear the SharePoint Designer cache
  - Use the workflow activity from the Custom group in the Actions list
- Publishing a workflow activity:
  - Publish to an alternative site:
    - Clear the SharePoint Designer cache
    - Use the workflow activity
  - Publish to the file system:
    - Copy the .wsp file to the target server
    - Upload the solution
    - Activate the solution



## Lab B: Creating Workflow Actions in Visual Studio 2012

### Scenario

Your colleague, Dominik Dubicki, who is responsible for publishing the Contoso product information leaflets mentions to you that whenever he publishes a leaflet he has to use another application to discover the distribution area for the leaflet. He asks if you can use this application in your workflow to include the information in his task title.

You decide to develop a workflow custom activity in Visual Studio to query the distribution web service and output the distribution area. You will then incorporate this action into the existing workflow and add the distribution area to the task assigned to Dominik.

### Objectives

After completing this lab, you will be able to:

- Create a custom workflow action.
- Deploy and use a custom workflow action.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-11
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

### Exercise 1: Creating Custom Workflow Actions

#### Scenario

In this exercise, you will use Visual Studio to create and deploy the workflow custom activity. You will implement the workflow logic, create the .actions4 file, and then deploy the workflow to the team site.

The main tasks for this exercise are as follows:

1. Create a Custom Workflow Action Project
2. Implement Workflow Logic
3. Create the actions4 File
4. Deploy the Workflow

#### ► Task 1: Create a Custom Workflow Action Project

- Start the 20488B-LON-SP-11 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio.
- Create a new SharePoint empty project named **ProductDistributionProject** in the E:\Labfiles\Starter folder using the http://team.contoso.com site for debugging and deploying as a sandboxed solution.
- Add a new Workflow Custom Activity item named **Distribution** to the project.

#### ► Task 2: Implement Workflow Logic

- Create two new variables with Sequence scope:
  - a. **WebUri** as type **String**
  - b. **WebResponse** as type **DynamicValue**

- Create two new arguments:
  - a. **ProdName** as an **In** argument
  - b. **ProdDist** as an **Out** argument
- Add an **Assign** activity to the Sequence and use it to assign the following text to the **WebUri** variable:

```
"http://localhost:32999/ProductDistribution/Products.svc/ProductDists('" + ProdName +
"')/Distribution"
```

- Add an **HttpSend** activity under the Assign activity, setting properties as follows:
  - a. Method: **GET**
  - b. RequestHeaders: **Accept = "application/json; odata=verbose"**
  - c. Content-Type = **"application/json; odata=verbose"**
  - d. Uri: **WebUri**
  - e. ResponseContent: **WebResponse**
- Add a **GetDynamicValueProperty<T>** activity under the HttpSend activity, using a string type.
- Set properties as follows:
  - a. PropertyName: **"d/Distribution"**
  - b. Result: **ProdDist**
  - c. Source: **WebResponse**
- Save the solution.

### ► Task 3: Create the actions4 File

- Open the Distribution.actions4 file.
- Modify the **Sentence** attribute of the **RuleDesigner** to the following:

```
Get distribution area for %1 (output to %2)
```

- Add a **FieldBind** element to the **RuleDesigner** element, using the following information:
  - a. Field: **ProdName**
  - b. Text: **Product name**
  - c. Id: **1**
  - d. DesignerType: **TextArea**
- Add another **FieldBind** element to the **RuleDesigner** element, using the following information:
  - a. Field: **ProdDist**
  - b. Text: **Product distribution**
  - c. Id: **2**
  - d. DesignerType: **ParameterNames**
- Add a **Parameter** element to the body of the **Action** element, using the following information:
  - a. Name: **ProdName**
  - b. Type: **System.String, mscorlib**

- c. Direction: **In**
- d. DesignerType: **TextArea**
- e. Description: **The name of the product**
- Add another **Parameter** element to the body of the **Action** element, using the following information:
  - a. Name: **ProdDist**
  - b. Type: **System.String, mscorlib**
  - c. Direction: **Out**
  - d. DesignerType: **ParameterNames**
  - e. Description: **The distribution for the product**
- Save all files.
- ▶ **Task 4: Deploy the Workflow**
  - Build the solution.
  - Deploy the solution.
  - Open Internet Explorer and connect to the team site as **Administrator** with the password **Pa\$\$w0rd**.
  - On the **Manage site features** page, verify that the **ProductDistributionProject** has been successfully deployed.
  - Close Internet Explorer and then close Visual Studio.

**Results:** After completing this exercise, you should have created and deployed a workflow custom activity.

## Exercise 2: Using a Custom Workflow in SharePoint Designer

### Scenario

In this exercise, you will use the custom action in SharePoint Designer and then test to confirm that it functions as expected.

The main tasks for this exercise are as follows:

1. Add the Custom Workflow to the SharePoint Designer Workflow
2. Test the Workflow

### ▶ Task 1: Add the Custom Workflow to the SharePoint Designer Workflow



**Note:** Before opening SharePoint Designer, you need to clear the cache folders so that it can download the new workflow content from the server.

- Using File Explorer, delete the contents of the following two folders:
  - a. %APPDATA%\Microsoft\Web Server Extensions\Cache
  - b. %USERPROFILE%\AppData\Local\Microsoft\WebsiteCache
- Open SharePoint Designer and connect to the team site as **Dominik** with the password **Pa\$\$w0rd**.
- Edit the Leaflet Workflow.

- Add a stage named **Get product distribution**.
  - Add a **Distribution** action to this stage.
  - Set the **ProductName** for the action to **Title**.
  - Set the transition for the stage to go to the **Publish** stage.
  - In the **Publish** stage, modify the task title to include the contents of the **Product distribution** variable.
  - In the **Copy Edit** stage, modify the approved transition to go to the **Get Product Distribution** stage.
  - Check the workflow for errors.
  - Save the workflow.
  - Publish the workflow.
  - Close SharePoint Designer.
- **Task 2: Test the Workflow**
- Open Internet Explorer, and then log on to SharePoint as **Paul** with the password **Pa\$\$w0rd**.
  - Add **ProductC.docx** from the **E:\Labfiles\Starter** folder to the **Product Leaflets** library.
  - Review the workflow tasks, and then edit the **Write product leaflet** task for **ProductC.docx** which is assigned to Paul to have a status of **approved**.
  - Sign out of SharePoint and close the Internet Explorer window.
  - Open Internet Explorer, log on to SharePoint as **Danny** with the password **Pa\$\$w0rd**.
  - Review the workflow tasks, and then edit the **Full copy edit of product leaflet** task assigned to Danny to have a status of **approved**.
  - Sign out of SharePoint and close the Internet Explorer window.
  - Open Internet Explorer, log on to SharePoint as **Dominik** with the password **Pa\$\$w0rd**.
  - Review the workflow tasks and verify that the task assigned to Dominik now includes where to publish the workflow.
  - Sign out of SharePoint and close the Internet Explorer window.

**Results:** After completing this exercise, you should have used and tested a custom workflow action.

## Module Review and Takeaways

In this module, you learned how to use SharePoint workflows to automate business processes. You learned how SharePoint workflows are built on Workflow Manager 1.0 and execute outside of SharePoint to enhance scalability. You learned how to develop workflows in Visio, SharePoint Designer, and Visual Studio and how to deploy and publish them to your SharePoint sites.

### Review Question(s)

#### Test Your Knowledge

Question	
Which type of workflow cannot be triggered by an event?	
Select the correct answer.	
<input type="checkbox"/>	List workflow
<input type="checkbox"/>	Site workflow
<input type="checkbox"/>	Reusable workflow

**Question:** You are developing a workflow in Visio and have added an **Assign a task** shape to the workflow. However, you cannot locate the **Properties** button to customize the task. What do you need to do?

#### Test Your Knowledge

Question	
Which of the following controls can you use to call a method in a web service from a Visual Studio workflow custom activity?	
Select the correct answer.	
<input type="checkbox"/>	Interop
<input type="checkbox"/>	WorkflowInterop
<input type="checkbox"/>	HttpSend
<input type="checkbox"/>	WaitForCustomEvent
<input type="checkbox"/>	FlowSwitch<T>



# Module 12

## Managing Taxonomy

### Contents:

Module Overview	12-1
Lesson 1: Managing Taxonomy in SharePoint 2013	12-2
Lesson 2: Working with Content Types	12-10
Lab A: Working with Content Types	12-20
Lesson 3: Working with Advanced Features of Content Types	12-24
Lab B: Working with Advanced Features of Content Types	12-30
Module Review and Takeaways	12-37

## Module Overview

A taxonomy is a system of classification. In information management systems such as SharePoint, the taxonomy specifies how you organize and classify the documents and content that you add to SharePoint sites and lists. A good taxonomy should make it easy for business users to find, retrieve, and manage content. In this module, you will learn how to work with the key building blocks that SharePoint provides for implementing taxonomy.

### Objectives

After completing this module, you will be able to:

- Work with taxonomy building blocks in SharePoint 2013.
- Create and configure content types declaratively and programmatically.
- Work with advanced features of content types.

## Lesson 1

# Managing Taxonomy in SharePoint 2013

Fields provide the foundations for all taxonomies in SharePoint 2013. Fields enable users to categorize, sort, and filter information. They enable you to define managed properties and scopes for search, and to create relationships between data in different repositories. In this lesson, you will learn how to implement and manage fields programmatically in a SharePoint 2013 deployment.

## Lesson Objectives

After completing this lesson, you will be able to:

- Explain the building blocks that you can use to design and implement a taxonomy in SharePoint 2013.
- Define site columns declaratively by using Collaborative Application Markup Language (CAML).
- Create site columns programmatically by using server-side or client-side code.
- Retrieve and edit site columns programmatically.
- Define list relationships by creating lookup fields.

## Understanding Taxonomy in SharePoint

In an information management system such as SharePoint, a robust taxonomy is essential. It enables content creators to organize and categorize their data, and it enables content consumers to locate data quickly through search or through navigation.

In SharePoint, taxonomy is closely tied to metadata, which you can define as data about data. For example, in the case of a Microsoft Word document, the data is the contents of the document, whereas the metadata is information about the document such as the author, the date it was created, and so on. Metadata enables users to find data by filtering, searching, or navigating.

SharePoint provides three core building blocks that solution architects can use to design and implement a taxonomy:

- *Site columns.* To collect additional metadata about list items or files in SharePoint, you add a column to the list or library. Site columns are reusable columns that you can add to lists or libraries within the site. SharePoint includes many built-in site columns that you can choose from, and you can create your own site columns if none of the built-in columns meet your requirements.
- *Content types.* A content type defines and classifies a document or a list item according to the requirements of the organization. Content types can specify the site columns, workflows, policies, and document templates that should be associated with a particular item. For example, you might define a content type named *Invoice*. If a user uploads a document and specifies a content type of *Invoice*, SharePoint will prompt the user for any required field values, such as the invoice amount, the supplier, and the due date.

- A taxonomy is a system of classification
- In SharePoint, taxonomy is closely associated with metadata
- Building blocks:
  - Site columns
  - Content types
  - Term sets



- *Managed metadata term sets.* A term set is a hierarchical set of terms that is typically maintained by one or more term set administrators. Term sets provide a way of ensuring that users use a consistent nomenclature when adding metadata. You can create site columns based on a term set, so that users can only add terms from the term set when they populate certain fields. You can also use term sets to drive navigation on publishing sites.

## Creating Site Columns Declaratively

A site column is effectively a template for a column in a list or a library. Site columns are useful for two main reasons:

- *Reusability.* In many cases, you will want to add the same column to multiple lists or libraries. For example, a date-based column named *Expiry Date* could be useful in many different scenarios. Instead of manually creating the same column in multiple lists and libraries, users can simply add the column from the site column collection.
- *Consistency.* If you are capturing the same information in multiple lists or libraries, you want that information to be captured in a consistent way. For example, if you use a choice field to represent distinct stages of a production process, you want those choices to be described consistently in different lists and libraries.

### Creating site columns

To create a site column declaratively, you define a **Field** element within an element manifest.

The following code example shows an element manifest that defines several site columns of different types:

#### Creating Site Columns

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

 <Field ID="{40dca724-d8f9-4983-856b-f8ca48bd6895}"
 Name="Ingredients"
 DisplayName="Ingredients"
 Type="Note"
 NumLines="5"
 Required="TRUE"
 Group="Contoso Columns">
 </Field>

 <Field ID="{f02f0407-ed85-463d-bca6-61ba79b5f74e}"
 Name="LeadChemist"
 DisplayName="Lead Chemist"
 Type="User"
 UserSelectionMode="0"
 Required="TRUE"
 Group="Contoso Columns">
 </Field>

 <Field ID="{60616688-e1ff-49b6-9ede-c4674e7a59fa}"
 Name="ExpiryDate"
 DisplayName="Expiry Date"
 Type="DateTime"
 Format="DateOnly"
 </Field>
</Elements>
```

- Define a Field element within an element manifest

```
<Field ID="{f02f0407-ed85-463d-bca6-61ba79b5f74e}"
 Name="LeadChemist"
 DisplayName="Lead Chemist"
 Type="User"
 UserSelectionMode="0"
 Required="TRUE"
 Group="Contoso Columns">
</Field>
```

- Deploy in a Web-scoped Feature
- Deploy using apps or solutions

```

 Required="TRUE"
 Group="Contoso Columns">
</Field>

<Field ID="{9c2fb13f-62ee-45e6-b73f-ade0489cb69c}"
 Name="ProductionType"
 DisplayName="Production Type"
 Type="Choice"
 Required="TRUE"
 Group="Contoso Columns">
 <CHOICES>
 <CHOICE>Research</CHOICE>
 <CHOICE>Prototype</CHOICE>
 <CHOICE>Trial</CHOICE>
 <CHOICE>Release</CHOICE>
 </CHOICES>
</Field>
</Elements>

```

As you can see from the example, the **Field** element includes attributes that are common to all field types, such as **ID**, **Name**, **DisplayName**, **Type**, **Required**, and **Group**. However, a **Field** element can also include attributes and elements that are specific to certain field types. For example, if you create a choice field, you define the available choices in a **CHOICES** child element.



**Additional Reading:** For comprehensive information about attributes and child elements, see *Field Element (Field)* at <http://go.microsoft.com/fwlink/?LinkID=307105>.

## Deploying site columns

Site columns are deployed in Web-scoped Features. You can make a site column available to every web in a site collection by activating the feature on the root web of a site collection.

You can use both apps and solutions to deploy site columns. If you include a declarative site column in an app, the site column is provisioned on the app web.



**Additional Reading:** For more information about creating site columns declaratively, see *Field Definitions* at <http://go.microsoft.com/fwlink/?LinkID=307106>.

## Creating Site Columns Programmatically

In some circumstances, you may want to create site columns programmatically, rather than declaratively. To some extent, this is a matter of personal preference: some developers prefer writing code to building CAML. The programmatic approach is also useful if you need to add columns dynamically, such as in feature receiver code.

If you are developing an app, the programmatic approach to creating site columns is useful if you want to add a site column to the host web.

Declarative components such as site columns or content types within an app are deployed to the app web, so writing code is the only option if you want to provision artifacts to an alternative location.

- Add new fields to a field collection
  - Site columns: SPWeb.Fields
  - List columns: SPList.Fields
- From server-side code:
  - SPFieldCollection.Add
  - SPFieldCollection.AddFieldAsXml
- From client-side code:
  - FieldCollection.AddFieldAsXml

## Creating site columns by using server-side code

To create a site column programmatically, you must add a new field to the **Fields** collection of a SharePoint web. Depending on your requirements, you can do this in various ways. If you are working with the server-side object model, you can:

- Call the **SPFieldCollection.Add** method, and specify the display name and the field type as arguments.
- Call the **SPFieldCollection.Add** method, and provide an **SPField** instance as an argument.
- Call the **SPFieldCollection.AddFieldAsXml** method, and provide a CAML-based **Field** element as a string argument.

The approach you use will likely depend on the complexity of your field requirements. If you want to create a field that requires little additional configuration, it is easier to call the **Add** method and specify a display name and a field type. If you want to change some of the field properties, it is easier to create an **SPField** instance before you call the **Add** method. Finally, if you want to customize the field extensively, you may find it easier to define the field in CAML and call the **AddFieldAsXml** method.

If you create a new **SPField** instance, you must specify the collection to which you want to add the new field as an argument to the constructor. Fields typically include two constructors: one to instantiate a field that already exists within the collection, and one to create a field that does not already exist within the collection. The difference is that the constructor that creates a new instance additionally requires you to specify the type name of the new field.

The following code example shows how to add fields in server-side code:

### Creating Fields in Server-Side Code

```
var site = SPContext.Current.Site;
var web = site.RootWeb;

// Get the SPFieldCollection for the root web.
var fields = web.Fields;

// Add a new date field.
var fieldExpiryDate = new SPFieldDateTime(fields, SPFieldType.DateTime.ToString(), "Expiry
Date");
fieldExpiryDate.StaticName = "ExpiryDate";
fieldExpiryDate.DisplayFormat = SPDateTimeFieldFormatType.DateOnly;
fieldExpiryDate.Group = "Contoso Columns";
fieldExpiryDate.Required = true;
fieldExpiryDate.Update();
fields.Add(fieldExpiryDate);

// Add a new choice field.
var fieldProductionType = new SPFieldChoice(fields, SPFieldType.Choice.ToString(), "Production
Type");
fieldProductionType.StaticName = "ProductionType";
fieldProductionType.Choices.Add("Research");
fieldProductionType.Choices.Add("Prototype");
fieldProductionType.Choices.Add("Trial");
fieldProductionType.Choices.Add("Release");
fieldProductionType.Group = "Contoso Columns";
fieldProductionType.Required = true;
fieldProductionType.Update();
fields.Add(fieldProductionType);

web.Dispose();
```

## Creating site columns by using client-side code

The SharePoint client object models enable you to work with fields in much the same way as the server-side object model. One key difference is that you can only create a new field by using the **AddFieldAsXml** method of the field collection class.



**Note:** The JavaScript object model and the managed client object model both include **FieldCollection.Add** methods. However, these are intended for adding existing fields to a new collection—for example, to add site columns to the fields collection of a list—rather than for creating new fields.

The following code example shows how to add fields by using the JavaScript object model:

### Creating Fields in Client-Side Code

```
var context;
var web;
var fields;

var addFields = function () {
 context = new SP.ClientContext.get_current();
 web = context.get_web();
 fields = web.get_fields();
 var fieldSchema = '<Field Type="DateTime" \
 Name="ExpiryDate" \
 DisplayName="Expiry Date" \
 Format="DateOnly" \
 Required="TRUE" \
 Group="Contoso Columns" />';
 fields.addFieldAsXml(fieldSchema, false, SP.AddFieldOptions.addFieldCheckDisplayName);
 context.executeQueryAsync(onAddFieldsSuccess, onAddFieldsFail);
}

var onAddFieldsSuccess = function () {
 // Field added successfully. Display a confirmation as required.
}

var onAddFieldsFail = function () {
 // Alert the user that the new field was not created successfully.
}
```

## Retrieving and Editing Site Columns

In some scenarios, you may want to edit site columns programmatically. For example, you might want to provide a more informative description, or amend a list of choices, or change the way your columns are grouped. You can use the following high-level process to retrieve and edit a site column:

1. Retrieve the column from a field collection and cast it to an appropriate type.
2. Update the properties of the field.
3. Call the **Update** method on the field to persist your changes to the content database.

Use the following high-level process to retrieve and update site columns:

1. Retrieve the field from a field collection and cast to an appropriate type.
2. Update field properties as required.
3. Call the **Update** method to persist changes.

The following code example shows how to retrieve and edit a site column in server-side code:

### Retrieving and Editing Fields in Server-Side Code

```
var site = SPContext.Current.Site;
var web = site.RootWeb;

// Get the SPFieldCollection for the root web.
var fields = web.Fields;

// Retrieve the Production Type field and make changes.
var fieldProductionType = fields["Production Type"] as SPFieldChoice;
fieldProductionType.Choices.Clear();
fieldProductionType.Choices.Add("Phase 1 Trial");
fieldProductionType.Choices.Add("Phase 2 Trial");
fieldProductionType.Choices.Add("Phase 3 Trial");
fieldProductionType.Choices.Add("Production");

// Call the Update method.
// Specify false to prevent changes from being cascaded to list columns.
fieldProductionType.Update(false);
```

The process for updating site columns in client-side code is broadly similar to the server-side code process. The major difference is that you also need to manage the client context.

The following code example shows how to retrieve and edit a site column by using the JavaScript object model:

### Retrieving and Editing Fields in Client-Side Code

```
var context;
var web;
var fields;

var updateField = function () {
 context = new SP.ClientContext.get_current();
 web = context.get_web();
 fields = web.get_fields();
 var fieldExpiryDate = context.castTo(fields.getInternalNameOrTitle("ProductionType"),
 SP.FieldChoice);
 var choices = Array("Phase 1 Trial", "Phase 2 Trial", "Phase 3 Trial", "Production")
 fieldExpiryDate.set_choices(choices);
 context.ExecuteQueryAsync(onUpdateFieldSuccess, onUpdateFieldFail);
}

var onUpdateFieldSuccess = function () {
 // Field updated successfully. Display a confirmation as required.
}

var onUpdateFieldFail = function () {
 // Alert the user that the new field was not created successfully.
}
```

## Working with Lookup Fields

SharePoint enables you to create relationships between lists through the use of lookup fields. A lookup field works in a similar way to a choice field, in that the user can select a value from a list of choices. However, in a lookup field, the choices are actually list items from another list within the same site collection. This enables architects and developers to create data models that behave like relationship databases, where the lookup field in a SharePoint list is analogous to the foreign key in a database table. Lookup fields also enable you to create sophisticated queries using **join** statements in LINQ to SharePoint or CAML.

Suppose you use one SharePoint list to track major programs of work, and another SharePoint list to track individual project statuses. Each program contains multiple projects, and each project is associated with a program. You can relate the two lists by adding a Program lookup field to the Projects list. When a user adds a new project to the Projects list, he or she can select the program name in the Program lookup field. If you need to query project or program data, you can then use **join** statements to retrieve related field values from both lists.

The following code example shows how to define a lookup field in CAML:

### Creating a Lookup Field

```
<Field ID="{7fce20b8-9b48-4672-b4c2-011241766c0d}"
 Name="ProgramsLookup"
 DisplayName="Programs"
 Type="Lookup"
 List="Lists\Programs"
 ShowField="ProgramName"
 Overwrite="true"
 Group="Contoso Columns">
</Field>
```

When you define a lookup field in CAML, the **Lists** attribute indicates which list the lookup field should retrieve data from. The **ShowField** attribute indicates which field from the list should be displayed when the lookup field is added to a list. In this example, if a user adds the **ProgramsLookup** field to a list, he or she will be able to select from a list of programs represented by the **ProgramName** value. You can also include *projected fields*, which enable you to display additional fields from the related list in the view that includes the lookup field.

SharePoint enables you to enforce list relationships by specifying deletion rules. There are two types of deletion rule:

- *Cascade delete rule*. When you delete a list item, this rule deletes all list items in related lists that reference the primary list item through a lookup field.
- *Restrict delete rule*. This rule prevents you from deleting a list item if it is referenced by items in a related list through a lookup field.



**Note:** For a detailed description of lookup fields and list relationships, see *List Relationships in SharePoint 2010* at <http://go.microsoft.com/fwlink/?LinkID=307107>. The article was written for SharePoint 2010, but the concepts apply equally to SharePoint 2013.

- Use lookup fields to create list relationships
- Conceptually similar to foreign keys in relational databases
- Enables sophisticated query construction

```
<Field ID="{7fce20b8-9b48-4672-b4c2-011241766c0d}"
 Name="ProgramsLookup"
 DisplayName="Programs"
 Type="Lookup"
 List="Lists\Programs"
 ShowField="ProgramName"
 Overwrite="true"
 Group="Contoso Columns">
</Field>
```

## Discussion: Using Lookup Fields

The instructor will now lead a brief discussion around the types of scenarios in which you would use lookup fields.

- In what scenarios might you use lookup fields?

## Lesson 2

# Working with Content Types

Content types are a powerful tool for ensuring that content is organized consistently. Before content types were introduced in SharePoint 2007, metadata was managed at the individual list level. There was no way of associating metadata requirements with specific types of content, and no easy way of ensuring consistency across multiple lists. By breaking the coupling between lists and metadata requirements, content types can help you to manage content consistently regardless of its location in the SharePoint logical architecture. In this lesson, you will learn how to work with content types in both client-side and server-side solutions.

### Lesson Objectives

After completing this lesson, you will be able to:

- Create content types declaratively.
- Construct content type IDs that specify an inheritance hierarchy.
- Create and edit content types programmatically.
- Add site content types to lists.

### Creating Content Types Declaratively

Content types are designed to define and encapsulate all the metadata, business processes, and UI requirements for a particular type of content. Content types should classify information by business purpose, rather than by file type. For example, you might create content types to represent invoices, or vacation requests, or expense claims. You can use content types to define:

- The metadata (fields) you require to manage the item.
- The document template for the item.
- The Display, Edit, and New forms that are used to manage the item.

#### Creating content types

You can define content types declaratively by using the **ContentType** element in CAML. The **ContentType** element enables you to define all the key aspects of the content type properties and behaviors.

- Use the **ContentType** element to define:
  - Metadata
  - Document template
  - Custom forms

```
<ContentType
 ID="0x010100742830d3B25349C7A83DF4AEF639BFD5"
 Name="Contract"
 Inherits="TRUE"
 Version="0">
 <FieldRefs>
 <RemoveFieldRef ID="{...}" Name="..." />
 <FieldRef ID="{...}" Name="..." Required="TRUE"/>
 ...
 </FieldRefs>
</ContentType>
```



The following example shows how to define a content type in CAML:

### Defining a Content Type

```
<ContentType ID="0x010100742830D3B25349C7A83DF4AEF639BFD5"
 Name="Contract"
 Group="Contoso Content Types"
 Description="A contract to supply goods or services"
 Inherits="TRUE"
 Version="0">
 <FieldRefs>
 <RemoveFieldRef ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}" Name="Title" />
 <FieldRef ID="{ff77d8e3-c172-43e6-b729-d3e6492e6334}"
 Name="CustomerName"
 Required="TRUE" />
 <FieldRef ID="{fd125d1e-ad63-436e-9f1f-efce7ef00967}"
 Name="StartDate"
 Required="TRUE" />
 <FieldRef ID="{b2db5ee2-e09e-4864-b00f-3f652187c843}"
 Name="FinalEffectiveDate"
 Required="TRUE" />
 <FieldRef ID="{8da52b24-493d-4475-a812-875ba2c0faf4}"
 Name="Owner"
 Required="TRUE" />
 <FieldRef ID="{cc4fd24e-f357-45e7-bb4c-a7567aae6347}"
 Name="Department"
 Required="TRUE" />
 </FieldRefs>
 <DocumentTemplate TargetName="ContosoContract.dotx" />
</ContentType>
```

In the example, the **Inherits** attribute indicates whether the content type should inherit fields from its parent. Content type inheritance is specified by the content type **ID** value, which is described in more detail in the next topic. After specifying that the content type should inherit fields from its parent, you can use **RemoveFieldRef** child elements to remove any inherited fields that you do not require.

To add new fields to the content type, you use the **FieldRef** element. You can reference fields that already exist on the target site, or fields that are defined within the same feature as the content type.

### Deploying content types

Like site columns, content types are deployed in Web-scoped or Site-scoped Features. To make a content type available to every web in a site collection, you should use a site-scoped feature. In this case, the content types are deployed to the root web in the site collection.

You can use both apps and solutions to deploy content types. If you include a declarative content type in an app, the content type is provisioned on the app web.

## Understanding Content Type IDs

All content types belong in an inheritance hierarchy, and ultimately inherit from one of the built-in base content types. However, the **ContentType** element does not explicitly specify the parent of the content type. Instead, inheritance is specified through the content type ID.

Every content type ID begins with the ID of a base content type. The following table shows the ID values of some common base content types:

Name	ID
Item	0x01
Document	0x0101
Event	0x0102
Announcement	0x0104
Task	0x0108

You can use the following rules to build a content type ID:

1. Start with the ID of the parent content type.
2. Append **00**, which you can think of as a spacer.
3. Append an uppercase GUID, without any hyphens, spaces, or braces.

For example, suppose you want to create a content type named **Invoice** that inherits from **Document**. You start with the ID of the **Document** content type:

**0x0101**

Next, you append a double zero:

**0x010100**

Finally, you append a GUID:

**0x0101005AF7FDFCE5FD4C359A7AE34DFB008661**

When you deploy this content type, SharePoint will deduce the parentage from your content type ID. In this case, your content type will inherit from the **Document** content type. If you have set the **Inherits** attribute to **TRUE**, your content type will include all the fields that are present in the **Document** content type.

- All content types inherit from a parent
- Inheritance is specified through the content type ID
- To create a content type ID:
  1. Start with the ID of the parent content type  
**0x0101**
  2. Append a double-zero  
**0x010100**
  3. Append a GUID  
**0x0101005AF7FDFCE5FD4C359A7AE34DFB008661**

Suppose you now want to create a more specialized type of Invoice content type named **Northwind Traders Invoice**. The **Northwind Traders Invoice** content type should inherit from the **Invoice** content type. You can use the same approach to build the new content type ID. You start with the ID of the **Invoice** content type:

```
0x0101005AF7FDFCE5FD4C359A7AE34DFB008661
```

Next, you append a double zero:

```
0x0101005AF7FDFCE5FD4C359A7AE34DFB00866100
```

Finally, you append a GUID:

```
0x0101005AF7FDFCE5FD4C359A7AE34DFB00866100BB4707E674E049D48B2BB38FE625C4B2
```

You can use this approach to build as many levels of content type inheritance as you require.

## Demonstration: Using the Visual Studio 2012 Content Type Designer

The instructor will now demonstrate how you can use the Visual Studio 2012 content type designer to build a content type definition interactively.

### Demonstration Steps

- Start the 20488B-LON-SP-12 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
- In Visual Studio, on the start page, click **New Project**.
- In the **New Project** dialog box, in the navigation pane, expand **Template**, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
- In the center pane, click **SharePoint 2013 - Empty Project**.
- In the **Name** box, type **ContentTypeDemo**.
- In the **Location** box, type **E:\Democode\**, and then click **OK**.
- In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://team.contoso.com**.
- Click **Deploy as a farm solution**, and then click **Finish**.
- In Solution Explorer, right-click the **ContentTypeDemo** project node, point to **Add**, and then click **New Item**.
- In the **Add New Item - ContentTypeDemo** dialog box, click **Site Column**.
- In the **Name** box, type **ClientName**, and then click **Add**.

- In the **Elements.xml** file, amend the **Field** element as shown by the bolded text:

```
<Field
 ID="{...}"
 Name="ClientName"
 DisplayName="Client Name"
 Type="Text"
 Required="TRUE"
 Group="Contoso Columns">
</Field>
```

- Save and close the **Elements.xml** file.
- In Solution Explorer, right-click the **ContentTypeDemo** project node, point to **Add**, and then click **New Item**.
- In the **Add New Item - ContentTypeDemo** dialog box, click **Site Column**.
- In the **Name** box, type **PaymentDueDate**, and then click **Add**.
- In the **Elements.xml** file, amend the **Field** element as shown by the bolded text:

```
<Field
 ID="{...}"
 Name="PaymentDueDate"
 DisplayName="Payment Due Date"
 Type="DateTime"
 Format="DateOnly"
 Required="TRUE"
 Group="Contoso Columns">
</Field>
```

- Save and close the **Elements.xml** file.
- In Solution Explorer, right-click the **ContentTypeDemo** project node, point to **Add**, and then click **New Item**.
- In the **Add New Item - ContentTypeDemo** dialog box, click **Site Column**.
- In the **Name** box, type **Amount**, and then click **Add**.
- In the **Elements.xml** file, amend the **Field** element as shown by the bolded text:

```
<Field
 ID="{...}"
 Name="Amount"
 DisplayName="Amount"
 Type="Currency"
 LCID="1033"
 Required="TRUE"
 Group="Contoso Columns">
</Field>
```

- Save and close the **Elements.xml** file.
- In Solution Explorer, right-click the **ContentTypeDemo** project node, point to **Add**, and then click **New Item**.
- In the **Add New Item - ContentTypeDemo** dialog box, click **Content Type**.
- In the **Name** box, type **Invoice**, and then click **Add**.
- In the **SharePoint Customization Wizard** dialog box, in the **Which base content type should this content type inherit from** list, click **Document**, and then click **Finish**.
- On the **Columns** tab, in the first row, type **Client Name**, and then press Tab three times.

- In the second row, type **Payment Due Date**, and then press Tab three times.
- In the third row, type **Amount**, and then press Tab three times.
- On the **Content Type** tab, in the **Description** box, type **Invoices sent to Contoso clients**.
- In the **Group Name** box, type **Contoso Content Types**.
- On the **FILE** menu, click **Save All**, and then close the **Invoice** page.
- In Solution Explorer, expand the **Invoice** node, and then double-click **Elements.xml**.
- Review the **ContentType** element that Visual Studio has created for you. In particular, note that the content type ID follows the inheritance pattern described in the previous topic.
- Close Visual Studio.

## Working with Content Types in Code

In addition to creating content types declaratively in CAML, you can create and edit content types programmatically using server-side or client-side code. Choosing the declarative approach or the programmatic approach to creating content types is largely a matter of personal preference: some developers prefer the declarative approach for readability, whereas other developers consider the programmatic approach more flexible and more robust. In most cases, the programmatic approach is the only option if you need to edit existing content types.

- In server-side code:
  - Create an **SPContentType** object to represent the content type
  - Create **SPFieldLink** objects to represent field references
- In client-side code:
  - Create a **ContentTypeCreationInformation** object to represent content type properties
  - Create **FieldLinkCreationInformation** objects to represent field reference properties

### Creating content types by using server-side code

To create a new content type using server-side code, you create an object of type **SPContentType**. To create a new **SPContentType** instance, you must provide three pieces of information:

- Either the ID of the parent content type, or an **SPContentType** instance that represents the parent content type.
- The content type collection to which you want to add the new content type. Both the **SPWeb** class and the **SPList** class expose a content type collection through the **ContentTypes** property.
- The name of the new content type.

In the server-side object model, field references are represented by the **SPFieldLink** class. You must create an **SPFieldLink** instance for every field that you want to add to the content type. You then add the **SPFieldLink** instances to the **FieldLinks** collection of the content type.

The following example shows how to create a content type in server-side code:

### Creating a Content Type in Server-Side Code

```
var web = SPContext.Current.Site.RootWeb;
var fields = web.fields;

// Get the ID of the parent content type.
var parentId = SPBuiltInContentTypeId.Document;

// Create the Invoice content type.
var ctInvoice = new SPContentType(parentId, web.ContentTypes, "Invoice");

// Create field links.
var fldClientName = fields.GetField("ClientName");
var fldPaymentDueDate = fields.GetField("PaymentDueDate");
var fldAmount = fields.GetField("Amount");
var fldLinkClientName = new SPFieldLink(fldClientName);
var fldLinkPaymentDueDate = new SPFieldLink(fldPaymentDueDate);
var fldLinkAmount = new SPFieldLink(fldAmount);

// Add the field links to the content type.
ctInvoice.FieldLinks.Add(fldLinkClientName);
ctInvoice.FieldLinks.Add(fldLinkPaymentDueDate);
ctInvoice.FieldLinks.Add(fldLinkAmount);

// Persist the changes to the content database.
ctInvoice.Update();

// Add the content type to the collection.
web.ContentTypes.Add(ctInvoice);

web.Dispose();
```

### Creating content types by using client-side code

Creating content types in client-side code requires a slightly different approach. To create a content type, you must first create a **ContentTypeCreationInformation** object. Similarly, to create a field link, you must first create a **FieldLinkCreationInformation** object.

The following example shows how to create a content type by using the JavaScript object model:

### Creating a Content Type in Client-Side Code

```
var context;
var web;
var fields;
var contentTypes;

var addContentType = function () {
 context = new SP.ClientContext.get_current();
 web = context.get_web();
 fields = web.get_availableFields();
 contentTypes = web.get_contentTypes();

 // Get a reference to the parent content type.
 var parent = contentTypes.getById("0x0101");

 // Create an SP.ContentTypeCreationInformation object.
 var ctInfo = new SP.ContentTypeCreationInformation();
 ctInfo.set_parentContentType(parent);
 ctInfo.set_name("Invoice");
 ctInfo.set_group("Contoso Content Types");

 // Create the Invoice content type.
 var ctInvoice = contentTypes.add(ctInfo);
```

```

// Add field links to the content type.
var fieldLinks = ctInvoice.get_fieldLinks();

var fldClientName = fields.getByInternalNameOrTitle("ClientName");
var fldLinkInfoClientName = new SP.FieldLinkCreationInformation();
fldLinkInfoClientName.set_field(fldClientName);
fieldLinks.add(fldLinkInfoClientName);

var fldPaymentDueDate = fields.getByInternalNameOrTitle("PaymentDueDate");
var fldLinkInfoPaymentDueDate = new SP.FieldLinkCreationInformation();
fldLinkInfoPaymentDueDate.set_field(fldPaymentDueDate);
fieldLinks.add(fldLinkInfoPaymentDueDate);

var fldAmount = fields.getByInternalNameOrTitle("Amount");
var fldLinkInfoAmount = new SP.FieldLinkCreationInformation();
fldLinkInfoAmount.set_field(fldAmount);
fieldLinks.add(fldLinkInfoAmount);

// Persist the changes to the content type.
ctInvoice.update();

context.executeQueryAsync(onAddContentTypeSuccess, onAddContentTypeFail);
}

var onAddContentTypeSuccess = function () {
 // Content type added successfully. Display a confirmation as required.
}

var onAddContentTypeFail = function () {
 // Alert the user that the new content type was not created successfully.
}

```

## Adding Content Types to Lists

To use a content type, you must add it to a list or library. Users with sufficient permissions can add content types to lists through the user interface. If you need to automate the process, you can use both declarative and programmatic approaches to add content types to lists. The association between a specific content type and a specific list is known as a *content type binding*.

### Creating content type bindings declaratively

You can add a content type to a list declaratively by adding a **ContentTypeBinding** element to an element manifest, and then deploying the element manifest within a Web-scoped or Site-scoped Feature. The **ContentTypeBinding** element simply requires you to specify the ID of the content type and the site-relative URL of the list.

- Add content type to list declaratively:

```

<ContentTypeBinding
 ContentTypeId="0x010100..."
 ListUrl="Invoices"
 RootWebOnly="FALSE" />

```

- Add content type to list programmatically:

```

var web = SPContext.Current.Web;

var list = web.GetList("Invoices");
var contentType = web.AvailableContentTypes["Invoice"];

list.ContentTypeEnabled = true;
list.ContentTypes.Add(contentType);

list.Update();

```

The following code example shows how to define a content type binding declaratively:

### Defining a Content Type Binding

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <ContentTypeBinding
 ContentTypeId="0x0101005AF7FDfce5FD4C359A7AE34DFB008661"
 ListUrl="Invoices"
 RootWebOnly="FALSE"
 />
</Elements>
```

### Creating content type bindings programmatically

To create a content type binding programmatically, you essentially need to retrieve a content type from the site's collection of content types and add it to the list's collection of content types. You also need to ensure that content types are enabled on the list before you attempt to add to the collection.

The following code example shows how to add a list content type in server-side code:

### Adding a List Content Type in Server-Side Code

```
var web = SPContext.Current.Web;

// Get the list.
var list = web.GetList("Invoices");

// Get the site content type.
var contentType = web.AvailableContentTypes["Invoice"];

// Enable content types on the list.
list.ContentTypesEnabled = true;

// Add the content type to the list.
list.ContentTypes.Add(contentType);

// Persist the changes to the database.
list.Update();
```



You can use a similar pattern to add list content types in client-side code. The following code example shows how to add a list content type by using the JavaScript object model:

### Adding a List Content Type in Client-Side Code

```
var addListContentType = function () {
 var context = new SP.ClientContext.get_current();
 var web = context.get_web();

 // Get a reference to the site content type.
 var contentTypes = web.get_availableContentTypes();
 var invoiceCT = contentTypes.getById("0x0101005AF7FDFCE5FD4C359A7AE34DFB008661");

 // Get a reference to the list.
 var lists = web.get_lists();
 var list = lists.getByTitle("Invoices");

 // Enable content types on the list.
 list.set_contentTypesEnabled(true);

 // Add the content type to the list.
 var listContentTypes = list.get_contentTypes();
 listContentTypes.addExistingContentType(invoiceCT);

 // Persist the changes to the database.
 list.update();

 // Submit the query.
 context.executeQueryAsync(onAddListContentTypeSuccess, onAddListContentTypeFail);
}

var onAddListContentTypeSuccess = function () {
 // List content type added successfully. Display a confirmation as required.
}

var onAddListContentTypeFail = function () {
 // Alert the user that the list content type was not added successfully.
}
```

## Lab A: Working with Content Types

### Scenario

The HR team at Contoso requires a solution for managing vacation requests. The team already uses a list named Vacation Tracker to maintain a record of the remaining vacation entitlement for each employee. They now want to enable employees to submit requests for vacations by specifying a start date and an end date. Your task is to develop site columns, content types, and a list template to implement this solution.

### Objectives

After completing this lab, you will be able to:

- Create and configure site columns declaratively.
- Create and configure content types declaratively.
- Create and configure list templates and list instances declaratively.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-12
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

### Exercise 1: Create a System to Capture Vacation Requests

#### Scenario

In this exercise, you will create site columns, content types, and a list in a SharePoint solution package. You will then deploy and test the solution.

The main tasks for this exercise are as follows:

1. Create an Empty SharePoint 2013 Project
2. Create New Site Columns
3. Create the Vacation Request Content Type
4. Create the Vacation Requests List Instance
5. Deploy and Test the Vacation Requests List

#### ► Task 1: Create an Empty SharePoint 2013 Project

- Start the 20488B-LON-SP-12 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio 2012, and create a new project with the following properties:
  - Use the **SharePoint 2013 - Empty Project** template.
  - Set the Visual Studio solution name to **VacationRequests**.
  - Create the solution in the **E:\Labfiles\Starter** folder.
  - Use the site at **http://hr.contoso.com** for debugging.
  - Specify the farm solution deployment option.

► **Task 2: Create New Site Columns**

- In the **VacationRequests** project node, add a new site column named **ContosoDepartment**.
- Make the following changes to the **ContosoDepartment** field:
  - Change the type of the field to **Choice**.
  - Specify that the field must be assigned a value.
  - Add the field to the **Contoso Columns** group.
  - Add the following choices to the **ContosoDepartment** field:
    - Finance
    - Human Resources
    - IT
    - Legal
    - Manufacturing
    - Research
- Add a new site column named **LineManager**.
- Make the following changes to the **LineManager** field:
  - Change the type of the field to **User**.
  - Specify that the field must be assigned a value.
  - Add the field to the **Contoso Columns** group.
- Add a new site column named **VacationRequestStatus**.
- Make the following changes to the **VacationRequestStatus** field:
  - Change the type of the field to **Choice**.
  - Specify that the field must be assigned a value.
  - Add the field to the **Contoso Columns** group.
  - Add the following choices to the **VacationRequestStatus** field:
    - Pending
    - Approved
    - Rejected
    - Booked
- Set the default value of the **VacationRequestStatus** field to **Pending**.
- Save your work, and then build the solution and check for errors.

► **Task 3: Create the Vacation Request Content Type**

- In the **VacationRequests** project node, add a new site content type named **VacationRequest**. The content type should inherit from the **Item** base type.
- Add the following columns to the content type:
  - Employee
  - Contoso Department

- Line Manager
- Start Date
- End Date
- Vacation Request Status



**Note:** The **Employee** site column is already deployed to the HR site collection, in the **Contoso Columns** group. The **Contoso Department**, **Line Manager**, and **Vacation Request Status** columns are defined within the current project. **Start Date** and **End Date** are built-in base columns.

- Make sure that all fields are compulsory.
  - Set the content type name to **Vacation Request**.
  - Set the content type description to **Request a new vacation booking**.
  - Set the content type group to **Contoso Content Types**.
  - Save your work, and then rebuild the solution and check for errors.
- ▶ **Task 4: Create the Vacation Requests List Instance**
- In the **VacationRequests** project node, add a new list named **VacationRequests** with the following properties:
    - Set the display name of the list to **Vacation Requests**.
    - Use the **Create a customizable list template and a list instance of it** option.
    - Use **Default (Custom List)** as the basis for the list template.
    - Add the **Vacation Request** content type to the list, and remove all other content types.
    - Set the list description to **Use this list to create and manage requests for vacations**.
  - Save your work, and then rebuild the solution and check for errors.
- ▶ **Task 5: Deploy and Test the Vacation Requests List**
- Rename the project feature from **Feature1** to **VacationRequests**.
  - Set the feature title to **Vacation Requests**.
  - Set the feature description to **Deploys site columns, a content type, and a list for managing vacation requests**.
  - Save your work, and then on the **DEBUG** menu, click **Start Without Debugging**.
  - If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
  - In Internet Explorer, after the page finishes loading, on the Quick Launch navigation menu, click **Vacation Requests**.
  - On the ribbon, on the **ITEMS** tab, on the **New Item** menu, click **Vacation Request**.
  - In the **Title** box, type **Two weeks trekking in the Rockies!**.
  - In the **Employee** box, type **Joel**, and then click **Joel Frauenheim**.
  - In the **Contoso Department** list, click **IT**.

- In the **Line Manager** box, type **Tihomir**, and then click **Tihomir Sasic**.
- In the **Start Date** box, select a date within the next twelve months.
- In the **End Date** box, select a date two weeks from the start date, and then click **Save**.



**Note:** In a real-world deployment, you would use a workflow to manage the **Vacation Request Status** field, rather than allowing users to edit it from the default forms.

- Close Internet Explorer.
- Close Visual Studio 2012.

**Results:** After completing this exercise, you should have deployed all the components required to capture and manage vacation requests.

## Lesson 3

## Working with Advanced Features of Content Types

Content types are at their most powerful when you use them in conjunction with other SharePoint capabilities. For example, you can associate workflows with content types, so that the workflow runs on any list to which the content type is added. Similarly, you can add event receivers to content types, so that events for relevant items are handled consistently, regardless of which list contains the item. In this lesson, you will learn how to make use of these advanced features of content types.



**Note:** You can also associate policies with content types. However, policies and policy associations are managed through the **Microsoft.Office.RecordsManagement.InformationPolicy** namespace and not through the **SPContentType** API. Creating policies programmatically is covered in Course 20489, *Developing Microsoft SharePoint Server 2013 Advanced Solutions*.

### Lesson Objectives

After completing this lesson, you will be able to:

- Manage the association between content types and document templates.
- Configure workflow associations for content types.
- Add event receivers to content types.

### Managing Document Templates

In many scenarios, organizations may find it useful to associate document templates with content types. For example, if you are creating an invoice, you would be very unlikely to start with a blank document. Instead, you are likely to use an invoice template with formatting and boilerplate text, together with placeholders for amounts, dates, and descriptions. If you associate a template with the Invoice content type, SharePoint will launch the template every time a user creates a new Invoice item from a document library menu.

In the previous lesson, you saw how you can add a **DocumentTemplate** child element to a declarative content type definition. Although the declarative approach is useful when you are creating a new content type, it does not allow you to change the document template associated with an existing content type. Fortunately, document templates are easy to manage using either server-side or client-side code.

The following code example shows how to set or update the document template for a site content type in server-side code:

#### Setting a Document Template in Server-Side Code

```
var web = SPContext.Current.Web;
var invoiceCT = web.AvailableContentTypes["Invoice"];
invoiceCT.DocumentTemplate = "SiteAssets/Invoice.dotx";
invoiceCT.Update(true);
```

- Retrieve the content type
- Set the **DocumentTemplate** property
- Call **Update(true)** to cascade changes to the content type in lists

```
var web = SPContext.Current.Web;
var invoiceCT = web.AvailableContentTypes["Invoice"];
invoiceCT.DocumentTemplate = "SiteAssets/Invoice.dotx";
invoiceCT.Update(true);
```

In this example, note that we pass an argument value of **true** to the **Update** method. This specifies that the updates should be cascaded to children of the content type. In practice, this means that any instances of the content type in lists are also updated to use the new document template.

You can use the same approach to manage document templates with client-side code. The following code example shows how to set or update the document template for a site content type by using the JavaScript object model:

### Setting a Document Template in Client-Side Code

```
var updateDocumentTemplate = function () {
 var context = new SP.ClientContext.get_current();
 var web = context.get_web();
 var contentTypes = web.get_availableContentTypes();
 var invoiceCT = contentTypes.getById("0x0101005AF7FDFCE5FD4C359A7AE34DFB008661");
 invoiceCT.set_documentTemplate("SiteAssets/Invoice.dotx");
 invoiceCT.update(true);
 context.executeQueryAsync(onUpdateTemplateSuccess, onUpdateTemplateFail);
}

var onUpdateTemplateSuccess = function () {
 // List content type added successfully. Display a confirmation as required.
}

var onUpdateTemplateFail = function () {
 // Alert the user that the list content type was not added successfully.
}
```

## Configuring Workflow Associations

Content types are a tool for modeling business content. Workflows are a tool for modeling business processes. As such, it makes sense to be able to associate workflows with content types. Rather than adding workflows to individual lists, adding a workflow to the content type ensures that the workflow is available and active wherever the content type is added.

You can associate workflows with content types by using the SharePoint object model (there is no declarative approach). Both the workflow template and the content type must be deployed to the SharePoint site before you create the workflow association. The high-level process for associating a workflow with a content type is as follows:

1. Retrieve the workflow template from the SharePoint web.
2. Create a new workflow association object, by specifying the workflow template, providing a name for the workflow instance, and specifying the task list and the history list that you want to use.
3. Retrieve the content type from the SharePoint web.
4. Add the workflow association to the content type.

The following code example shows how to associate a workflow with a content type by using the server-side object model:

#### Why associate workflows with content types?

- Content types represent business content
- Workflows represent business processes
- Linking the two is logical

#### High-level process:

1. Retrieve the workflow template from the site
2. Create a workflow association for the workflow template
3. Retrieve the content type from the site
4. Add the workflow association to the content type
5. Cascade the changes to list content types

### Associating a Workflow with a Content Type in Server-Side Code

```

var web = SPContext.Current.Web;

// Get the workflow template.
SPWorkflowTemplate template = web.WorkflowTemplates.GetTemplateByName("InvoiceWorkflow",
 web.Locales);

// Create a new workflow association.
SPWorkflowAssociation association =
 SPWorkflowAssociation.CreateWebContentTypeAssociation(template, "Process Invoice", "Invoice
 Tasks", "Process Invoice History");

// Get the site content type.
var contentType = web.AvailableContentTypes["Invoice"];

// Add the workflow association to the content type.
if(contentType.WorkflowAssociations.GetAssociationByName("Process Invoice", web.Locales) == null)
{
 contentType.WorkflowAssociations.Add(association);
 contentType.UpdateWorkflowAssociationsOnChildren(false, true, true, false);
}

```



**Note:** The **SPWorkflowAssociation.CreateWebContentTypeAssociation** method requires you to specify the name of a tasks list and a workflow history list. If these lists do not already exist, SharePoint will create them when the workflow instance is used for the first time.

Associating a workflow with a content type in client-side code is slightly different. Rather than adding a workflow association to the content type, you construct an object of type **WorkflowAssociationCreationInformation** and pass this object to the **add** method of the workflow association collection.

The following code example shows how to associate a workflow with a content type by using the JavaScript object model:

### Associating a Workflow with a Content Type in Client-Side Code

```

var addWorkflowAssociation = function () {
 var context = new SP.ClientContext.get_current();
 var web = context.get_web();

 // Get a reference to the workflow template.
 var templates = web.get_workflowTemplates();
 var template = templates.getByNamed("InvoiceWorkflow");

 // Create a WorkflowAssociationCreationInformation object.
 var info = new SP.Workflow.WorkflowAssociationCreationInformation();
 info.set_template(template);
 info.set_name("Process Invoice");
 info.set_contentTypeAssociationTaskList("Invoice Tasks");
 info.set_contentTypeAssociationHistoryList("Invoice History");

 // Get a reference to the site content type.
 var contentTypes = web.get_availableContentTypes();
 var invoiceCT = contentTypes.getById("0x0101005AF7FDCE5FD4C359A7AE34DFB008661");

 // Add the association to the site content type.
 var associations = invoiceCT.get_workflowAssociations();
 associations.add(info);
 invoiceCT.update(true);

 // Submit the query.
 context.executeQueryAsync(onAddAssociationSuccess, onAddAssociationFail);
}

```



```

}

var onAddAssociationSuccess = function () {
 // List content type added successfully. Display a confirmation as required.
}

var onAddAssociationFail = function () {
 // Alert the user that the list content type was not added successfully.
}

```

## Associating Event Receivers with Content Types

You can enhance the functionality of content types by associating event receivers with site content types and list content types. To be compatible with a content type, the event receiver class must derive from **SPItemEventReceiver**. You can use the event receiver class to handle the full range of item events for your content type, including **ItemAdding** and **ItemAdded**, **ItemDeleting** and **ItemDeleted**, **ItemCheckingIn** and **ItemCheckedIn**, and many more.

- Event receiver class must inherit from **SPItemEventReceiver**
- Add to content type declaratively or programmatically
  - Declaratively for new content types
  - Programmatically for new or existing content types

The main advantage of associating an event receiver with a content type, rather than a list, is flexibility. Suppose you need to perform some computation every time a user adds an item of content type Invoice, such as running a currency conversion or calculating a tax rate. You could use an item event receiver to complete these tasks. You could add the event receiver to your Invoices list, but what would happen if you store invoices in more than one location? Every time you add the Invoice content type to a list, you would also need to proactively add the event receiver to the list. By associating the event receiver with the Invoice content type, you ensure that the event receiver functionality is active wherever the Invoice content type is used.



**Note:** Creating and deploying event receivers is covered in Module 5, *Working with Server-Side Code*. This topic focuses on how to associate a deployed event receiver with a content type.



**Note:** You cannot currently associate a remote event receiver with a content type, because the client-side **SP.ContentType** class does not have an **EventReceivers** collection.

### Adding event receivers declaratively

You can associate an event receiver with a content type as part of the declarative content type definition. The **ContentType** element can include a child element of type **XmlDocuments**, which can in turn contain child elements of type **XmlDocument**. You can use the **XmlDocument** element to add custom or supplementary information to your content type definition. For example, you can specify the form templates you want to use with your content type by adding a **FormTemplates** element within an **XmlDocument** element. In this case, you can add event receivers to your content type by adding a **Receivers** element within an **XmlDocument** element.

The following code example shows how to register an event receiver as part of a content type definition:

### Adding an Event Receiver Declaratively

```
<ContentType ID="0x0101005AF7FDFCE5FD4C359A7AE34DFB008661"
 Name="Invoice"
 Group="Contoso Content Types"
 Description="Invoices sent to Contoso clients."
 Inherits="TRUE"
 Version="0">
 <FieldRefs>
 <FieldRef ID="{...}" Name="ClientName" Required="TRUE" />
 <FieldRef ID="{...}" Name="PaymentDueDate" Required="TRUE" />
 <FieldRef ID="{...}" Name="Amount" Required="TRUE" />
 </FieldRefs>
 <DocumentTemplate TargetName="SiteAssets/ContosoInvoice.dotx" />
 <XmlDocuments>
 <XmlDocument NamespaceURI="http://schemas.microsoft.com/sharepoint/events">
 <Receivers xmlns:spe=" http://schemas.microsoft.com/sharepoint/events">
 <Receiver>
 <Name>InvoiceReceiver</Name>
 <Type>ItemAdding</Type>
 <SequenceNumber>10050</SequenceNumber>
 <Assembly>ContosoEvents, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=...</Assembly>
 <Class>ContosoEvents.InvoiceEventReceiver</Class>
 <Data></Data>
 <Filter></Filter>
 </Receiver>
 </Receivers>
 </XmlDocument>
 </XmlDocuments>
</ContentType>
```

When you add event receiver definitions declaratively, the relevant properties of each event receiver are represented by child elements of the **Receiver** element. For example:

- Use the **Name** element to specify a name that uniquely identifies your event receiver.
- Use the **Type** element to specify which type of event receiver you are defining (for example, **ItemAdding** or **ItemUpdated**).
- Use the **SequenceNumber** element to specify the order in which your event receivers should execute. Lower numbers are executed before higher numbers. To avoid conflicts with built-in event receivers, you should use an integer value of 10,000 or greater.
- Use the **Assembly** element to specify the strong name of the assembly that contains your event receiver class.
- Use the **Class** element to specify the fully-qualified name of your event receiver class.

Note that you cannot use a declarative approach to add an event receiver definition to an existing content type. If you want to add or remove event receiver definitions from existing content types, you must use the server-side object model.

### Adding event receivers programmatically

You can use the server-side object model to add or remove event receivers from existing site or list content types. The **SPContentType** class includes an **EventReceivers** property, which is a collection of **SPEventReceiverDefinition** objects. To associate an event receiver with a content type, you add a new item to the **EventReceivers** collection.

The following code example shows how to register an event receiver with a site content type programmatically:

### Adding an Event Receiver Programmatically

```
// Get the Invoice content type.
var invoiceCT = web.ContentTypes["Invoice"];
if (invoiceCT != null)
{
 // Add a new event receiver definition to the collection.
 SPEventReceiverDefinition erd = invoiceCT.EventReceivers.Add();

 // Specify the properties of the event receiver.
 erd.Assembly = "<Assembly strong name>";
 erd.Class = "<Fully-qualified class name>";
 erd.Type = SPEventReceiverType.ItemAdding;
 erd.Name = "ItemAdding Event Receiver";
 erd.Synchronization = SPEventReceiverSynchronization.Synchronous;
 erd.SequenceNumber = 10050;
 erd.Update();

 // Update the content type and its children.
 invoiceCT.Update(true);
}
```

## Lab B: Working with Advanced Features of Content Types

### Scenario

The HR team at Contoso is pleased with the new Vacation Requests list. The team now wants to integrate the Vacation Requests list with the existing Vacation Tracker list:

- When a user requests a new vacation booking through the Vacation Requests list, SharePoint should automatically check the Vacation Tracker list to make sure that the user has sufficient vacation days remaining.
- If the user does not have sufficient vacation days remaining, SharePoint should prevent the user from submitting the request and display an informative error message.
- When a line manager approves a vacation request, SharePoint should automatically update the user's remaining vacation entitlement in the Vacation Tracker list.
- After the Vacation Tracker list is updated, SharePoint should set the status of the vacation request to "Booked". In practice, this functionality might also update a third-party staff scheduling system.

Your task is to use event receivers to implement this functionality. You will use an **ItemAdding** event receiver to check that a user has sufficient vacation days to cover the request, and you will use an **ItemUpdated** event receiver to update the Vacation Tracker list when a vacation request is approved. You will then add your event receivers to the Vacation Request content type.

### Objectives

After completing this lab, you will be able to:

- Implement an item event receiver.
- Programmatically register an event receiver with a site content type.

### Lab Setup

Estimated Time: 45 minutes

- Virtual Machine: 20488B-LON-SP-12
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

If you need to debug event receiver functionality, use the following procedure:

1. In Visual Studio 2012, on the **DEBUG** menu, click **Start Without Debugging**.



**Note:** In a classroom environment, the **Start Debugging** option will cause SharePoint and Visual Studio to run very slowly. In addition, Visual Studio may not automatically attach to the correct process for debugging event receivers.

2. Insert breakpoints in your event receiver code as required.
3. In Visual Studio, on the **DEBUG** menu, click **Attach to Process**.
4. Select **Show processes from all users**, and then click **Refresh**.
5. Select the **w3wp.exe** process with the user name **CONTOSO\SPContent**, and then click **Attach**.
6. In the browser, perform the actions that trigger the event receiver.

## Exercise 1: Creating an Event Receiver Assembly

### Scenario

In this exercise, you will create and compile an event receiver class that handles **ItemAdding** and **ItemUpdated** events.

The main tasks for this exercise are as follows:

1. Create an Empty SharePoint 2013 Project
2. Create the Event Receiver Class
3. Create the ItemAdding Method
4. Create the ItemUpdated Method

#### ► Task 1: Create an Empty SharePoint 2013 Project

- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio 2012, and create a new project with the following properties:
  - Use the **SharePoint 2013 - Empty Project** template.
  - Set the Visual Studio solution name to **VacationRequestsEventReceiver**.
  - Create the solution in the **E:\Labfiles\Starter** folder.
  - Use the site at **http://hr.contoso.com** for debugging.
  - Specify the farm solution deployment option.

#### ► Task 2: Create the Event Receiver Class

- Add a new class file named **VacationRequestEventReceiver.cs** to the project.



**Note:** You are adding a class manually, rather than adding an **Event Receiver** project item, because you do not want Visual Studio to generate event receiver registration components. The built-in project items enable you to associate event receivers with lists, but they do not enable you to associate event receivers with content types.

- Add a using statement for the **Microsoft.SharePoint** namespace.
- Amend the **VacationRequestEventReceiver** class to inherit from **SPItemEventReceiver**.
- Override the **ItemAdding** method.
- Override the **ItemUpdated** method.
- Save your work and rebuild the solution.

#### ► Task 3: Create the ItemAdding Method

- In the **VacationRequestEventReceiver** class, within the **ItemAdding** method, add code to calculate the duration of the vacation request in days, as an integer value. You may find the following hints and tips useful:
  - a. Use **properties.AfterProperties["StartDate"]** to retrieve the start date provided by the user.
  - b. Use **properties.AfterProperties["\_EndDate"]** to retrieve the end date provided by the user.



**Note:** In practice, you would add logic to exclude non-working days, such as weekends and public holidays, from the calculation of the vacation duration. However, as a proof of concept, a straightforward calculation of the number of days will suffice.

- Add code to retrieve the **SPUser** instance that corresponds to the **Employee** field in the list.



**Note:** Retrieving an **SPUser** instance in an **ItemAdding** method is notoriously tricky, because the **SPFieldUserValue** instance may not have a valid **LookupId** property at this point. One approach is to use the **SPWeb.EnsureUser** method. The following code example illustrates this approach:

```
var web = properties.OpenWeb();
SPUser employee;
var fieldVal = new SPFieldUserValue(web, properties.AfterProperties["Employee"].ToString());
if (fieldVal.LookupId == -1)
{
 employee = web.EnsureUser(web, fieldVal.LookupValue);
}
else
{
 employee = fieldVal.User;
}
```

If you use this approach, be sure to dispose of the **SPWeb** instance properly.

- Add code to retrieve the list item that contains the employee's holiday entitlement from the Vacation Tracker list.



**Note:** You can use the following CAML query to match items based on a user's ID value:

```
<Where>
 <Eq>
 <FieldRef Name=""Employee"" LookupId=""TRUE"">/FieldRef>
 <Value Type=""Integer"">{0}</Value>
 </Eq>
</Where>
```

- Retrieving the **Days Remaining** field value from the Vacation Tracker list item.
- If the number of vacation days requested exceeds the number of days remaining, cancel the event and display an explanatory error message.
- Rebuild the solution and resolve any errors.

#### ► Task 4: Create the ItemUpdated Method

- In the **VacationRequestEventReceiver** class, within the **ItemAdding** method, retrieve the value of the **Vacation Request Status** field.



**Note:** Use **properties.AfterProperties["VacationRequestStatus"]** to get the value of the **Vacation Request Status** field.

- If the vacation request status is anything other than "Approved", take no further action.
- If the vacation request status is "Approved", calculate the duration of the vacation request in days.
- Add code to retrieve the **SPUser** instance that corresponds to the **Employee** field in the list.
- Add code to retrieve the list item that contains the employee's holiday entitlement from the Vacation Tracker list.
- Deduct the length of the vacation from the employee's holiday entitlement, and then update the list item in the Vacation Tracker list.
- Change the vacation request status of the current list item to "Booked".



**Note:** Remember to call the **SPListItem.Update** method after making changes to list item field values.

**Results:** After completing this exercise, you should have created an event receiver assembly.

## Exercise 2: Registering an Event Receiver with a Site Content Type

### Scenario

In this exercise, you will associate your **ItemAdding** and **ItemUpdated** event receivers with the Vacation Request content type. To do this, you will create an empty Site-scoped Feature and add a feature receiver class. Within the feature receiver class, you will add the event receivers to the content type when the Feature is activated, and you will remove the event receivers from the content type when the Feature is deactivated.

The main tasks for this exercise are as follows:

1. Add a Feature and a Feature Receiver Class
2. Add Code to Register the ItemAdded Event Receiver When the Feature Is Activated
3. Add Code to Register the ItemUpdated Event Receiver When the Feature Is Activated
4. Add Code to Remove the ItemAdding Event Receiver When the Feature Is Deactivated
5. Add Code to Remove the ItemUpdated Event Receiver When the Feature Is Deactivated
6. Test the ItemAdding Event Receiver
7. Test the ItemUpdated Event Receiver

#### ► Task 1: Add a Feature and a Feature Receiver Class

- Rename the project Feature from **Feature1** to **VacationEntitlementChecker**.
- Set the Feature title to **Vacation Entitlement Checker**.
- Set the Feature description to **Checks vacation requests against remaining vacation entitlement, and updates remaining vacation entitlement when a vacation is booked**.
- Set the Feature scope to **Site**.
- Add an event receiver to the Feature.
- In the **VacationEntitlementCheckerEventReceiver** class, uncomment the **FeatureActivated** and **FeatureDeactivating** method.

- Save your work and rebuild the solution.

### ► Task 2: Add Code to Register the ItemAdded Event Receiver When the Feature Is Activated

- In the **VacationEntitlementCheckerEventReceiver** class, add the following class constant:

```
const string itemAddingName = "Vacation Request ItemAdding";
```



**Note:** You will use this constant to provide a unique name for the **ItemAdding** event receiver.

- In the **FeatureActivated** method, retrieve the **Vacation Request** content type from the root web of the site collection.
- Add a new event receiver to the event receivers collection of the Vacation Request content type.
- Set the **Assembly** property of the new event receiver to the strong name of the assembly for the current solution.



**Note:** You can use the following Windows PowerShell commands to retrieve the strong name for the assembly:

```
$assembly = "E:\Labfiles\Starter\VacationRequestsEventReceiver\
VacationRequestsEventReceiver\bin\Debug\VacationRequestsEventReceiver.dll"
[System.Reflection.AssemblyName]::GetAssemblyName($assembly).FullName
```

- Set the **Class** property of the new event receiver to the fully-qualified class name of the event receiver class you created in the previous exercise (for example, **VacationRequestsEventReceiver.VacationRequestEventReceiver**).
- Set the **Type** property of the new event receiver to **SPEventReceiverType.ItemAdding**.
- Set the **Name** property of the new event receiver to the **itemAddingName** constant.
- Set the **SequenceNumber** property of the new event receiver to an integer value of 10,000 or more.
- Call the **Update** method on the new event receiver.
- Call the **Update** method on the content type. Make sure that your changes are cascaded to list content types.
- Rebuild the solution and resolve any errors.

### ► Task 3: Add Code to Register the ItemUpdated Event Receiver When the Feature Is Activated

- In the **VacationEntitlementCheckerEventReceiver** class, add the following class constant:

```
const string itemUpdatedName = "Vacation Request ItemUpdated";
```



**Note:** You will use this constant to provide a unique name for the **ItemUpdated** event receiver.

- In the **FeatureActivated** method, retrieve the **Vacation Request** content type from the root web of the site collection.



- Add a new event receiver to the event receivers collection of the Vacation Request content type.
- Set the **Assembly** property of the new event receiver to the strong name of the assembly for the current solution.
- Set the **Class** property of the new event receiver to the fully-qualified class name of event receiver class you created in the previous exercise.
- Set the **Type** property of the new event receiver to **SPEventReceiverType.ItemUpdated**.
- Set the **Name** property of the new event receiver to the **itemUpdatedName** constant.
- Set the **SequenceNumber** property of the new event receiver to an integer value of 10,000 or more.
- Call the **Update** method on the new event receiver.
- Call the **Update** method on the content type. Ensure that your changes are cascaded to list content types.
- Rebuild the solution and resolve any errors.

#### ► Task 4: Add Code to Remove the ItemAdding Event Receiver When the Feature Is Deactivated

- In the **FeatureDeactivating** method, retrieve the **Vacation Request** content type from the root web of the site collection.
- Remove the **ItemAdding** event receiver from the event receivers collection of the content type.



**Note:** Enumerate the event receivers collection and look for an event receiver with a matching **Name** value.

- Update the content type, and ensure that changes are cascaded to list content types.

#### ► Task 5: Add Code to Remove the ItemUpdated Event Receiver When the Feature Is Deactivated

- In the **FeatureDeactivating** method, remove the **ItemUpdated** event receiver from the event receivers collection of the Vacation Requests content type.
- Update the content type, and ensure that changes are cascaded to list content types.
- Rebuild the solution and resolve any errors.

#### ► Task 6: Test the ItemAdding Event Receiver

- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, after the page finishes loading, on the Quick Launch navigation menu, click **Vacation Tracker**.
- Notice that the user **Ankur Chavda** has eight days of vacation remaining.
- On the Quick Launch navigation menu, click **Vacation Requests**.
- On the **Vacation Requests** page, click **New Item**.
- In the **Title** box, type **Two weeks in the Caribbean!**
- In the **Employee** box, type **Ankur Chavda**, and then press Enter.

- In the **Department** drop-down list, click **Legal**.
  - In the **Line Manager** box, type **Dominik Dubicki**, and then press Enter.
  - In the **Start Date** box, select a date within the next 12 months.
  - In the **End Date** box, select a date two weeks after the start date.
  - Click **Save**.
  - Verify that you are prevented from submitting the vacation request with the message **User Ankur Chavda only has 8 vacation days remaining**.
  - In the **Title** box, type **One week in the Caribbean!**
  - In the **End Date** box, select a date one week after the start date, and then click **Save**.
  - Verify that the vacation request was created successfully.
- **Task 7: Test the ItemUpdated Event Receiver**
- In the **Vacation Requests** list, select the **One week in the Caribbean** row.
  - On the ribbon, on the **ITEMS** tab, click **Edit Item**.
  - In the **Vacation Request Status** drop-down list, click **Approved**, and then click **Save**.
  - Verify that the vacation request status of the list item is changed to **Booked**.
  - On the Quick Launch navigation menu, click **Vacation Tracker**.
  - Verify that the list now shows **Ankur Chavda** as having one day of vacation remaining.
  - Close Internet Explorer.
  - Close Visual Studio.
  - Close Windows Powershell ISE.

**Results:** After completing this exercise, you should have used a feature receiver to add an event receiver to a site content type.

## Module Review and Takeaways

In this module, you learned about how to work with the key building blocks for implementing a taxonomy in SharePoint 2013. You learned about how to create site columns and content types, both declaratively and programmatically. You learned how to work with key taxonomy building blocks from both server-side and client-side code. You also learned about how to work with more advanced features of content types, such as adding workflow associations and event receiver definitions.

### Review Question(s)

**Question:** Contoso uses two lists to manage invoices. The **Invoice** list contains general information, such as the name of the client, the billing address, and the due date. The **Invoice Items** list contains more detailed information about individual invoice lines, such as the product or service and the unit costs. You want to link the two lists. How should you proceed?

**Question:** What are the advantages of creating a content type programmatically, rather than declaratively?

**Question:** You have added a workflow association to a site content type. How do you ensure that the workflow association is also applied to list content types?



# Module 13

## Managing Custom Components and Site Life Cycles

### Contents:

Module Overview	13-1
<b>Lesson 1:</b> Defining Custom Lists	13-2
<b>Lesson 2:</b> Defining Custom Sites	13-7
<b>Lesson 3:</b> Managing SharePoint Sites	13-16
<b>Lab:</b> Managing Custom Components and Site Life Cycles	13-22
Module Review and Takeaways	13-29

## Module Overview

SharePoint uses sites and lists to organize your solution. Planning and designing those components will help to ensure your SharePoint solution meets your business requirements. Often, you will need to create sites or lists that differ from the standard templates provided by SharePoint. This module introduces you to how you can create custom component definitions and templates, which enable you to deploy custom sites, lists and other components across your farm.

### Objectives

After completing this module, you will be able to:

- Define and provision custom lists
- Define and provision custom sites.
- Manage the SharePoint site life cycle.

## Lesson 1

# Defining Custom Lists

Lists are the main component in SharePoint for storing structured data. SharePoint includes out-of-the-box list definitions, but you will often need to customize the fields, forms, or views to more accurately meet your needs. In this lesson, you will learn about how you can create list definitions to model a list, and then how you can deploy instances of out-of-the-box, or custom list definitions either by using declarative markup in a Feature or by using code.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe the purpose of list definitions.
- Create a list definition.
- Deploy a list definition.
- Provision lists.

### Introduction to List Definitions

SharePoint contains several list definitions you can use in your SharePoint solutions, but you will often want to develop a custom list, with fields that meet your specific requirements. Although you can add a custom list on an ad-hoc basis by using the SharePoint interface, you should usually create a SharePoint list by using a template. A list definition can contain content types, fields, views, forms, toolbars, and a default description.

You create a list definition by using declarative markup and a Feature, although you may include other files; for example, you can include custom forms as part of the list definition. After you define and deploy a list definition, you can create an instance of that list either by using a Feature or by using the SharePoint object model.

- Define list definition:
  - Content types
  - Fields
  - Views
  - Forms
- Create list instance:
  - Declarative (Feature)
  - Programmatic (SharePoint object model)

### Developing List Definitions

To develop a new list definition, you must create a new Feature. To create a simple list definition, your Feature must contain the following items:

- *A Feature.xml file.* The Feature.xml file must be in the root of the Feature folder; it should contain the Feature definition and details of the files, including the folder structure, that are used by the Feature.
- *An element manifest file.* The element manifest file can have any name, providing it has an

- List template Feature:
    - Feature.xml file (in the root of the Feature folder)
    - Manifest.xml file (optionally in a subfolder, can have any name, but must be referenced in Feature.xml file)
    - ListName folder (determined by the value of the **Name** attribute of the **ListTemplate** element in the manifest)
      - Schema.xml file
- |                |                |
|----------------|----------------|
| • Manifest.xml | • Schema.xml   |
| <Elements>     | <List>         |
| <ListTemplate> | <MetaData>     |
|                | <ContentTypes> |
|                | <Fields>       |
|                | <Views>        |
|                | <Forms>        |

".xml" file name extension. You must include the element manifest file in the Feature.xml file by using an **ElementManifest** element. The element manifest file should contain a **ListTemplate** element. This element defines the properties of the list definition—for example, the name, display name, and ID value—but does not contain any details of the forms, field, views, and so on that are included in the list definition.

- *A subfolder that contains a file named Schema.xml.* The name of the subfolder is determined by the value of the **Name** attribute of the **ListTemplate** element in your element manifest file. If the name of the subfolder and the value of the **Name** attribute do not match, your list definition will not work correctly. The Schema.xml file should contain Collaborative Application Markup Language (CAML), with a root **List** element, with child elements to add fields, views, content types, and so on to the list definition. The Schema.xml file should be included in the Feature.xml file by using an **ElementFile** element, rather than an **ElementManifest** element.

The following code example shows an example of an element manifest file for a list definition.

### ListTemplate Element Manifest Code Example

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <ListTemplate
 Name="AssistantList {Name for the list definition; the name of the sub-folder for the schema.xml
file}"
 Type="10001 {Unique identifier for the list definition, use values above 10000 to avoid clashes}"
 BaseType="0 {Identifier of the list on which this list is based, 0 for generic list}"
 OnQuickLaunch="TRUE {Specifies whether list instances should appear on the quick launch by
default}"
 DisplayName="Assistants List {Display name for the list definition, displayed when users create
new lists}"
 Description="List to store details of assistants {Description for the list definition}"
 </ListTemplate>
</Elements>
```



**Additional Reading:** For more information about defining a list definition in a Feature element manifest file, see *ListTemplate Element (List Template)* at <http://go.microsoft.com/fwlink/?LinkID=307083>.

The following code example shows the key elements of a list schema in a Schema.xml file.

### List Schema Code Example

```
<List xmlns="http://schemas.microsoft.com/sharepoint" ... >
 <Metadata>
 <ContentTypes>
 <ContentType ID="..." Name="..." ... </ContentType>
 <ContentTypeRef ID="..." ... />
 </ContentTypes>
 <Fields>
 <Field ID="..." Name="..." ... />
 <FieldRef ID="..." Name="..." />
 </Fields>
 <Views>
 <View ... > ... </View>
 </Views>
 <Forms>
 <Form Type="DisplayForm" Url="DispForm.aspx" ... />
 </Forms>
 </Metadata>
</List>
```



**Additional Reading:** For more information about defining a list schema, see *List Schema* at <http://go.microsoft.com/fwlink/?LinkID=307084>.

## The Visual Studio List Designer

Although you can develop a list definition manually, it is usually significantly easier and faster to use the list designer in Visual Studio. The list designer enables you to select items from lists instead of writing XML code. This is usually significantly easier because you do not need to look up the GUIDs of content types or columns that you want to include in your list. It is also usually faster because most developers can select options from a list faster than write XML code. Because the XML code is automatically generated, the risk of an error is also reduced.

- Manage:
  - Columns
  - Content types
  - Views
  - Title
  - URL
  - Description
- Faster (select from lists, no need to lookup GUIDs)
- Easier (no need to write XML)
- Reduced risk (less chance of typographical error)

To use the Visual Studio list designer, you must add a list to a Visual Studio SharePoint solution. Visual Studio will ask you what type of list you want to create. You can choose to add a list instance based on an existing list definition. In this case, SharePoint will simply add the files necessary to deploy a list instance. You can choose to create a customizable list definition and list instance. In this case, Visual Studio will also add all of the necessary files to define a list, including the manifest file, and schema file. In addition, regardless of your selection, if your solution does not already contain a Feature, Visual Studio will add a Feature to deploy the list instance (and template if appropriate).

After you add a list definition to your solution, you can use the list designer to:

- Add columns to the list definition.
- Manage the content types associated with the list definition (you can select either content types that you have added to the same Visual Studio solution or content types that are already deployed to the development SharePoint site, you can).
- Define views for the list definition, by simply selecting from the list of available columns to include in each view.
- Specify the default list tile, URL, and description, as well as whether the list should be visible on the Quick Launch menu, and whether the list definition should be visible in the browser.

Visual Studio will automatically update the Schema.xml, manifest file, and Feature with the changes you make. This significantly reduces development time and means that you only need to edit the XML files if you need to make more advanced customizations, for example to configure custom forms.



## Discussion Question

The instructor will now lead a discussion around the question: When would you choose to create a list definition?

- When would you choose to create a list definition?

## Provisioning Lists

In addition to deploying lists by using the SharePoint interface, you can also deploy list instances by using either a Feature or the SharePoint object model.

If you choose to add a new list instance by using a Feature, you must create an Elements.xml file that contains a **ListInstance** element. You should provide a title, description, URL, and template type. The value for the **TemplateType** attribute should be the value of the Type attribute of the **ListTemplate** element in the list definition deployment Feature. Standard list definitions provided out-of-the-box with SharePoint are installed by using Features. You can find the ID of built-in lists by viewing the XML relevant Manifest.xml file. If you use Visual Studio to create a list instance Feature, you can select from a list of installed list definitions (including custom list definitions) that will populate the **TemplateType** field automatically.

The following code example shows how to create a list instance by using a Feature.

### List Instance by Using a Feature Code Example

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <ListInstance Title="Products" TemplateType="10001"
 Description="A list to store product information."
 Url="Lists/Products">
 </ListInstance>
</Elements>
```

Alternatively, you can add a new list by writing code. To add a new list, you use the **Add** method on the **Lists** collection property of the **SPWeb** class. When you write code to add a list, for custom list definition, use an instance of the **SPListTemplate** class to identify the list definition, or for out-of-the-box template, you can use the **SPListTemplateType** enumeration to select a list definition.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <ListInstance Title="Products" TemplateType="10001"
 Description="A list to store product information."
 Url="Lists/Products">
 </ListInstance>
</Elements>

using(SPWeb web = site.RootWeb)
{
 SPListTemplate template =
 web.ListTemplates["ProductListTemplate"];
 Guid productsList = web.Lists.Add("Products",
 "A list to store product information.", template);
 Guid announcementsList = web.Lists.Add("Announcements",
 "A list to store announcements.",
 SPListTemplateType.Announcements);
 web.Update();
}
```

The following code example shows how to create a list instance by using code.

#### List Instance by Using Code Code Example

```
using(SPWeb web = site.RootWeb)
{
 // Obtain a reference to the list definition by using the ListTemplates collection, and by
 // specifying the
 // template name.
 SPListTemplate template = web.ListsTemplates["ProductListTemplate"];
 // Create a new instance of the list definition.
 Guid productsList = web.Lists.Add("Products", "A list to store product information.",
 template);

 // Create a new instance of the Announcements list definition by using the SPListTemplateType
 // enumeration to identify the list definition.
 Guid announcementsList = web.Lists.Add("Announcements", "A list to store announcements.",
 SPListTemplateType.Announcements);

 // Call the Update method on the SPWeb instance.
 web.Update();
}
```

## Lesson 2

# Defining Custom Sites

SharePoint includes site definitions out-of-the-box. When you create a new site, you can choose a site definition to apply. The site definition defines the initial pages, lists, libraries, and other items included in your site. You must create a site definition either by using an XML file to define the site's contents or by writing custom code to provision the site. Web templates provide an alternative to site definitions, and can be used to perform many of the same tasks. In this lesson, you will learn about site definitions and web templates, how to create, deploy, and use them.

### Lesson Objectives

After completing this lesson, you will be able to:

- Explain the differences between a site definition and a web template.
- Develop a custom site definition.
- Deploy a site definition.
- Write code to provision a SharePoint site programmatically.
- Provision sites by using out-of-the-box and custom site definitions, and web templates.
- Describe the purpose of the Webtemp\*.xml and Onet.xml files.
- Explain Feature stapling.

### Introduction to Site Definitions and Web Templates

Site definitions provide the most powerful mechanism for developing custom sites. With a custom site definition, you can customize the pages, lists, and Features included in a site, and then deploy multiple copies within your organization. In addition, you can include content in a site definition.

Site definitions do not have any dependencies on other site definitions and define the entire contents of a site at the point of creation. You can fully customize a site definition; however, you must either create a site definition by using declarative markup, or by using code. This can make developing a site definition too time consuming for some projects. Another issue with site definitions is that you must deploy them to the file system on the web server. This means that they cannot be used in sandboxed solutions, or with SharePoint Online; however, they are cached by the IIS worker process, which results in better performance compared to alternatives.

Web templates provide an alternative to a site definition, and are simpler to develop. You create a web template by saving an existing site as a template. You can save a site as a web template programmatically, or by using the SharePoint UI. A significant difference when using a web template is that a web template does not define everything that is required to recreate a site; instead, it stores only differences between a base site definition and the site when it was saved. To create a web template, you can use the **SaveAsTemplate** method of the **SPWeb** class.

- Site definition:
  - Deploy with a farm solution
  - Not dependent on any other component
  - Time consuming to develop
  - Cached by the IIS worker process for performance
- Web template:
  - Deploy with a sandboxed or farm solution
  - Dependent on source site definition
  - Fast to develop – SPWeb.SaveAsTemplate
  - Not cached for performance

If you create a site by using a web template, you must have the corresponding site definition installed in your SharePoint farm. If the original site definition is missing, you cannot use the web template. Another difference is how web templates are stored and cached. Site definitions are stored on the file system, and are cached by the IIS worker process; whereas, web templates are stored in the content database. When you use a web template, it must be retrieved from the database. This is less efficient than a site definition.



**Additional Reading:** For more information about saving an existing site as a template, see *SPWeb.SaveAsTemplate method* at <http://go.microsoft.com/fwlink/?LinkID=307085>.

## Developing a Site Definition Declaratively

A declarative site definition contains two main components, a `Webtemp*.xml` file and a template folder that contains an `Onet.xml` file. Although you may (and usually will) include other files with your site definition, all declarative site definitions must contain these files.

### Webtemp\*.xml file

SharePoint uses `Webtemp*.xml` files to identify site definitions that are available on the farm. You can include multiple site definitions in a `Webtemp*.xml` file, and each site definition can include multiple configurations. `Webtemp*.xml` files are stored in the `15\TEMPLATE\LCID\XML` folder (where *LCID* is the locale ID; for example, **1033** for U.S. English). Every `Webtemp*.xml` file must have a unique name, and you should neither overwrite nor edit any of the out-of-the-box `Webtemp*.xml` files. You can add any value in place of the `*` in the file name, but the file name must start with **webtemp**; for example, **webtemp123.xml** and **webtemp\_ab.xml** are both valid file names. There is no link between the name of the `Webtemp*.xml` file and the corresponding site definition.

The `Webtemp*.xml` file contains a root **Templates** element, which contains a set of **Template** elements. You should add one **Template** element per site definition. The **Template** element has a **Name** attribute and an **ID** attribute. Both of these attributes must be unique in your farm. The **ID** should be an integer; you should use a number above 10000 to avoid values that are used by SharePoint Foundation site definitions. The value of the **Name** attribute must be a string value, and you must use this as the name of the folder that contains the `Onet.xml` file.

A **Template** element should contain one or more **Configuration** elements. You use the properties of the configuration element to determine how the site definition appears in the user interface. When you create an instance of a site definition, you must make reference to both the **Name** of the template (specified at the **Template** element level) and the **ID** of the configuration (specified at the **Configuration** element level). The **ID** of the configuration is an integer, and each **Configuration** element must have a unique configuration value; however, these do not need to be unique across the template; usually, your first configuration will have the ID **0**.

- `Webtemp*.xml`
  - Template and configuration definitions
- `Onet.xml`
  - Site content and structure
- Supporting files (pages, images, and so on)
- Assemblies

The following code example shows a Webtemp\*.xml file.

### Webtemp\*.xml File Code Example

```
<Templates>
 <Template Name="ContosoSiteDefinition" ID="12000">
 <Configuration ID="0"
 Title="Contoso Support Team Site {Site definition title used in the SharePoint interface}"
 Hidden="FALSE {Used to determine whether to display the definition in the interface}"
 ImageUrl="{Server relative URL for a thumbnail image which represents the site definition}"
 Description="Support team site definition description {Description of the definition for
the interface}"
 DisplayCategory="Contoso Sites {Category in which this site definition should be included}"
 </Template>
 </Template>
```



**Additional Reading:** For more information about the Webtemp\*.xml file, see *Understanding WebTemp\*.xml Files* at <http://go.microsoft.com/fwlink/?LinkID=307086>.

### Template folder and Onet.xml file

Although the Webtemp\*.xml file is important for SharePoint to identify available site definitions, it does not contain any detail of what each site should include. You must create a folder in the **15\TEMPLATES\SiteTemplates** folder to store the files used by your site definition. You must name the folder the same as the value of the **Name** attribute of the **Template** element. You should add any files you need for your site definition to this folder; for example, you should include any ".aspx" webpages and any images to this folder. You can create subfolders and organize your file as appropriate.

The folder must contain a subfolder named **XML**, which must contain an XML file named **Onet.xml**. This file defines the Features, lists, pages, and so on, that are included in each configuration of your site definition. Some settings defined in the site definition are common across all configurations of the site definition, whereas others are unique to individual configurations. By creating multiple configurations of a site definition, you can create site definitions that are easier to maintain than developing a new site definition for each configuration (for example, if you need to specify an email footer, you only need to specify this once rather than multiple times), but with multiple site definitions, you have more control over each site definition.

You define configurations by including **Configuration** elements in the Onet.xml file. The **ID** of each of these configurations must match the **ID** of a configuration specified in the Webtemp\*.xml file.



**Additional Reading:** For information about when to create multiple site definitions, and when a single site definition with multiple configurations would be more appropriate, see *Deciding Between Multiple Definitions or Multiple Configurations* at <http://go.microsoft.com/fwlink/?LinkID=307087>.

The following code example shows the main components of an Onet.xml file.

### Onet.xml File Code Example

```
<Project xmlns="http://schemas.microsoft.com/sharepoint/" Title="ContosoSiteDefinition">
 <NavBars>
 <!-- Add NavBar and NavBarLink elements here to add navigation bars and links to your
site. -->
 </NavBars>
 <Configurations>
 <!-- Add a Configuration element for each configuration defined in the webtemp*.xml file
here. -->
 <Configuration ID="0" Title="SalesSiteDefintion">
 <Lists>
 <!-- Add lists to include in this configuration here. -->
 </Lists>
 <SiteFeatures>
 <!-- Add site collection Features to include in this configuration here. -->
 </SiteFeatures>
 <WebFeatures>
 <!-- Add site Features to include in this configuration here. -->
 </WebFeatures>
 <Modules>
 <!-- Add modules to include in this configuration here. -->
 </Modules>
 </Configuration>
 </Configurations>
 <Modules>
 <!-- Add Module elements here, for example to include pages or files in your site
definition. -->
 </Modules>
 <Components>
 <!-- Add custom components such as an external security provider here. -->
 </Components>
 <ServerEmailFooter>
 <!-- Specify the server email footer here. -->
 </ServerEmailFooter>
</Project>
```



**Additional Reading:** For more information about developing a custom site definition, see *How To: Create a Custom Site Definition and Configuration* at <http://go.microsoft.com/fwlink/?LinkID=307088>.

### Assemblies

In addition to files in the **SiteTemplates** folder, and the Webtemp\*.xml file, you may need to include assemblies that include compiled code required by your site definition. If you need to include any compiled code, you should deploy the assemblies to the global assembly cache. A common reason to include an assembly with a site definition is to include a class to customize the provisioning process for a site definition. More information about customizing the site provisioning process by using code is included later in this lesson.


## Deploying Site Definitions


You must deploy site definitions to the file system on your SharePoint servers. You must deploy the files on every server; you should use a SharePoint farm solution to make sure your site definition is deployed to every server correctly.

A site definition should be considered in three parts for deployment: the `Webtemp*.xml` file, any assemblies, and then everything else (which will include the `Onet.xml` file). The following list describes how you should deploy each of these three parts:

- *Webtemp\*.xml file.* You must add the `Webtemp*.xml` file to the `15\TEMPLATE\LCID\XML` folder. You must replace `LCID` with the locale ID for your locale. For U.S. English, this is 1033. You should add the files as an additional file, and never alter or overwrite any of the standard `Webtemp*.xml` files. You can add any text after **webtemp** to make your file name unique.
- *Assemblies.* You should deploy your signed compiled assemblies to the global assembly cache.
- *Everything else.* You should add your other files to a folder in the `15\TEMPLATES\SiteTemplates` folder. The name of the folder must match the value of the **Name** attribute of the **Template** element in the `Webtemp*.xml` file. You can include any files needed (such as ".aspx" page files) in this folder. You should add your **onet.xml** file in an **XML** subfolder.

- `Webtemp*.xml`
  - `15\TEMPLATE\LCID\XML`
- **Assemblies**
  - Global assembly cache
- **Everything else** (including `Onet.xml` file):
  - `15\TEMPLATE\SiteTemplates\SiteTemplateName`
  - `15\TEMPLATE\SiteTemplates\SiteTemplateName\XML\onet.xml`

 **Note:** When you add a site definition to a solution in Visual Studio, Visual Studio adds a Feature to the project—you do not need to include this Feature if you are only developing a site definition, so you can remove this from the project. Visual Studio also compiles an assembly whenever you build the project. If you have not added any code files to your solution (and so the assembly is empty), you can prevent Visual Studio from including the assembly in the SharePoint solution by changing the **Include Assembly in Package** project property.

 **Additional Reading:** For more information about locale IDs, see *Locale IDs Assigned by Microsoft* at <http://go.microsoft.com/fwlink/?LinkID=307089>.

## Using Code to Provision a Custom Site

In addition to deploying a custom site definition, you can also deploy code to provision new sites. You can deploy a site by using code in addition to declarative markup, or you can write code to provision a site.

To use code to provision a custom site, you must create a class that inherits from the **SPWebProvisioningProvider** class. You must then override the **Provision** method with your custom logic to provision the new site. The **Provision** method accepts a **SPWebProvisioningProperties** parameter. You can use the properties of this

```
public class ContosoSite : SPWebProvisioningProvider
{
 public override Provision(
 SPWebProvisioningProperties props)
 {
 SPWeb newWeb = props.Web;
 newWeb.ApplyWbTemplate("STS#1");
 string data = props.Data;
 // Add lists, libraries, Features pages etc.
 }
}
<Template Name="ContosoSite" ID="12001">
 <Configuration ID="0" ...
 ProvisionAssembly="Assemblystrong name"
 ProvisionClass="Contoso.ContosoSite"
 ProvisionData="String data" />
</Template>
```

parameter to obtain a reference to the new **SPWeb** object. When you use code to provision a site, the site does not have any initial state. You can use the **ApplyWebTemplate** method of the **SPWeb** object to apply another site template as an initial state, for example the blank site template (STS#0). In addition, you can use the **Data** property of the **SPWebProvisioningProperties** parameter to retrieve a string value that is defined in the site definition configuration (Webtemp\*.xml) file.

The following code example shows how to provision a site by using code.

### Custom Site Provisioning Code Example

```
public class ContosoSite : SPWebProvisioningProvider
{
 public override Provision(SPWebProvisioningProperties props)
 {
 // Obtain a reference to the new SPWeb object.
 SPWeb newWeb = props.Web;

 // Apply the blank site template.
 newWeb.ApplyWbTemplate("STS#1");

 // Retrieve configuration data.
 string data = props.Data;

 // Add lists, libraries, Features, pages, and other components to the site.
 // web.Lists.Add(...);
 // web.Features.Add(...);
 }
}
```

If you develop a custom site provisioning provider, you do not need to deploy a folder to the **SiteTemplates** folder, and you do not need to create an Onet.xml file; however, you must make some changes to the Webtemp\*.xml file. In your Webtemp\*.xml file, for each configuration you must add the **ProvisionAssembly**, **ProvisionClass**, and optionally **ProvisionData** attributes.

The **ProvisionAssembly** attribute should contain the strong name key for your assembly. The **ProvisionClass** attribute should contain the fully qualified class name for the provisioning provider. The **ProvisionData** attribute is optional. If you choose to provide this attribute, the value of this attribute is available in the **Provision** method by using the **Data** property of the parameter. Because this attribute only enables you to specify a single string value, and if you need to pass significant configuration data to your custom provisioning class, you may consider storing a file name in the **ProvisionData** attribute, and then create a corresponding configuration file for use by your custom provisioning class.

You should deploy the assembly containing your custom site provisioning provider to the global assembly cache.



The following code example shows a Webtemp\*.xml file that specifies a custom provisioning provider.

### Custom Provisioning Provider Webtemp\*.xml Code Example

```
<Templates xmlns:ows="Microsoft.SharePoint">
 <Template Name="ContosoSite" ID="12001">
 <Configuration ID="0"
 Title="Contoso Research Site"
 Hidden="FALSE"
 ImageUrl="/_layouts/15/ContosoSiteImage.png"
 Description="This is a custom site template for Contoso research projects."
 DisplayCategory="Contoso Sites"
 ProvisionAssembly=
 "ContosoAssembly, Version=1.0.0.0, Culture=neutral,
 PublicKeyToken=abcde1234567890a"
 ProvisionClass="Contoso.ContosoSite"
 ProvisionData="ResearchConfiguration.xml"
 RootWebOnly="FALSE"
 </Template>
 </Templates>
```

## Demonstration: Examining the Standard Site Definitions

This demonstration reviews the out-of-the-box site definitions. You will review a Webtemp\*.xml file, and then review the corresponding site template files and Onet.xml file. Finally, you will review the Webtemp\*.xml file for the developer site definition that is provisioned by using custom code rather than declarative markup.

### Demonstration Steps

- Start the **20488B-LON-SP-13** virtual machine if it is not already running.
- Log on as **CONTOSO\Administrator**, with the password **Pa\$\$w0rd**.
- On the Start screen, click **Desktop**.
- On the taskbar, click **File Explorer**.
- In File Explorer, browse to the **C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\1033\XML** folder. Explain that all of the **webtemp\*.xml** files are stored in this folder.
- Double-click **WEBTEMP.xml**
- When prompted click **Microsoft Visual Studio 2012**.
- In the **WEBTEMP.xml** file, point out the existing templates and configurations.
- Point out the **STS** template, and the **Team Site** and **Blank Site** configurations.
- Close Visual Studio.
- In File Explorer, browse to the **C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\SiteTemplates** folder.
- Point out the **sts** folder that corresponds with the **STS** template in the **webtemp.xml** file.
- Browse to the **sts** folder.
- Point out the **aspx** pages that are included in the site definition and then browse to the **xml** folder.
- Double-click the **ONET.XML** file.

- Point out the various aspects of the site definition, including the navigation bars, and document templates.
- Point out the **Configurations** element and child **Configuration** elements. Explain how the **ID** attribute of each **Configuration** element corresponds with the configurations in the **WEBTEMP.xml** file.
- Point out how different configurations specify different child items (for example, lists and Features).
- Close Visual Studio.
- In File Explorer, browse to the **C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\1033\XML** folder.
- Double click the **webtempdev.xml** file.
- Point out the **Developer Site** configuration. Explain that there is no corresponding folder in the **SiteTemplates** folder for this configuration because this configuration is provisioned by using code. Point out the **ProvisionAssembly** and **ProvisionClass** attributes, and explain that the corresponding assembly is stored in the global assembly cache.
- Close Visual Studio.

## Feature Stapling

A common requirement is to add a new Feature to a site definition. One option is to create a new site definition that includes the Feature. If an existing site definition already fulfills the remainder of your requirements, developing a new site definition is likely to introduce duplication and introduces risks of errors being introduced if a site definition is updated. In addition, you would need to communicate to your administrators that they should use the new site definition. Your administrators may continue to use the existing or out-of-the-box site definitions despite your best efforts.

Another option is to update an existing site definition to include the Feature; however, there are problems with this solution:

- You must not change out-of-the-box site definitions.
- If you update an existing custom site definition, you may break any existing sites that use that site definition.
- You may need to add a Feature to a site definition only for sites in a particular site collection or web application.

Feature stapling provides an alternative approach. You can use Feature stapling to add a Feature to an existing site definition. To add a Feature (the stapled Feature) to an existing site definition, you must create a new Feature (the stapler Feature).

The stapler Feature should contain an **Elements.xml** file that contains a **FeatureSiteTemplateAssociation** element. The **FeatureSiteTemplateAssociation** element has two attributes that you must specify, the **Id** attribute, which should contain the ID of the stapled Feature, and the **TemplateName** attribute, which should contain the name and configuration of the site definition to which the stapled Feature should be

```

• Add a Feature to an existing site definition:
 • Out-of-the-box site definitions
 • Custom site definitions that are in use

• Stapled Feature

• Stapler Feature

<Elements ... >
 <FeatureSiteTemplateAssociation
 ID="StapledFeatureId"
 TemplateName="SiteDefinition#ConfigurationId"
 />
</Elements>

```

added in the format *SiteName#ConfigurationId*. To staple more than one Feature, to a site definition, or to staple the Feature to more than one site definition, you can add additional **FeatureSiteTemplateAssociation** elements.

The following code example shows how to staple a Feature to the Team Site site definition.

### Feature Stapling Code Example

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <FeatureSiteTemplateAssociation
 Id="3E00ABF6-1301-4698-9BD1-5500725000F0"
 TemplateName="STS#0"
 </FeatureSiteTemplateAssociation>
</Elements>
```

### Feature stapling scope

A Feature stapler only impacts sites that are created within the scope at which the Feature stapler is activated. This enables you to add a Feature to a site definition selectively. You can choose the **Farm**, **Web Application**, or **Site** scope for the Feature stapler. The stapled Feature should have either the **Site** or **Web** scope.

After you deploy the stapled and stapler Features, SharePoint will add the stapled Feature to any new sites that are created, within the activation scope of the stapler Feature, and that are based on the site definition. SharePoint will not change any existing sites. If you need to add a Feature to existing sites that are based on a particular site definition, you must either: write code, use Windows PowerShell, or use the SharePoint interface, to add the new Feature to those sites.

### Discussion Question

The instructor will now lead a discussion around the question: When would you choose to create a site definition, a web template, or use Feature stapling?

- When would you choose to create a site definition, a web template, or use Feature stapling?

## Lesson 3

# Managing SharePoint Sites

Typically, SharePoint solutions include a hierarchy of sites. Each of these sites has a life cycle, and you can control or handle the life cycle events. This lesson introduces you to the site hierarchy, how to control the site life cycle, and events you can handle as part of that life cycle.

### Lesson Objectives

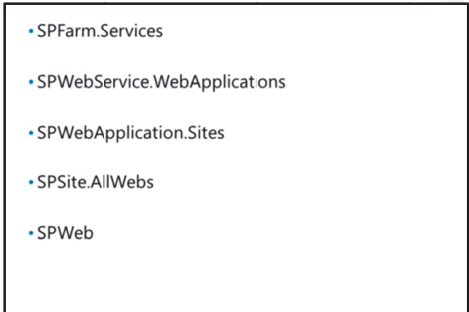
After completing this lesson, you will be able to:

- Describe the site hierarchy.
- Create new sites and site collections.
- Explain the difference between path-based and host-named site collections.
- Develop code to back up and restore site collections.
- Manage the site life cycle.
- Upgrade SharePoint 2010 sites.

### Understanding the Site Hierarchy

All SharePoint sites form part of the SharePoint site hierarchy. If you start with the root of the hierarchy, you can obtain references to any site in the farm by using collection classes that expose each item in the hierarchy:

1. The **SPFarm** class represents the farm and is the root of the hierarchy. The **SPFarm** class exposes the **Services** property of type **SPServiceCollection**.
2. The **SPService** class represents services within the farm. The most common services are the content service and the administration service, both represented by the **SPWebService** class (the **SPWebService** class derives from the **SPService** class). The **SPWebService** class exposes the **WebApplications** property of type **SPWebApplicationCollection**.
3. The **SPWebApplication** class represents web applications. A typical SharePoint farm will have many web applications. This will include web applications for the Central Administration site, for SharePoint sites, and for service applications. The **SPWebApplication** class exposes the **Sites** property of type **SPSiteCollection**.
4. The **SPSite** class represents site collections. The number of site collections in your farm will depend on your deployment requirements. The **SPSite** class exposes the **AllWebs** property of type **SPWebCollection**.
5. The **SPWeb** class represents sites. Like site collections, the number of sites in your farm will depend on your deployment.



```

• SPFarm.Services
• SPWebService.WebApplications
• SPWebApplication.Sites
• SPSite.AllWebs
• SPWeb

```

Although the **SPWeb** class is the bottom component in the site hierarchy, it does not represent the bottom of the SharePoint hierarchy. For example, an **SPWeb** instance contains a collection of lists, which in turn contain a collection of list items, each with a collection of fields, and a collection of attachments.

## Provisioning Sites

You can provision sites either by using Windows PowerShell, by using the SharePoint object model, or by using the SharePoint user interface. You must provision all sites in a site collection, which must exist in a web application. You can obtain a reference to a web application in code by using the static **Lookup** method of the **SPWebApplication** class, and specify a valid **Uri** object as the parameter. If you need to obtain a reference to a web application on a local port, that does not specify a host header, you can use **localhost** as the hostname to obtain the reference. If you use **localhost** instead of the machine name, your code will work on all servers in your server farm, whereas if you specify a server name, you will need to either look up the server name each time, or recompile your code for each server on which it might run.

- SharePoint object model:
  - SPSite site = webApplication.Sites.Add(...);
  - SPWeb web = site.AllWebs.Add(...);
- Windows PowerShell:
  - New-SPSite
  - New-SPWeb
- SharePoint user interface

To provision a new site collection, you should use the **Add** method of the **SPSiteCollection** class. You can obtain a reference to an instance of the **SPSiteCollection** class by using the **Sites** property of an instance of the **SPWebApplication** class. Alternatively, you can use the **New-SPSite** cmdlet in Windows PowerShell to create a new site collection.



**Additional Reading:** For more information about how to create a site collection by using Windows PowerShell, see *New-SPSite* at <http://go.microsoft.com/fwlink/?LinkID=307090>.

To provision a new site, you should use the **Add** method of the **SPWebCollection** class. You can obtain a reference to an instance of the **SPWebCollection** by using the **AllWebs** property of an instance of the **SPSite** class. You can obtain a reference to an instance of the **SPSite** class by using the class constructor, specifying the URL of the root of the site collection as the parameter. Alternatively, you can use the **New-SPWeb** cmdlet in Windows PowerShell to create a new site.



**Additional Reading:** For more information about how to create a site by using Windows PowerShell, see *New-SPWeb* at <http://go.microsoft.com/fwlink/?LinkID=307091>.

The following code example shows how to create a new site collection and a new site by using the SharePoint object model.

### New Site Collection and Site Object Model Code Example

```
// Obtain a reference to the parent web application.
SPWebApplication webApplication = SPWebApplication.Lookup(new
Uri("http://sharepoint.contoso.com"));

// Create a new site collection at the root of the web application.
SPSite rootSite = webApplication.Sites.Add("", @"CONTOSO\Administrator",
"administrator@contoso.com");

// Create a new site collection by using a server relative path. Use null instead of providing an
email address.
SPSite site = webApplication.Sites.Add("Sites/SubSiteCollection", @"CONTOSO\Administrator",
null);

// Create a new site at a specified URL
SPWeb web = rootSite.AllWebs.Add("Sites/SalesSite");

// Create a new site by specifying a template and other initial properties.
SPWeb teamWeb = rootSite.AllWebs.Add(
 "TeamSite", // Site collection relative URL
 "Team Site", // Site title
 "Team site description", // Site description
 1033, // Locale ID
 "STS#0", // Template ID and configuration
 false, // Boolean to indicate if this site should have unique permissions from the parent
site collection.
 false); // Boolean to indicate is SharePoint should covert the existing site if it already
exists
```

## Host-named Site Collections

When you create a web application, you can choose to specify a host header. If you specify a host header, SharePoint will create a website in IIS, with a binding to match that host header. IIS routes requests that include that host header to that web application. If you do not specify a host header, the web application will handle all requests that are on the correct port and do not have a host header that matches another web application.

If you choose to create a web application with a host header, you can simply use a domain name or subdomains to access your site collections. For example, instead of `http://sharepoint.contoso.com/sites/sales`, you could create a web application with the host header `http://sales.contoso.com`. All sites within the sales site collection would then exist within the `sales.contoso.com` domain (these are known as path-based sites). In addition, you can create path-based site collections under the root site on any managed path. This provides flexibility for site collection URLs, but if you need to create several site collections, this may decrease performance and add complexity for administrators, because every site collection requires its own web application, which in turn results in an extra process running on the server.

With SharePoint 2013, you can create site collections with host names. In this situation, you create a single web application without specifying a host header. Typically, this will respond to requests on port 80. With

- Web application A (port 80):
  - `http://sales.contoso.com` (site collection)
  - `http://sales.contoso.com/sites/subsite` (site)
  - `http://sales.contoso.com/sites/subsite2` (site)
  - `http://research.contoso.com` (site collection)
  - `http://sales.contoso.com/sites/sitecollection` (site collection)
- Web application 2 (IIS host header – `sharepoint.contoso.com`):
  - `http://sharepoint.contoso.com` (site collection)
  - `http://sharepoint.contoso.com/sites/sitecollection` (site collection)
  - `http://sharepoint.contoso.com/site/subsite` (site)

this approach, IIS routes all requests to the same web application, and SharePoint uses the host header to determine which site collection to process. Host-named site collections are the recommended approach for creating site collections; however, under some circumstances, they are not viable or would add significant complexity. For example, corporate security policies or data protection policies may require that site collections run in separate web applications. In these circumstances, a web application with a host header may be a better selection.

### Creating host-named site collections

You cannot create a host-named site collection by using the SharePoint user interface; you must use the object model, or Windows PowerShell. First, you must create a web application to host the site collections. When you create the web application, you should not provide a host header. When you create a new site collection, you must use an overload of the **Add** method that includes the **useHostHeaderAsSiteName** parameter, and you must supply the value **true** to identify that the URL you have provided includes a host name. If you use Windows PowerShell to create a new host-named site collection, you should specify the **HostHeaderWebApplication** parameter to achieve the same result.

The following code example shows how to use code to create a new host-named site collection.

#### Create Host-named Site Collection CodeExample

```
// Obtain a reference to the web application.
// Use localhost as the web application does not have a host header.
SPWebApplication webApplication = SPWebApplication.Lookup(new Uri("http://localhost"));

// Add a new site by specifying true for the useHostHeaderAsSiteName parameter of the Add method
// overload.
webApplication.Sites.Add(
 "http://team.contoso.com", // URL
 "Team Site", // Site title
 "This is a team site.", // Site description
 1033, // Locale ID
 "STS#0", // Site definition ID
 @"CONTOSO\Administrator", // Primary site collection administration user name
 "Administrator", // Primary site administrator display name
 "administrator@contoso.com", // Primary site administrator email address
 null, // Secondary site collection administrator user name (null to only have a single site
 collection
 // administrator)
 null, // Secondary site collection administrator display name
 null, // Secondary site collection administrator email address
 true); // True if this is a host-named site collection, false if this is a path based site
 collection
```



**Additional Reading:** For more information and considerations for when to use host-named site collections, see *Host-names site collection architecture and deployment (SharePoint 2013)* at <http://go.microsoft.com/fwlink/?LinkID=307092>.

## Backing-up Site Collections

Backups are usually the domain of IT professionals, and you should always have a plan to recover the entire farm in the event of a disaster. SharePoint provides out-of-the-box functionality to back up the farm; however, you may want to extend that functionality, or you may need to back up components before making changes, for example if there is a risk of failure, or if you need to maintain the data for archival.

The SharePoint object model includes the **Backup** and **Restore** methods of the **SPSiteCollection** class. When you invoke these methods, you must provide the URL of the site collection that you are either backing up or restoring, the file name and path for the backup on the server file system, and a Boolean indicating whether the process should overwrite either the existing file for a backup or the existing site collection for a restore.

The following code example shows how to backup and restore a site collection.

### Backup and Restore Site Collection Code Example

```
using(SPSite site = new SPSite("http://sales.contoso.com"))
{
 // Obtain a reference to the SPSiteCollection which contains this SPSite.
 // The SPSiteCollection is exposed as a property of the current SPSite instance's parent web
 application.
 SPSitemCollection siteCollection = site.WebApplication.Sites;

 // Backup the site collection.
 siteCollection.Backup(site.Url, @"C:\Backups\SiteBackup.backup", false);

 // Restore the site collection (overwrite the existing site collection).
 siteCollection.Restore(site.Url, @"C:\Backups\SiteBackup.backup", true);
}
```

```
using(SPSite site = new SPSite("http://sales.contoso.com"))
{
 // Obtain a reference to the SPSiteCollection which contains
 // this SPSite. The SPSiteCollection is exposed as a property
 // of the current SPSite instance's parent web application.
 SPSitemCollection siteCollection = site.WebApplication.Sites;

 // Backup the site collection.
 siteCollection.Backup(site.Url,
 @"C:\Backups\SiteBackup.backup", false);

 // Restore the site collection
 // (overwrite the existing site collection).
 siteCollection.Restore(site.Url,
 @"C:\Backups\SiteBackup.backup", true);
}
```

## Managing the Site Life Cycle

A site raises a number of events that you can handle during the site's life cycle. You can handle these events to customize the site, or the site's life cycle. For example, you might set security permissions when a site is created, or you might save a backup before a site or site collection is deleted. Events you can handle include:

- **WebAdding.** Occurs when a new site is being created.
- **WebProvisioned.** Occurs after a new site has been created.
- **WebMoving.** Occurs before a site moves.
- **WebMoved.** Occurs after a site moves.
- **WebDeleting.** Occurs before a site is deleted.

- Life cycle events:
  - WebAdding
  - WebProvisioned
  - WebMoving
  - WebMoved
  - WebDeleting
  - WebDeleted
  - SiteDeleted
- Identify inactive site collections:
  - SPSite.LastContentModifiedDate
- Delete sites and site collections:
  - SPWeb.Delete()
  - SPSite.Delete()



- **WebDeleted.** Occurs after a site is deleted.
- **SiteDeleted.** Occurs after a site collection is deleted.

### Inactive sites and site deletion

In addition to handling life cycle events, you may want to also alter the site life cycle. A common requirement is to delete site collections that are inactive. To identify inactive site collections, you can use the **LastContentModifiedDate** property of the **SPSite** class. This property returns a **DateTime** object in UTC format identifying the last time content on the site collection was modified. You can compare this against the current date to establish how long has passed since the last change, and use your logic to determine whether the site collection is inactive.

If you decide that you want to delete a site or site collection, both the **SPWeb** and **SPSite** classes expose a **Delete** method. If you call the **Delete** method, you should still dispose the object reference after you delete the object. If you call the **Delete** method on a site that is the root site of the site collection, the **Delete** method will throw an exception. You must delete the site collection to delete the root site.

The following code example shows how to check whether a site collection is inactive, and if it is inactive, delete the site collection.

#### Inactive Site Collection Detection and Deletion Code Example

```
using(SPSite site = new SPSite("http://inactive.contoso.com"))
{
 int daysSinceLastChange = (DateTime.UtcNow - site.LastContentModifiedDate).Days;
 if(daysSinceLastChange > 60)
 {
 site.Delete();
 }
}
```

## Upgrading SharePoint 2010 Sites

SharePoint 2013 enables you to run site collections in SharePoint 2010 mode. This enables you to make any necessary updates, such as for any custom solutions, before you upgrade the site. You may also want to delay upgrading the site for business reasons. You can force the upgrade by using either the object model or Windows PowerShell.

To force an upgrade by using the object model, you should use the **Upgrade** method of the **SPSite** class. You can then query the properties of the **UpgradeInfo** property of the **SPSite** class to check the status of the upgrade.

To force an upgrade by using Windows PowerShell, you should use the **Upgrade-SPSite** cmdlet. You must specify the **VersionUpgrade** parameter to indicate that this is a SharePoint version upgrade rather than an incremental upgrade between SharePoint builds.

- Sites in SharePoint 2010 mode and SharePoint 2013 mode can coexist
- Delay upgrade of individual site collections based on business or technical requirements
- Start upgrade:
  - `SPSite.Upgrade()`
  - `Upgrade-SPSite`

# Lab: Managing Custom Components and Site Life Cycles

## Scenario

Most products sold by Contoso have a dedicated sales team. Each sales team requires site-to-store sales documents and to track sales leads. Sales documents fall into two categories: documents created to support a specific sales opportunity, which may contain customer specific information, and generic documents that can be used for any sales opportunity. To make sure that documents created for a specific customer are not given to another customer, generic documents must be stored in a different document library. Although there are several distinct teams, the needs of the teams are consistent across the company.

You will create a site definition for a generic sales site. The site will contain two document libraries and links to provide sales persons with easy access to Contoso sites.

You will then create a custom list definition for a list that members of the sales teams can use to store details of sales leads. You will then add an instance of the list definition to the sales site definition.

Finally, you will add an event receiver to the site that triggers if the sales site is deleting. You will add code to the event receiver, which creates a backup of the parent site collection before the sales site is deleted.

## Objectives

After completing this lab, you will be able to:

- Create a site definition.
- Create a list definition.
- Handle a site life cycle event and back up a site collection.

Estimated Time: 60 minutes

- Virtual machine: 20488B-LON-SP-13
- User Name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

## Exercise 1: Creating a Site Definition

### Scenario

In this exercise, you will create a site definition. First, you will create a new farm solution, and then you will add a site definition to the farm solution. You will then add two document libraries to the site definition, and add custom navigation to include links to Contoso's global sales site, and a link to Contoso's public website.

You will then modify the home page for the site definition. You will add links in the main body to enable sales persons to easily access the document libraries, and you will add an empty Web Part zone. The Web Part zone will enable sales managers to easily add components to their sales site.

Finally, you will deploy the site definition, and create an instance of the site definition for one of Contoso's product's sales team.

The main tasks for this exercise are as follows:

1. Create a New Solution
2. Add a Site Definition to the Solution
3. Customize the Site Definition
4. Modify the Site Home Page

5. Test the Site Definition

► **Task 1: Create a New Solution**

- Start Visual Studio 2012.
- Create a new project named **SalesSite** in the **E:\Labfiles\Starter\** folder by using **the SharePoint 2013 - Empty Project** template. Configure the project to deploy as a farm solution and use the **http://dev.contoso.com** site for debugging.

► **Task 2: Add a Site Definition to the Solution**

- Add a site definition named **SalesSiteDefinition** to the project.
- In the **webtemp** XML file, in the **Template** element, change the value of the **ID** attribute to **12000**.
- **Note:** The Template ID for a custom site definition should be a value over 10000 that is not used by any other site definitions in each locale.
- In the **Configuration** element, set the following properties:

Property	Value
Title	Sales Site
Description	A site for Contoso sales teams.
DisplayCategory	Contoso Sites

- Save all changes.

► **Task 3: Customize the Site Definition**

- Modify the Onet.xml file to include the top-level navigation, navigation bar. The top-level navigation bar has ID **1002**, and you can obtain the localized name by using the following code to retrieve the name from the resources file:

```
$Resources:core,category_Top;
```

- Add links with the following properties to the top-level navigation bar:

Name	Url
Global Sales Site	http://sales.contoso.com
Contoso Home Page	http://www.contoso.com

- Add a document library to the site definition with the following properties (the **Type** ID for a document library is **101**):

Property	Value
Url	Lists/SalesDocuments
Title	Sales Documents
Description	Use this library to store documents related to specific sales opportunities.

- Add a document library to the site definition with the following properties:

Property	Value
Url	Lists/SalesResources
Title	Sales Resources
Description	Use this library to store generic documents that all members of the sales team can use to support any sales opportunity.

- Save all changes.

#### ► Task 4: Modify the Site Home Page

- On the **default.aspx** page, replace the welcome messages with **Welcome to the Sales Portal**.
- Add the message **Use the Sales Documents library to store documents related to a specific sales opportunity**, as a new paragraph after the page header. Make the **Sales Documents** text a link to the **Sales Documents** document library.
- Add the message **Use the Sales Resources library to store generic documents to support any sales opportunity**, as a new paragraph. Make the **Sales Resources** text a link to the **Sales Resources** document library.
- Save all changes.

#### ► Task 5: Test the Site Definition

- Because the solution does not include any compiled code, configure the properties of the **SalesSite** project to not include the assembly in the SharePoint solution package.
- Save all changes.
- Start the solution without debugging to deploy the solution.



**Note:** The first time that you run the app, your browser may display an error due to a timeout issue. You can ignore this error and continue with the next step.

- In Internet Explorer, browse to the **Pain Relief** site (<http://pain-relief.contoso.com>).
- On the **Pain Relief** site, create a subsite named **Sales Site** with the URL <http://pain-relief.contoso.com/sales>, by using the **Sales Site** template.
- On the **Sales** site, verify that the home page contains links to the two document libraries.
- View the site contents, and verify that the site contains two document libraries, one named **Sales Documents** and one named **Sales Resources**.
- Close Internet Explorer.

**Results:** After completing this exercise, you should have created a site definition.

## Exercise 2: Creating a List Definition

### Scenario

In this exercise, you will create a list definition. The list definition will define a list that sales people can use to track sales leads. You will use a Feature to deploy the list definition. You will also use the Feature to deploy an instance of the new list definition.

Next, you will modify your custom site definition to include the list Feature. You will then deploy the updated site definition and test the new list functionality.

The main tasks for this exercise are as follows:

1. Add a List Definition and Instance
2. Modify the List Deployment Feature
3. Add the SalesLeadsList Feature to the Site Definition
4. Test the Sales Leads List and Sales Site Definition

#### ► Task 1: Add a List Definition and Instance

- In Visual Studio, add a **List** named **SalesLeads**, with the display name **Sales Leads** to the project.
- Modify the **SalesLeads** list to include the following fields:

Field name	Type
Title	Single Line of Text
Customer	Single Line of Text
Estimated Value	Currency
Estimated Completion	Date and time
Sales Person	Person or group
Quote Number	Single Line of Text
Lead Notes	Multiple Lines of Text

- Ensure the title of the list is set to **Sales Leads**.
- Set the URL of the list to **Lists/SalesLeads**.
- Set the description of the list to **Use this list to record and track sales opportunities**.
- Save all changes.

#### ► Task 2: Modify the List Deployment Feature

- Change the **Title** property of the **Feature1** Feature to **Sales Leads List**.
- Rename the **Feature1** Feature to **SalesLeadsList**.
- Save all changes.

► **Task 3: Add the SalesLeadsList Feature to the Site Definition**

- Add the **SalesLeadsList** Feature as a web Feature in the sales site definition **onet.xml** file.
- Save all changes.

► **Task 4: Test the Sales Leads List and Sales Site Definition**

- Start the solution without debugging to deploy the solution.
- In Internet Explorer, browse to the **Anti Virals** site (<http://anti-virals.contoso.com>).
- On the **Anti Virals** site, create a subsite named **Sales Site** with the URL <http://anti-virals.contoso.com/sales>, by using the **Sales Site** template.
- On the **Sales** site, verify that the home page contains links to the two document libraries.
- View the site contents, and verify that the site contains two document libraries, one named **Sales Documents** and one named **Sales Resources**, and a list named **Sales Leads**.
- Browse to the **Sales Leads** list, and then add a new item to the list with the following properties:

Property	Value
Title	Antiviral Trial
Customer	Fabrikam
Estimated Value	5000
Estimated Completion	<i>Choose a date two weeks from today's date</i>
Sales Person	Dan Park
Quote Number	Q1001
Lead Notes	The customer showed an interest in trialing our antiviral products.

- Verify that the sales lead appears in the **Sales Leads** list, and that the default view contains all seven fields.
- Close Internet Explorer.
- In Visual Studio, close the solution.

**Results:** After completing this exercise, you should have created a list definition, created a list instance, and modified a site definition to include a Feature.

## Exercise 3: Developing an Event Receiver

### Scenario

In this exercise, you will create an event receiver that handles the site deleting event. You will add code to the event receiver that creates a backup copy of the parent site collection, and saves the backup to disk on the SharePoint server. You will use a Feature to deploy the event receiver.

You will then create another Feature, with a Farm scope, that you will use to staple the event receiver Feature to the sales site definition that you created in the previous exercises. Finally, you will deploy the new Features to the farm. You will test the Features by creating a new instance of the sales site definition. You will verify that the event receiver Feature has been correctly stapled to the site definition and test the functionality by deleting the site.

The main tasks for this exercise are as follows:

1. Create a New Solution
2. Add an Event Receiver to the Solution to Back Up a Site Collection
3. Modify the Event Receiver Deployment Feature
4. Add a Feature to Staple the Event Receiver Feature to the Sales Site Definition
5. Test the Feature Stapler and Event Receiver

#### ► Task 1: Create a New Solution

- Create a new project named **SiteArchiveFeature** in the **E:\Labfiles\Starter\** folder by using the **SharePoint 2013 - Empty Project** template.
- Configure the project to deploy as a farm solution and to use the **http://dev.contoso.com** site for debugging.

#### ► Task 2: Add an Event Receiver to the Solution to Back Up a Site Collection

- Add an **Event Receiver** to the project, named **SalesWebDeletingReceiver**, that handles the **WebDeleting** event.
- Modify the **WebDeleting** event handler method to create a backup of the current site collection named **Backup.backup** in the **E:\Labfiles\Starter** folder.

 **Note:** Use the **Backup** method of the **SPSiteCollection** class to perform the backup. You can retrieve an **SPSiteCollection** instance from the **Sites** property of an **SPWebApplication** instance.

- Build the solution and resolve any errors.

#### ► Task 3: Modify the Event Receiver Deployment Feature

- Set the title of the **Feature1** Feature to **Web Deleting Event Receiver**.
- Rename the **Feature1** Feature to **WebDeletingEventReceiver**.
- Save all changes.

#### ► Task 4: Add a Feature to Staple the Event Receiver Feature to the Sales Site Definition

- Add a new Feature to the solution with the following properties:

Property	Value
Title	Sales Site Event Receiver Feature Stapler
Scope	Farm
Name	SalesSiteEventReceiverFeatureStapler

- Add a new **Empty Element** named **FeatureStapler** to the solution.
- Add a **FeatureSiteTemplateAssociation** element to the **elements.xml** file to associate the **WebDeletingEventReceiver** Feature with the sales site definition. The template name for the sales site definition is **SalesSiteDefinition#0**.



**Note:** Make sure the **FeatureStapler** element is added to the **SalesSiteEventReceiverFeatureStapler** Feature, and not the **WebDeletingEventReceiver** Feature.

- Save all changes.
- **Task 5: Test the Feature Stapler and Event Receiver**
- Start the solution without debugging to deploy the solution.
- In Internet Explorer, browse to the **Cosmetics** site (<http://cosmetics.contoso.com>).
- On the **Cosmetics** site, create a subsite named **Sales Site** with the URL <http://cosmetics.contoso.com/sales>, by using the **Sales Site** template.
- On the **Sales** site, browse to the **Site Features** page.
- Verify that the **Web Deleting Event Receiver** Feature has a status of **Active**.
- Delete the **Sales** site.
- Use **File Explorer** to browse to the **E:\Labfiles\Starter** folder.
- Verify that SharePoint has added a backup file named **Backup.backup** to the folder.
- Close File Explorer.
- Close Visual Studio.

**Results:** After completing this exercise, you should have developed an event receiver that responds to a site deleting event. You should also have used Feature stapling to add a Feature to an existing site definition.



## Module Review and Takeaways

In this module, you learned about custom sites and components. You learned how to create templates to enable you to quickly deploy common new sites and lists, and how you can use site columns and content types to group related data, and minimize development effort by sharing columns and content types between sites and lists. Finally, you reviewed the site life cycle, how you can influence the site life cycle, and how you can respond to site life cycle events.

### Review Question(s)

#### Test Your Knowledge

Question	
Which of the following options accurately describes the process used by SharePoint when you add a site content type to a list instance?	
Select the correct answer.	
<input type="checkbox"/>	SharePoint adds the site content type to the list.
<input type="checkbox"/>	SharePoint creates a new content type, which is a copy of the original content type, and inherits from the same parent content type as the site content type.
<input type="checkbox"/>	SharePoint creates a new content type, which is a copy of the original content type, and inherits from the site content type.
<input type="checkbox"/>	SharePoint creates a new list-scoped content type, which inherits from the site content type, and adds the new content type to the list.
<input type="checkbox"/>	SharePoint creates a new, hidden, site-scoped content type, which inherits from the original site content type, and adds the new content type to the list.

#### Test Your Knowledge

Question	
Which of the following items can you not include in a list schema?	
Select the correct answer.	
<input type="checkbox"/>	Fields
<input type="checkbox"/>	Toolbars
<input type="checkbox"/>	Content types
<input type="checkbox"/>	Views
<input type="checkbox"/>	Content/Data

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
Feature stapling adds a Feature to both existing and new sites that are based on the stapled site definition.	

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
Host-named site collections are always a better choice when compared to multiple web applications, each with a host header.	

# Module 14

## Customizing User Interface Elements

### Contents:

Module Overview	14-1
Lesson 1: Working with Custom Actions	14-2
Lesson 2: Using Client-Side User Interface Components	14-14
Lab A: Using the Edit Control Block to Launch an App	14-22
Lesson 3: Customizing the SharePoint List User Interface	14-26
Lab B: Using jQuery to Customize the SharePoint List User Interface	14-34
Module Review and Takeaways	14-36

## Module Overview

SharePoint 2013 provides users with a functional and consistent interface that they can quickly learn how to use. However, you may want to sometimes customize this interface to meet the demands of your organization. In this module, you will learn different ways of customizing the SharePoint user interface: from adding buttons to the ribbon to modifying the appearance of list views.

### Objectives

After completing this module, you will be able to:

- Use custom actions to modify the SharePoint user interface.
- Use JavaScript to work with client-side SharePoint user interface components.
- Describe how to modify the appearance and behavior of list views and forms.

## Lesson 1

# Working with Custom Actions

When you provision your SharePoint site, it provides you with out-of-the-box controls and functionality that enable your users to navigate and use your site. However, you can also extend the SharePoint user interface by creating controls and functionality specific to your site, by creating custom actions. By using custom actions, you can add buttons to the ribbon, extend list item menus to provide additional options, and remove or replace existing controls. This functionality enables you to extend the user interface of your site, while ensuring that your users remain in the familiarity of a SharePoint environment.

In this lesson, you will learn how to use custom actions to modify the SharePoint user interface.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe how custom actions work.
- Use the **CustomAction** element.
- Customize the SharePoint ribbon.
- Customize other SharePoint objects.
- Use the Custom Action templates.
- Deploy custom actions.

### Introduction to Custom Actions

Custom actions enable you to add, remove, or replace controls that already exist in your SharePoint site. The areas that you can use custom actions include:

- The SharePoint ribbon.
- The Site settings menu.
- Navigational menus.
- List item menus.

You can restrict the use of your custom actions to specific types of lists or files. For example, you can set your custom ribbon item to be available when viewing a specific type of list or display a custom item only in a menu for Word documents, not other file types. When you create your custom action, you also define its scope, which restricts the items for which it will be available.

You can create custom actions as part of a sandboxed solution, a farm solution, or an app for SharePoint. When you create the custom action, Visual Studio either adds it to an existing Feature in the project or adds a new Feature to the project so that when you deploy the project, the custom action is deployed as part of the Feature.



**Note:** In SharePoint Online, custom actions can only be deployed as a sandbox solution. Custom actions in an app are deployed to either the host web or app web.

- You can create custom actions for:
  - The SharePoint ribbon
  - The Site settings menu
  - Navigational menus
  - List item menus
- You can deploy custom actions as part of a Feature in a:
  - Sandbox solution
  - Farm solution
  - App for SharePoint

## Using the CustomAction Element

In sandbox and farm solutions, you define a custom action by adding an Empty Element item to your SharePoint project. This generates an XML file that contains an **Elements** element. Each Elements file can contain one or more custom actions, and you can also include more than one Elements file in a project to logically group your custom actions. Each Elements file is automatically referenced to a feature; therefore, you can deploy them all as part of a single feature.

```
<Elements
 xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction

 Id="Microsoft.SharePointStandardMenu.SiteActions.Contact"

 GroupId="SiteActions"
 Location="Microsoft.SharePoint.StandardMenu"
 Title="Contact"
 Description="Contact page" >
 <UrlAction Url="-site/_layouts/15/contact.aspx"/>
 </CustomAction>
</Elements>
```



**Note:** If you add an Empty Element item to a project that does not contain a Feature, a new Feature is automatically added to the project.

### CustomAction element

To begin creating your custom action, you add a **CustomAction** element to the **Elements** element. The **CustomAction** element defines the look and behavior of your custom action, whether it is a menu item or ribbon component. The information in the following table describes the key attributes for the **CustomAction** element.

Attribute	Type	Description
Description	Text	A short description to explain the purpose of the custom action.
GroupId	Text	The name of the group to which the custom action belongs. This can be the ID of an existing group or a custom group.
Id	Text	A unique name to identify the custom action.
Location	Text	The place in your SharePoint site in which to set the custom action.
RegistrationID	Text	The type of list or content type with which the custom action is associated.
Title	Text	The text that describes the purpose of the custom action to the user.

The only required attribute for a **CustomAction** element is the **Title** attribute; all other attributes are optional.



**Additional Reading:** For more information about the attributes of the **CustomAction** element, see *CustomAction Element* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311868>.

The **CustomAction** element requires one child element. For a link on a page or menu, it requires the **UrlAction** element; whereas for a ribbon customization, it requires the **CommandUIExtension** element.

MCT USE ONLY. STUDENT USE PROHIBITED

The following code sample shows the definition for a custom action that adds a custom link to the Site settings menu.

### Site Settings Custom Action

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 Id="Microsoft.SharePointStandardMenu.SiteActions.Contact"
 GroupId="SiteActions"
 Location="Microsoft.SharePoint.StandardMenu"
 Title="Contact"
 Description="Contact page" >
 <UrlAction Url="~site/_layouts/15/contact.aspx"/>
 </CustomAction>
</Elements>
```

The **Elements** element can also contain two other elements:

- The **CustomActionGroup** element, which enables you to create a custom group containing multiple custom actions. In the example, the custom action is assigned to a predefined group called SiteActions.
- The **HideCustomAction** element, which enables you to hide an existing actions. When you hide an action, you can either simply hide it or replace it with a custom action.



**Additional Reading:** For more information about the **CustomActionGroup** and **HideCustomAction** elements, see *Custom Action Definition Schema* at <http://go.microsoft.com/fwlink/?LinkID=311869>.

## Customizing the Ribbon

The SharePoint ribbon provides users with the controls relevant to the page they are viewing on your SharePoint site. For example, if they are viewing a document library page, document-related controls appear on the ribbon.

The ribbon has a hierarchical structure and is divided into the following key components:

- Contextual tab groups
- Tabs
- Groups
- Controls

These components are designed to enable users to easily locate the control they require when they are on a particular page.

The following image shows the ribbon components for the default Document library.

Image not available in the media folder

You can customize the ribbon at any of the component levels. For example, you can add a button to an existing group on a tab, or you can create your own tab group that contains custom groups and buttons.

- Using the CustomAction element to customize the ribbon:
  - CommandUIDefinitions element
  - CommandUIHandlers element
- Laying out the controls:
  - Group layouts
  - Scaling

In addition to adding components to the ribbon, you can also remove standard components or replace them with custom components.

### Using the **CustomAction** element to customize the ribbon

When you are customizing the ribbon, the **CustomAction** element contains a **CommandUIExtension** element with two child elements: **CommandUIDefinitions** and **CommandUIHandlers**.

#### **CommandUIDefinitions element**

The **CommandUIDefinitions** element contains one or more **CommandUIDefinition** elements, which define the various components of your ribbon custom action. The element contains one attribute, an optional text attribute named **Location**, which specifies where the root element of the custom action will be situated on the ribbon.



**Additional Reading:** For more information about the locations available on the ribbon, see *Default Server Ribbon Customization Locations* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311871>.

The child element of the **CommandUIDefinition** element defines the type of custom control to add to the ribbon. For example, if you want to add a custom group to a tab, use the **Group** element. When adding custom controls, you should consider the hierarchical structure of the ribbon. For example, you cannot directly add a custom button to a tab without first creating a group. Similarly, you cannot create a custom tab by itself without adding a group and control. The further up the hierarchy you start, the more elements you will have to create.

You can add a whole range of different types of controls to the ribbon as described in the following table.

Control	Description
Button	A standard Windows push button
CheckBox	A standard Windows check box
ComboBox	A standard Windows combo box
DropDown	A standard Windows drop-down list box
FlyoutAnchor	A button with a down arrow used for opening a menu
GalleryButton	A button that can host an image of any size
Label	A standard Windows label control
MRUSplitButton	A combination of a button and a drop-down menu that displays a list of the most-recently used items
Spinner	A standard Windows spinner control
SplitButton	A combination of a button and a drop-down menu
TextBox	A standard Windows text box control
ToggleButton	A standard Windows toggle button control



**Additional Reading:** For more information about the different controls, see the *Child Elements* list on the *Controls Element* page at <http://go.microsoft.com/fwlink/?LinkID=311870>.

### **CommandUIHandlers element**

The **CommandUIHandlers** element contains one or more **CommandUIHandler** elements, which describe the behavior of your custom action. The **CommandUIHandler** contains three attributes:

- The required **Command** attribute must match the **Command** attribute for the relevant **CommandUIDefinition** element and links the two elements together.
- The required **CommandAction** attribute defines the script statement to execute when the custom action is invoked.
- The optional **EnabledScript** attribute defines the script statement to execute to determine whether the command is enabled or disabled.



**Additional Reading:** For more information about the **CommandUIHandler** element, see *CommandUIHandler Element* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311872>.

The following code sample shows the markup to add a single button to the Share & Track group in the ribbon for a document library. The **Location** attribute of the **CommandUIDefinition** element defines the location of the button and the **Button** element defines the appearance of the button. The **CommandUIHandler** defines the code to run when a user clicks the button.

#### **Share & Track Custom Action**

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 Id="Ribbon.Library.Share.CustomAction"
 Location="CommandUI.Ribbon"
 RegistrationId="101"
 RegistrationType="List">
 <CommandUIExtension>
 <CommandUIDefinitions>
 <CommandUIDefinition
 Location="Ribbon.Library.Share.Controls._children">
 <Button
 Id="Ribbon.Library.Share.Control.CustomAction"
 Command="CustomActionCommand"
 LabelText="Custom Action"
 TemplateAlias="o2"
 Sequence="40"/>
 </CommandUIDefinition>
 </CommandUIDefinitions>
 </CommandUIExtension>
 <CommandUIHandlers>
 <CommandUIHandler
 Command="CustomActionCommand"
 CommandAction="javascript:alert('Custom action launched!');">
 </CommandUIHandler>
 </CommandUIHandlers>
 </CustomAction>
 </Elements>
```





**Additional Reading:** For a full list of the values that you can use for the **RegistrationId** attribute, see *SPListTemplateType enumeration* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311873>.

## Laying out ribbon controls

The controls on the ribbon can be displayed in different ways. For example, some groups contain two or three large controls in a line, whereas others have one large control with smaller controls displayed alongside in a vertical list. When you resize the browser window, these layouts can change to allow for the new browser window size. You define the control layouts and resizing behavior of your groups in the **CustomAction** element.

### Group layouts

You define the layout for your custom group by defining a **GroupTemplate** element in the **CommandUIDefinition** element. This has one child element, **Layout**, which you use to describe the layout that you require. The standard SharePoint group layouts are defined in the CMDUI.xml file located in the %Program Files%\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\GLOBAL\XML folder. You can copy these templates and then either use them as-is or customize them for your own requirements.

The following code sample shows the predefined group template for the clipboard group in SharePoint. You can copy this template into your own custom actions and then customize it for your own group and controls.

### GroupTemplate Element

```
<GroupTemplate Id="Ribbon.Templates.ClipboardGroup">
 <Layout Title="Large">
 <Section Type="OneRow">
 <Row>
 <ControlRef TemplateAlias="paste" DisplayMode="Large" />
 </Row>
 </Section>
 <Section Type="TwoRow">
 <Row>
 <ControlRef TemplateAlias="copy" DisplayMode="Medium" />
 </Row>
 <Row>
 <ControlRef TemplateAlias="cut" DisplayMode="Medium" />
 </Row>
 </Section>
 <Section Type="TwoRow">
 <Row>
 <ControlRef TemplateAlias="undo" DisplayMode="Medium" />
 </Row>
 <Row>
 <ControlRef TemplateAlias="redo" DisplayMode="Medium" />
 </Row>
 </Section>
 </Layout>
</GroupTemplate>
```



**Additional Reading:** For more information about the **GroupTemplate** element, see *GroupTemplate Element* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311874>.

After you define your **GroupTemplate** element, you assign the template to a control by setting the **Template** attribute of the **Group** element to the **ID** attribute of the **GroupTemplate** element.

## Scaling

When you resize the browser window, the SharePoint ribbon transforms to accommodate this change in size. You can use the **Scaling** element in the **Tab** element to define how your groups should react when the browser window is resized and also to define the maximum size for a group of controls.

Inside the **Scaling** element, you define the **MaxSize** element, which defines the maximum size for the group of controls, and the **Scale** element, which defines how the controls are sized.

The following code sample shows the predefined scaling for the clipboard group in SharePoint. You can copy this template into your own custom actions and then customize it for your own group and controls

### Scaling Element

```
<Scaling Id="Ribbon.PostListForm.Edit.Scaling">
 <MaxSize Id="Ribbon.PostListForm.Edit.Scaling.Clipboard.MaxSize" Sequence="10"
 GroupId="Ribbon.PostListForm.Edit.Clipboard" Size="LargeMedium" />
 <Scale Id="Ribbon.PostListForm.Edit.Scaling.Clipboard.Popup" Sequence="50"
 GroupId="Ribbon.PostListForm.Edit.Clipboard" Size="Popup" />
</Scaling>
```



**Additional Reading:** For more information about scaling, see *Scaling Element* in the MSDN library at <http://go.microsoft.com/fwlink/?LinkID=311875>.

## Demonstration: Customizing the Ribbon

In this demonstration, you will see how to add a custom action Library tab with a custom group and two custom buttons to a document page.

### Demonstration Steps

- Start the **20488B-LON-SP-14** virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Internet Explorer**.
- In Internet Explorer, in the address bar, type **http://team.contoso.com**, and then press Enter.
- On the Quick Launch toolbar, click **Documents**.
- On the ribbon, click **LIBRARY**.
- Review the groups that are present on the ribbon, and then close **Internet Explorer**.
- On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
- On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
- In the **Open Project** dialog box, browse to the **E:\Democode** folder, click **RibbonCustomActions.sln**, and then click **Open**.
- In Solution Explorer, expand **Features**, and then double-click **ContosoCustomActionsFeature**.
- In the **Items in the Feature** list, point out the **ContosoCustomActions** element.
- In Solution Explorer, expand **ContosoCustomActions**, and then double-click **LibraryCustomActions.xml**.
- Point out the attributes for the **CustomAction** element.
- Point out the **Location** for the first **CommandUIDefinition**.

- Point out the attributes for the **Group** element.
- Point out the **Button** elements and their **Command** attributes, and then point out the matching **Command** attributes in the **CommandUIHandler** elements.
- Point out the **CommandAction** attribute for each **CommandUIHandler**.
- On the **BUILD** menu, click **Deploy RibbonCustomActions**.
- On the Start screen, click **Internet Explorer**.
- In Internet Explorer, in the address bar, type **http://team.contoso.com**, and then press Enter.
- On the Quick Launch toolbar, click **Documents**.
- On the ribbon, click **LIBRARY**.
- Point out the **Contoso** tab, and the two new buttons.
- Click the **About** button, note the **Message from webpage** popup, and then click **OK**.
- Close Internet Explorer, and then close Visual Studio.

## Customizing Other SharePoint Objects

In addition to customizing the ribbon, you can also customize other SharePoint objects.

### Customizing the Edit Control Block

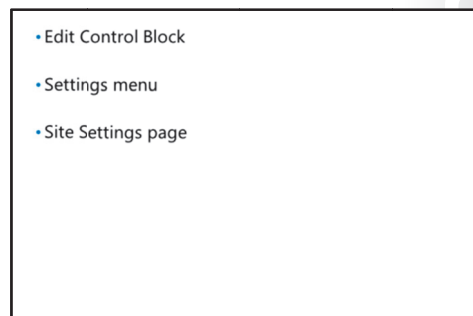
The Edit Control Block (ECB) is the menu that appears when you click the down arrow next to an item name in a list or library. It provides the user with a list of relevant options for the item type that they are using. For example, the ECB for a Word document gives the user the option to check out the item, whereas the ECB for a task provides the user with the option to send an alert. You can customize the ECB to add your custom actions to the list.

To create an ECB custom action, you set the **Location** attribute of the **CustomAction** element to **EditControlBlock** and then set the **UrlAction** child element to the page to which the user will be directed. You use the **RegistrationType** and **RegistrationId** attributes to define the registration attachment for the action.

The following code example shows how to define an ECB custom action.

#### ECB Custom Action

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 Id="EmailDocument"
 RegistrationType="FileType"
 RegistrationId="docx"
 Location="EditControlBlock"
 Title="Email Document">
 <UrlAction Url="~site/_layouts/15/sendemail.aspx"/>
 </CustomAction>
</Elements>
```



## Customizing the Settings menu

You can also add items to the SharePoint **Settings** menu by creating a custom action. In this scenario, you set the **Location** attribute of the **CustomAction** element to **Microsoft.SharePoint.StandardMenu**, set the **GroupId** attribute to **SiteActions**, and set the **UrlAction** child element to the page that you want to display.

The following code example shows how to define a custom action for the **Settings** menu.

### Settings Menu Custom Action

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 Id="CreateSite"
 GroupId="SiteActions"
 Location="Microsoft.SharePoint.StandardMenu"
 Title="Create Site">
 <UrlAction Url="~site/_layouts/15/newsbweb.aspx"/>
 </CustomAction>
</Elements>
```

## Customizing the Site Settings page

To customize the Site Settings page, you set the **Location** attribute of the **CustomAction** element to **Microsoft.SharePoint.SiteSettings**, set the **GroupId** attribute to the group that you want to customize, and set the **UrlAction** child element to the page that you want to display.

The following code example shows how to define a custom action for the Site Settings page.

### Site Settings Page Custom Action

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 Id="CreateSite"
 GroupId="SiteAdministration"
 Location="Microsoft.SharePoint.SiteSettings"
 Title="Create Site">
 <UrlAction Url="~site/_layouts/15/newsbweb.aspx"/>
 </CustomAction>
</Elements>
```

You can also use the **HideCustomAction** element to hide items on the Site Settings page. You use the **GroupId** and **Location** attributes of the **CustomAction** element to identify the item that you want to hide.

The following code example shows how to hide a custom action on the Site Settings page.

### Site Settings Page Hide Custom Action

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <HideCustomAction
 Id="HideDeleteSite"
 HideActionId="DeleteWeb"
 GroupId="SiteAdministration"
 Location="Microsoft.SharePoint.SiteSettings" />
</Elements>
```

## Using the Custom Action Templates

When you are developing an app for SharePoint, you can use custom action templates to reduce the work involved in creating a custom action. You use a template by selecting either **Menu Item Custom Action** or **Ribbon Custom Item** in the **Add New Item** dialog box. You then enter information in the **Create Custom Action** dialog box to configure whether the custom action should use the host web or app web, where it is scoped to, what item it is scoped to, and how and where the custom action should appear. After you provide this information, Visual Studio adds a preconfigured Elements file, and a Feature if required, to your project containing template markup for the custom action.

- Menu Item Custom Action:
  - **Location** attribute set to "EditControlBlock"
  - Set **UrlAction** element to required page
- Ribbon Custom Action:
  - **Location** attribute set to "CommandUI.Ribbon"
  - Modify **CommandUIDefinitions** element as required

### Menu Item Custom Action

The Menu Item Custom Action template contains a **CustomAction** element with the **Location** attribute set to **EditControlBlock** to add the custom action to the menu. You can then just modify the **UrlAction** element to point to the page to which you want the custom action to navigate.

The following code sample shows the default markup in the menu item custom action template.

#### Menu Item Custom Action Template

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction Id="f9dfe454-7b0b-4abe-9174-e720e047fb5a.MenuItemCustomAction1"
 RegistrationType="List"
 RegistrationId="101"
 Location="EditControlBlock"
 Sequence="10001"
 Title="Invoke 'MenuItemCustomAction1' action">
 <!--
 Update the Url below to the page you want the custom action to use.
 Start the URL with the token ~remoteAppUrl if the page is in the
 associated web project, use ~appWebUrl if page is in the app project.
 -->
 <UrlAction
 Url="~appWebUrl/Pages/Default.aspx?{StandardTokens}&SPListItemId={ItemId}&SPListId={ListId}" />
 </CustomAction>
</Elements>
```

### Ribbon Custom Action

The Ribbon Item Custom Action template also contains a **CustomAction** template, but with the **Location** element set to **CommandUI.Ribbon**. In this template, you must modify the **CommandUIDefinitions** element to define the controls and action that you require for your custom action.

The following code sample shows the default markup in the ribbon item custom action template.

### Ribbon Item Custom Action Template

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction Id="6c87b148-75bc-449e-b0e1-16e7853c90e2.RibbonCustomAction1"
 RegistrationType="List"
 RegistrationId="101"
 Location="CommandUI.Ribbon"
 Sequence="10001"
 Title="Invoke 'RibbonCustomAction1' action">
 <CommandUIExtension>
 <!--
 Update the UI definitions below with the controls and the command actions
 that you want to enable for the custom action.
 -->
 <CommandUIDefinitions>
 <CommandUIDefinition Location="Ribbon.ListItem.Actions.Controls._children">
 <Button Id="Ribbon.ListItem.Actions.RibbonCustomAction1Button"
 Alt="Request RibbonCustomAction1"
 Sequence="100"
 Command="Invoke_RibbonCustomAction1ButtonRequest"
 LabelText="Request RibbonCustomAction1"
 TemplateAlias="o1"
 Image32by32="_layouts/15/images/placeholder32x32.png"
 Image16by16="_layouts/15/images/placeholder16x16.png" />
 </CommandUIDefinition>
 </CommandUIDefinitions>
 <CommandUIHandlers>
 <CommandUIHandler Command="Invoke_RibbonCustomAction1ButtonRequest"

CommandAction="~appWebUrl/Pages/Default.aspx?{StandardTokens}&SPListItemId={ItemId}&SPLis
tId={ListId}"/>
 </CommandUIHandlers>
 </CommandUIExtension >
 </CustomAction>
</Elements>
```

## Deploying Custom Actions

Custom actions are always deployed as a Feature, whether in a solution or in an app for SharePoint. Because you can attach multiple custom actions to a single Feature, you can deploy multiple actions together.

### Deploying custom actions in a solution

To deploy your custom actions to the local SharePoint site, you deploy the solution directly from Visual Studio. This uploads your solution to the SharePoint site and activates the solution.

If you want to deploy to an alternative site, such as your production site, you must publish the solution into a SharePoint package (a WSP file). For sandboxed solutions, you can publish to either a SharePoint site or to the file system. For farm solutions, you can only publish to the file system. When you publish a sandboxed solution to a SharePoint site, it is only uploaded to the site and you must manually activate it by using your browser. When you publish to the file system, you must then upload the package to your SharePoint site and then activate it.

- In a solution:
  - Deploy directly from Visual Studio to the local SharePoint site
  - Deploy to an alternative site by creating a SharePoint package
- In an app for SharePoint:
  - Deploy directly from Visual Studio to the local SharePoint site
  - Deploy to an alternative site or store by creating an app package

You can update your custom actions by making changes and then redeploying the feature to your site, using the method used to originally deploy it. If you want to remove your custom actions, you can disable the feature from the **Site Settings** menu in SharePoint.

### **Deploying custom actions in an app for SharePoint**

Similar to deploying custom actions in a solution, you can also deploy your app for SharePoint directly from Visual Studio to the local SharePoint site. After you deploy the app, it will be available on the Site Contents page in SharePoint.

If you want to deploy your app to an alternative site or make it available for other users, you can publish the app into a package and then upload it to the appropriate location.

## Lesson 2

# Using Client-Side User Interface Components

You can use client-side user interface components to provide users with information about the actions they are performing. For example, you can provide status bar messages to provide status information about an action or provide hints about elements by using callouts.

### Lesson Objectives

After completing this lesson, you will be able to:

- Display status bar messages.
- Display notifications.
- Use SharePoint dialog boxes.
- Display callouts.
- Describe the **SPAnimation** library.
- Use Focus on Content functionality.

### Displaying Status Bar Messages

Many applications use status bars to display information to the user in an unobtrusive way. You can use the SharePoint status bar to display messages to the user, customizing the color to indicate the priority of the message. You use methods of the **SP.UI.Status** class to display, customize, update, and remove status messages in SharePoint.

The following table describes the methods that are available when setting the message for the status bar.

```

• Use methods of the SP.UI.Status object:
 • addStatus
 • updateStatus
 • setStatusPriColor
 • removeStatus
 • removeAllStatus

function showStatusMessage () {
 strStatusID = SP.UI.Status.addStatus("Info:", "Message",
true);
 SP.UI.Status.setStatusPriColor(strStatusID, "yellow");
}

```

Method	Description	Parameters
addStatus	Adds a status message to the page and returns the id of that message	<b>strTitle</b> – The title of the status <b>strHTML</b> – The status message contents <b>atBeginning</b> – A Boolean value to specify whether the message should appear at the beginning of the list
updateStatus	Updates an existing status message	<b>id</b> – The ID of the status to update <b>strHml</b> – The new contents for the status message
setStatusPriColor	Customizes the color and icon for a status	<b>id</b> – The ID of the status to customize <b>strColor</b> – The color to set the status message
removeStatus	Removes a status message	<b>id</b> – The ID of the status to remove



Method	Description	Parameters
removeAllStatus	Removes all status messages	<b>Hide</b> – A Boolean value to specify that the status messages are to be hidden

The **setStatusPriColor** method enables you to customize the color of the status bar:

- Red – Very important
- Yellow – Important
- Green – Success
- Blue – Information

You can implement the **SP.UI.Status** methods on a page by using the HTML Editor or as part of the **CommandAction** element in a custom action.

The following code sample shows the code to display a status message on a site page.

#### Display a Status Message

```
function showStatusMessage () {
 strStatusID = SP.UI.Status.addStatus("Information : ", "Status message created.",
true);
 SP.UI.Status.setStatusPriColor(strStatusID, "yellow");
}
```

## Demonstration: Displaying Status Bar Messages

In this demonstration, you will see how to use the methods of the **SP.UI.Status** object to display and customize a message to the user.

### Demonstration Steps

- Start the **20488B-LON-SP-14** virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Visual Studio 2012**.
- On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
- In the **Open Project** dialog box, browse to **E:\Democode**, click **StatusBarMessage.sln**, and then click **Open**.
- In Solution Explorer, double-click **InformationBar**.
- Point out that the **CommandAction** attribute in the **CommandUIHandler** element contains JavaScript code to display a status message and change its color.
- On the **BUILD** menu, click **Deploy StatusBarMessage**.
- On the Start screen, click **Internet Explorer**.
- In Internet Explorer, in the address bar, type **http://team.contoso.com**, and then press Enter.
- On the Quick Launch toolbar, click **Documents**.
- On the ribbon, click **Library**.
- On the ribbon, in the **Share & Track** group, click **Display Status Message**.
- Point out the yellow information bar.

- Close Internet Explorer.
- Close Visual Studio.

## Displaying Notifications

In addition to displaying status bar messages to the user, you can also display notifications. These appear under the ribbon on the right side of the page, and by default, they display for only five seconds. If you want to use this area to display more permanent information in the notification area, you can specify to display the notification until it is removed through code. You use methods of the **SP.UI.Notify** class to add and remove notifications.

```

• Use methods of the SP.UI.Notify object:
 • addNotification
 • removeNotification

function showNotification(){
 SP.UI.Notify.addNotification("Notification...", true)
}

```

The following table describes the methods for displaying and removing notifications.

Method	Description	Parameters
addNotification	Adds a notification to the page and returns the id of that notification	<b>strHtml</b> – The contents for the notification <b>bSticky</b> – A Boolean value specifying whether the notification should remain on the page until removed
removeNotification	Removes a notification from the page	<b>nid</b> – The ID of the notification to remove

Similar to status bar messages, you can implement **SP.UI.Notify** methods on a page by using the HTML editor or as part of the **CommandAction** element in a custom action.

The following code sample shows the code to add a notification to a page.

### Display a Notification

```

function showNotification(){
 SP.UI.Notify.addNotification("Custom notification...", true)
}

```

## Demonstration: Displaying Notifications

In this demonstration, you will see how to use the methods of the **SP.UI.Notify** object to display notifications to the user.

### Demonstration Steps

- Start the **20488B-LON-SP-14** virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Visual Studio 2012**.
- On the **FILE** menu, point to **Open**, and then click **Project/Solution**.

- In the **Open Project** dialog box, browse to **E:\Democode**, click **Notification.sln**, and then click **Open**.
- In Solution Explorer, double-click **Notification**.
- Point out that the **CommandAction** attribute in the **CommandUIHandler** element contains JavaScript code to display a notification.
- On the **BUILD** menu, click **Deploy Notification**.
- On the Start screen, click **Internet Explorer**.
- In Internet Explorer, in the address bar, type **http://team.contoso.com**, and then press Enter.
- On the Quick Launch toolbar, click **Documents**.
- On the ribbon, click **Library**.
- On the ribbon, in the **Share & Track** group, click **Notification**.
- Point out the notification that appears.
- Close Internet Explorer.
- Close Visual Studio.

## Using SharePoint Dialog Boxes

Notifications and status bar messages are useful for displaying unobtrusive information to the user. However, there are times when you need to ensure that the user has seen a message before they can continue working. To do this, you can use SharePoint dialog boxes.

### Displaying modal dialog boxes

You display dialog boxes to the user by calling the **showModalDialog** method of the **SP.UI.ModalDialog** class. This method takes one parameter, the **options** parameter, which you use to configure the options for the dialog box. You create a new instance of the **SP.UI.DialogOptions** class, set the properties of the class, and then pass the class as the **options** parameter to the **showModalDialog** method.

The following table shows the properties of the **DialogOptions** class.

Property	Type	Description
allowMaximize	boolean	Specifies whether the dialog box can be maximized.
args	object	Contains data passed to the dialog box.
autosize	boolean	Specifies whether the dialog box can be resized.
dialogReturnValueCallback	function	Specifies the function pointer to a callback function that can execute after the dialog box is dismissed and use data entered in it.

```

• Displaying a modal dialog box:
var NewDialog = new SP.UI.DialogOptions();
newDialog.html = htmlElement;
SP.UI.ModalDialog.showModalDialog(newDialog)

• Displaying a wait screen:
var waitDialog =
 SP.UI.ModalDialog.showWaitScreenWithNoClose(
 'Loading page', 'Please wait...');
waitDialog.close();

```

Property	Type	Description
height	numeric	Specifies the height of the dialog box.
html	element	Defines the HTML of the page to be displayed in the dialog box.
showClose	boolean	Specifies whether a <b>Close</b> button appears in the dialog box.
showMaximized	boolean	Specifies whether a <b>Maximize</b> button appears in the dialog box.
title	string	Defines the title displayed in the dialog box.
url	string	Defines the URL of the page to display in the dialog box. If this property and the HTML property are both set, the URL property takes precedence.
width	numeric	The width of the dialog box.
x	numeric	The x offset of the dialog box.
y	numeric	The y offset of the dialog box.

The following code sample shows the JavaScript to display a dialog box from an HTML snippet.

### Display a Dialog Box

```
function showMyDialogBox()
{
 var htmlElement = document.createElement('p');
 helloWorldNode = document.createTextNode('Hello World');
 htmlElement.appendChild(helloWorldNode);

 var newDialog = new SP.UI.DialogOptions();
 newDialog.html = htmlElement;
 newDialog.width = 500;
 newDialog.height = 200;
 newDialog.title = 'My Dialog Box';
 SP.UI.ModalDialog.showModalDialog(newDialog);
}
```

### Displaying an app

You can also use dialog boxes to display your apps for SharePoint. If you are launching an app from a button on the ribbon or from an ECB, you can add attributes to the **CustomAction** element to display the app in a dialog window.

The information in the following table shows the properties that you need to add to launch an app in a dialog window.

Property	Type	Description
HostWebDialog	Boolean	Specifies whether to display the app in a dialog window.
HostWebDialogHeight	Numeric	Specifies the height of the dialog window.
HostWebDialogWidth	Numeric	Specifies the width of the dialog window.

The following code example shows the CustomAction element containing the HostWebDialog attributes.

### Launch an App in a Dialog Window

```
<CustomAction
 Id="DialogLaunch"
 RegistrationId="101"
 RegistrationType="List"
 Location="CommandUI.Ribbon"
 Title="Dialog Box"
 HostWebDialog="true"
 HostWebDialogHeight="400"
 HostWebDialogWidth="400">
 . . .
</CustomAction>
```

### Displaying wait screens

You can also access the standard SharePoint wait screen. By default, this displays the phrase "Working on it..."; however, you can also customize it to display your own message.

The following code example shows the JavaScript to display a customized message in the SharePoint wait screen.

### Use the SharePoint Wait Screen

```
var waitDialog = SP.UI.ModalDialog.showWaitScreenWithNoClose('Loading page', 'Please wait...');
// code to execute while wait screen is displayed
waitDialog.close();
```

Using the **showWaitScreenWithNoClose** method ensures that the user cannot close the dialog box; therefore, you need to close it in your code so the user can continue working. If you want the user to be able to close your dialog box, you can use the **SP.UI.ModalDialog.showWaitScreenSize** method, passing a function pointer to the callback function to run when the user closes dialog box.

## Displaying Callouts

Callouts are a new feature in SharePoint 2013 that provide users with additional information about an item on a page when they hover the mouse pointer over the link, similar to tooltips in other applications.

You create callouts by creating a new instance of the **CalloutOptions** class, setting its properties, calling methods, and then passing it to the **CalloutManager.createNew** method. Before you can use the **CalloutManager** and **CalloutOptions** classes, you must ensure that the callout.js library located in the C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS folder is loaded for the page by using the **SP.SOD.executeFunc** method.

```
SP.SOD.executeFunc("callout.js", "Callout", DisplayCallout());

function DisplayCallout()
{
 var calloutElement = document.getElementById("DocInfo");

 var docInfoCalloutOptions = new CalloutOptions();
 docInfoCalloutOptions.ID = "docinfocallout";
 docInfoCalloutOptions.launchPoint = calloutElement;
 docInfoCalloutOptions.beakOrientation = "topBottom";
 docInfoCalloutOptions.content = "Information to display to the user";
 CalloutManager.createNew(docInfoCalloutOptions);
}
```



**Additional Reading:** For more information about the `executeFunc` method, see *SP.SOD.executeFunc(key, functionName, fn) Method* at <http://go.microsoft.com/fwlink/?LinkID=311876>.

The information in the following table describes the key methods and properties of the `calloutOptions` class.

Property	Description
ID	Defines a string identifier for the callout.
launchPoint	Defines the element for which the callout provides information.
beakOrientation	Defines the direction of the arrow pointing to the relevant element (leftRight or topBottom).
content	Defines the information displayed in the callout.
addAction	Defines the text to display and function to call when including an action below the content in the callout.

Not all lists and libraries in SharePoint can use callouts. You can use them with the Document Library, the Asset Library, the Images Library, the Pages Library, and Task Lists. You cannot use them with Custom Lists, Announcement Lists, Calendar Lists, or Discussion Lists.

The following code shows how to create a callout for a DocInfo element.

### Displaying a Callout

```
SP.SOD.executeFunc("callout.js"), "Callout", DisplayCallout({});

function DisplayCallout(){
 var calloutElement = document.getElementById('DocInfo');

 var docInfoCalloutOptions = new CalloutOptions();
 docInfoCalloutOptions.ID = 'docinfocallout';
 docInfoCalloutOptions.launchPoint = calloutElement;
 docInfoCalloutOptions.beakOrientation = 'topBottom';
 docInfoCalloutOptions.content = 'Information to display to the user';
 CalloutManager.createNew(docInfoCalloutOptions);
}
```

## SPAnimation Library

SharePoint 2013 enables you to animate elements on your site by using the SPAnimation library. This library contains JavaScript functions that you can use to animate page elements. For example, it contains a **BasicAnimator** class with methods to fade elements in and out, to move elements, to resize elements, and to clone elements and a **TableAnimator** class with methods to animate the paging and sorting functionality.

The following code example shows how to add strikethrough formatting to a text element and then

```
• BasicAnimator class

function strikeThroughText(){
 var actionedItem =
 document.getElementById("Item1");

 SPAnimationUtility.BasicAnimator.StrikeThrough(
 actionedItem, null, null, null);

 SPAnimationUtility.BasicAnimator.FadeOut(
 actionedItem, null, null);
}
```

fade the element out on a page.

### Using the BasicAnimator Class

```
function strikeThroughAndFadeText(){
 var actionedItem = document.getElementById("Item1");
 SPAnimationUtility.BasicAnimator.StrikeThrough(actionedItem, null, null, null);
 SPAnimationUtility.BasicAnimator.FadeOut(actionedItem, null, null);
}
```

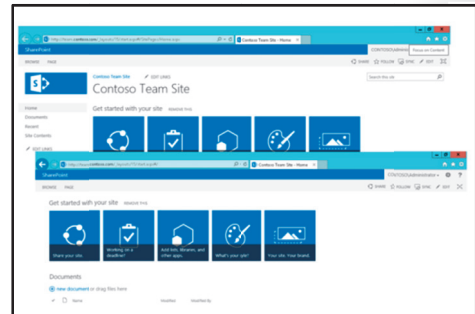
### Using Focus on Content

SharePoint 2013 includes a new button on the Sharing bar titled Focus on Content. When a user clicks this button, SharePoint hides the Quick Launch toolbar and headings to allow the user to focus on the content of the page. After a user clicks the button, the setting is persisted across all the pages in the site for that user until they click the button again.

You can disable this button to prevent users from using it if your organization requires. You do this by setting the visibility property of the style of the element to **hidden**, as shown in the following code.

#### Hide the Focus On Content Button

```
document.getElementById('ctl00_fullscreenmodeBtn').style.visibility = 'hidden';
```



## Lab A: Using the Edit Control Block to Launch an App

### Scenario

Contoso maintains a list of customers in a Contacts list. The sales team at Contoso would like to be able to view a summary of recent sales activities for each customer from the Contacts list. You plan to implement this functionality as an app for SharePoint. First, you will develop a SharePoint-hosted app that displays the order history for a selected customer. You will then develop a custom action that launches the app from the Edit Control Block for a Contacts list item. The custom action must provide the app with the URL of the selected list item. It must also display the app page as a dialog box.

### Objectives

After completing this lab, you will be able to:

- Configure an app to retrieve and display list information.
- Use a custom action to launch an app page as a dialog window.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-14
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

### Exercise 1: Configuring an App to Display Customer Orders

#### Scenario

In this exercise, you will use Visual Studio to set the permissions for an app to read from the Customers and Orders lists and then write a CAML query to retrieve past orders for a customer.

The main tasks for this exercise are as follows:

1. Set Permissions for the App to Read the List
2. Create a CAML Query to Find the Customer ID
3. Test the App

#### ► Task 1: Set Permissions for the App to Read the List

- Start the 20488B-LON-SP-14 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with a password of **Pa\$\$w0rd**.
- Open Visual Studio and then open **CustomerOrder.sln** from the **E:\Labfiles\Starter\CustomerOrder** folder.
- Open the **AppManifest.xml** file, configure **Full Control** permissions on lists, and then save the file.
- Open Internet Explorer and browse to **http://team.contoso.com**.
- Review the information in the **Customers** and **Orders** lists.
- Close Internet Explorer.

#### ► Task 2: Create a CAML Query to Find the Customer ID

- In Visual Studio, open **App.js** from the Scripts folder.
- Locate the **TODD: Task 1. Create Caml query to retrieve the orders list for a given customer ID** comment.



- Under the comment, add code to do the following:
- Create a new instance of a **SP.CamlQuery** object.
- Use the **set\_viewXml** method of the object to define a CAML query as follows:

```
query.set_viewXml("<View><Query><Where><Contains><FieldRef Name='Customer_x003a_ID' /><Value Type='Text'>" + customerID + "</Value></Contains></Where></Query></View>");
```

- Call the **getItem**s method of the **targetList** object, passing the query object and saving the result in the list object.



**Note:** Because the colon in the field name Customer:ID is a special character, in your code you must use the internal name which replaces the colon with `_x003a_`.

- Locate the **TODO: Task 2. Submit query results** comment.
- Under the comment, add code to do the following:
  - a. Call the **load** method of the **currentContext** object, passing the list that you have created.
  - b. Call the **executeQueryAsync** method to define the delegates as follows:

```
currentContext.executeQueryAsync(Function.createDelegate(this, onSuccess),
Function.createDelegate(this, onFail));
```

### ► Task 3: Test the App

- Run the app.
- When Internet Explorer loads, configure the app to have full control of the Orders list.
- Navigate to the Site Contents page, and then view the permissions of the CustomerOrder app.
- Configure the app to read items in the Customers list, and then click the “here” hyperlink.
- Navigate to the **CustomerOrder** page.
- In the **CustomerID** box, type **4**, and then click **Search**.
- Verify that four orders are displayed for Suroor Fatima.
- In the **Customer ID** box, type **2**, and then click **Search**.
- Verify that three orders are displayed for Eli Bowen.
- In the **Customer ID** box, type **1**, and then click **Search**.
- Verify that no orders are displayed for Gustavo Achong.
- Close Internet Explorer.
- In Visual Studio, close the solution.

**Results:** After completing this exercise, you should have implemented and tested the CustomerOrder app.

## Exercise 2: Use a Custom Action to Launch an App

### Scenario

In this exercise, you will add a custom action to your previous code that enables users to use a menu item on the Edit Control Block to retrieve details of a customer's orders.

The main tasks for this exercise are as follows:

1. Get the SPListItemID Token from the Query String
2. Add an ECB to the Customers List
3. Launch the App in a Dialog Box
4. Deploy the App

#### ► Task 1: Get the SPListItemID Token from the Query String

- In Visual Studio, open **EditControlBlock.sln** from the **E:\Labfiles\Starter\EditControlBlock** folder.
- Open **App.js** from the Scripts folder.
- Locate the **TODO: Task 1. Get the SPListItemId token from the query string comment**.
- Under this comment, add code to do the following:
  - a. Create a variable and assign the result of passing the **"SPListItemId"** to the **getQueryStringParams** function to the variable.
  - b. Call the **getList** function, passing your variable to it.
- Locate the **TODO: Task2. Modify the getList function and the caml query** comment.
- Modify the **getList** function signature to take a parameter named **customerID**.
- Remove the line of code that declares and assigns a value to the **customerID** variable.

#### ► Task 2: Add an ECB to the Customers List

- Add a new **Menu Item Custom Action** named **CustomerOrderECB** to the project.
- Expose the custom action to the host web and scope it to the **Contacts** item.
- Specify the text on the menu item to be **Customer Orders**.

#### ► Task 3: Launch the App in a Dialog Box

- Open the **Elements.xml** file for the **CustomerOrderECB** custom action.
- Add the following attributes to the **CustomAction** element:

```
HostWebDialog="true"
HostWebDialogHeight="650"
HostWebDialogWidth="650"
```

- Open **Default.aspx** from the **Pages** folder.
- Add the following element to the **PlaceHolderMain Content** element.

```
<WebPartPages:AllowFraming ID="AllowFraming" runat="server" />
```

- Remove the **span**, **input**, and **button** elements from the **PlaceHolderMain Content** element.

#### ► Task 4: Deploy the App

- Deploy the solution.
- When Internet Explorer loads, configure the app to have full control of the Orders list.
- Navigate to the **Site Contents** page and then view the permissions of the **CustomerOrder** app.
- Configure the app to read items in the **Customers** list and then click the **here** hyperlink.
- Navigate to the **Customers** list.
- In the list, click the ellipsis for **Fatima**, and then click **Customer Orders**.
- In the **Customer Orders** window, verify that four orders are displayed, and then close the dialog window.
- In the list, click the ellipsis for **Bowen**, and then click **Customer Orders**.
- In the **Customer Orders** window, verify that three orders are displayed, and then close the Customer Orders window.
- Close Internet Explorer.
- Close Visual Studio.

**Results:** After completing this exercise, you should have added a custom action to the project, written code to use the custom action, and deployed the app.

## Lesson 3

# Customizing the SharePoint List User Interface

Client-side rendering enables you to customize SharePoint user interface components by using JavaScript on the client as opposed to on the server. In this lesson, you will learn how to modify the appearance and behavior of list views and forms by using client-side rendering.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe client-side rendering.
- Use client-side rendering to customize the default list view.
- Use client-side rendering to create custom views.
- Use client-side rendering to customize list forms.
- Use client-side rendering to apply conditional formatting.
- Use jQuery UI in client-side rendering code.

### Introduction to Client-Side Rendering

In SharePoint 2010, you use XSLT to customize the appearance or behavior of list views, fields, and forms. In SharePoint 2013, you can now use client-side rendering in apps for SharePoint to use client-side JavaScript along with HTML to customize these interfaces. Client-side rendering provides advantages over XSLT:

- Performance – Because the rendering is performed on the client side, not on the server, pages can load faster and work on the server is minimized.
- Flexibility – You can perform customizations on a single field or an entire view using either your own JavaScript functions or libraries such as jQuery.
- Standard development language – Most web developers will already be familiar with JavaScript, resulting in reduced development time for your customizations.

There are two high-level tasks that you must perform to implement client-side rendering of list views in SharePoint:

1. Create a JavaScript code file containing the customization code.
2. Link the JavaScript code file to the list view.

The code that you write in your JavaScript file can range from adding HTML formatting to a list field to using custom formatting from the jQuery UI library.

- Advantages of client-side rendering over using XSLT:
  - Performance
  - Flexibility
  - Standard development language
- Two high-level tasks:
  - Create a JavaScript code file
  - Link the JavaScript code file to the list view

## Using Client-Side Rendering to Customize the Default List View

The first task when using client-side rendering for a list view is to develop the JavaScript code file that will contain the customization logic. In an app for SharePoint project, you add a JavaScript file to the Scripts folder and write your functions in that file.

The following code example shows a simple function that will display the Description field in the list in italics.

### Code to Format the List Field

```
(function () {
 var overrideCtx = {};
 overrideCtx.Templates = {};
 overrideCtx.Templates.Fields = {'Description' : { 'View' :
 '<i><#=ctx.CurrentItem.Description#></i>' }
 };
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();
```

In the preceding example, the **overrideCtx** variable holds the current context of the list item, and the **Templates.Fields** property enables you to access individual list fields. In this example, the function is changing the **Description** item to be italicized. The **RegisterTemplateOverrides** function call registers the templates for your list to use.

The second task is to link your JavaScript code to your list view. In Visual Studio, in the **Schema** file for the list, locate the **View** element that has the **DefaultView** attribute set to **TRUE**. This is the default view for your list. The **View** element has a child element named **JSLink**, which by default contains the text **clienttemplates.js**. Change this file name to the relative path and file name of your JavaScript code file.

The following XML example shows how to reference a JavaScript file named *ItalicCustomization.js* to modify the default view of a list.

### XML to Link the JavaScript Code to the List View

```
<View BaseViewID="1" Type="HTML" WebPartZoneID="Main"
 DisplayName="$Resources:core,objectiv_schema_mwsidcamlid24;" DefaultView="TRUE"
 MobileView="TRUE" MobileDefaultView="TRUE" SetupPath="pages\viewpage.aspx"
 ImageUrl="/_layouts/15/images/generic.png?rev=23" Url="AllItems.aspx">
 <Toolbar Type="Standard" />
 <XslLink>main.xsl</XslLink>
 <JSLink>~site/Scripts/ItalicCustomization.js</JSLink>
 <RowLimit Paged="TRUE">30</RowLimit>
 <ViewFields>
 <FieldRef Name="LinkTitle"></FieldRef><FieldRef Name="Description"> </FieldRef>
 </ViewFields>
</View>
```

After you complete these two tasks, your list view will display the **Description** field in italic font.

- Formatting list fields:
 

```
(function () {
 var overrideCtx = {};
 overrideCtx.Templates = {};
 overrideCtx.Templates.Fields = {'Description' : { 'View' :
 '<i><#=ctx.CurrentItem.Description#></i>' }
 };
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();
```
- Linking the JavaScript code to the list view:
 

```
<JSLink>~site/Scripts/ItalicCustomization.js</JSLink>
```

## Using Client-Side Rendering to Create Custom Views

In addition to customizing the default list view, you can also use client-side rendering to create new custom views and forms.

First, you create a new **View** element in the Schema file with a unique **BaseViewID** attribute. You then use the **JSLink** element of the view to link the JavaScript code file with the new view.

### Create the View Element

```
<View BaseViewID="2" Type="HTML"
WebPartZoneID="Main"

DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24;" MobileView="TRUE"
MobileDefaultView="TRUE" SetupPath="pages\viewpage.aspx"
ImageUrl="/_layouts/15/images/generic.png?rev=23" Url="CustomView.aspx">
 <Toolbar Type="Standard" />
 <XslLink>main.xsl</XslLink>
 <JSLink>~site/Scripts/ItalicCustomization.js</JSLink>
 <RowLimit Paged="TRUE">30</RowLimit>
 <ViewFields>
 <FieldRef Name="LinkTitle"></FieldRef>
 <FieldRef Name="Description"></FieldRef>
 </ViewFields>
</View>
```

```
• Creating the View element:
<View BaseViewID="2" Type="HTML" ... Url="CustomView.aspx">
...
<JSLink>~site/Scripts/ItalicCustomization.js</JSLink>
...
</View>

• Linking the function to the new view:
(function () {
...
 overrideCtx.BaseViewID = 2;
 overrideCtx.ListTemplateType = 10001;
...
})();
```

In your JavaScript code, you also need to specify to apply this override to the new view by setting the **BaseViewID** property of the context to the value of the attribute in the schema and setting the **ListTemplateType** property of the context to the **Type** attribute of your **ListTemplate** element in the **Elements** file belonging to the list.

### JavaScript Code to Link the Function to the New View Definition

```
(function () {
 var overrideCtx = {};
 overrideCtx.Templates = {};

 overrideCtx.BaseViewID = 2;
 overrideCtx.ListTemplateType = 10001;

 overrideCtx.Templates.Fields = {
 'Description': { 'View' : '<i><#>ctx.CurrentItem.Description#</i>' }
 };
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();
```

You can use the **Current View** drop-down list on the **List** ribbon to change to the new view for testing purposes, and in production you can use a custom action to provide easier access to the view.

## Using Client-Side Rendering to Customize List Forms

You can also use client-side rendering to customize the list forms, that is, the New form, the Edit form, and the Display form.

These forms are defined in the **Forms** element in the **Schema** file for the list and have a **JSLink** attribute that you can modify to reference your customization code.

### Modify the Form Element

```
<Form Type="NewForm" Url="NewForm.aspx"
 JSLink="~/site/ScriptsCustomize.js"
 SetupPath="pages\form.aspx"
 WebPartZoneID="Main" />
```

- Creating the View element:  

```
<Form BaseViewID="NewForm" Type="HTML"
 Url="NewForm.aspx"
 JSLink="~/site/ScriptsCustomize.js" >
```
- Linking the function to the new view:  

```
(function () {
 ...
 overrideCtx.BaseViewID = "NewForm";
 overrideCtx.ListTemplateType = 10000;
 ...
})();
```

In your JavaScript code, you create a function to customize the view. You need to set the **BaseViewID** property of the context to the same value as the **Type** attribute of the **Form** element, and set the **ListTemplateType** to the property of the context to the **Type** attribute of your **ListTemplate** element in the **Elements** file belonging to the list. You can then modify the fields on the form to meet your requirements. The following example shows how to modify the title of the form.

### Code to Modify the New Form

```
function modifyNewForm() {
 var overrideCtx = {};
 overrideCtx.Templates = {};

 overrideCtx.BaseViewID = "NewForm";
 overrideCtx.ListTemplateType = 10000;

 overrideCtx.Templates.Fields = {
 'Title': { 'NewForm': 'My Form' }
 };
 SPCClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
}();
```

## Using Client-Side Rendering to Apply Conditional Formatting

In addition to using client-side rendering to manipulate the HTML in a view, you can also call more complex functions to perform tasks such as conditional formatting. By accessing information stored in the **ctx.CurrentItem** object, you can perform actions such as calculations and conditional logic to further customize a view.

The following code example shows how to use a function named **getColor** to determine the color in which to display the progress bar depending on the due date for the task and the percentage complete.

```
function progressBar(ctx) {
 var percentage = ctx.CurrentItem.PercentComplete;
 var percentageNr =
 ctx.CurrentItem.PercentComplete.replace("%", "");

 var duedate = new Date(ctx.CurrentItem.DueDate);

 var color = getColor(percentageNr, duedate);
 return "<div style=\"background: #F3F3F3;
 display: block; height: 20px; width: 100px;\"><div
 style=\"background: " + color + "; height: 100%;
 width: " + percentageNr + "%;\"></div></div> " +
 percentage + "%";
}
```

## Applying Conditional Formatting

```
(function () {
 // Initialize the variable that store the overrides objects.
 var overrideCtx = {};
 overrideCtx.Templates = {};

 // Assign functions or strings to the header, footer and item.
 overrideCtx.Templates.Header = "";
 overrideCtx.Templates.Footer = "";
 overrideCtx.Templates.Fields = { 'PercentComplete': { 'View': progressBar } };

 // Register the template overrides.
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();

function progressBar(ctx) {
 var percentage = ctx.CurrentItem.PercentComplete;
 var percentageNr = ctx.CurrentItem.PercentComplete.replace(" %", "");

 var duedate = new Date(ctx.CurrentItem.DueDate);

 var color = getColor(percentageNr, duedate);
 return "<div style=\"background: #F3F3F3; display:block; height: 20px; width: 100px;\"><div
style=\"background: " + color + "; height: 100%; width: " + percentageNr + "%;\"></div></div> " +
percentage + "%";
}

function getColor(progress, duedate) {
 var datenow = new Date();
 if (parseInt(progress) < 100 && duedate.getTime() < datenow.getTime())
 return "Red";
 else
 return "Green";
}
```

## Demonstration: Applying Conditional Formatting

In this demonstration, you will see how to apply conditional formatting to a list view.

### Demonstration Steps

- Start the **20488B-LON-SP-14** virtual machine.
- Log on to the **LONDON** machine as **CONTOSO/Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Visual Studio 2012**.
- On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
- In the **Open Project** dialog box, browse to the **E:\Democode** folder, click **ProgressBarApp.sln**, and then click **Open**.
- In Solution Explorer, expand **Custom Tasks**, and then double-click **Schema.xml**.
- Locate the **View** element with the **BaseViewID="1"** attribute and point out the **JSLink** child element that links the JavaScript file to the view.
- In this **View** element, point out the contents of the **ViewFields** element, which defines which fields to display in the view.
- In Solution Explorer, expand **Scripts**, and then double-click **Customizelt.js**.
- In the main function, point out that the **PercentComplete** field is calling the **progressBar** function.



- In the **progressBar** function, point out that the function performs the following tasks:
  - a. Removes the percentage sign from the **PercentComplete** property and stores it and the **DueDate** property in variables.
  - b. Passes the variables to the **getColor** function, which determines whether to display the progress bar as red or green.
- In the **getColor** function, point out the **if** block, which determines the color to use.
- On the **DEBUG** menu, click **Start Debugging**.
- In Internet Explorer, point out that the overdue, but unfinished, task displays in red and the current task displays in green.
- Click **new item**.
- On the **EDIT** page, in the **Task Name** box, type **Write code**.
- By the **Due Date** box, click the **Calendar** icon, click yesterday's date.
- Click **SHOW MORE**, in the **% Complete** box, type **100**, and then click **Save**.
- Point out that the task displays in green.
- Close Internet Explorer and then close Visual Studio.

## Using jQuery UI In Client-Side Rendering Code

You can further extend the customization of your list views by calling functions in the jQuery UI JavaScript library. jQuery UI is a small, fast, feature-rich JavaScript library containing functions for navigating HTML documents, manipulating HTML documents, handling events, and animation. For example, you can use the jQuery UI **tabs** function to display list information in a tabbed format or you can use the **accordion** function to display list information in a collapsible list. You can also use the jQuery CSS to apply custom formatting to your jQuery elements to display them in a user-friendly manner.

1. Download the jQuery UI files
2. Add the jQuery UI files to your project
3. Add the jQuery UI files to your pages
4. Create a JavaScript code file that uses jQuery UI
5. Modify the JSLink element to reference your code file



**Additional Reading:** For more information about the functionality available in the jQuery UI library, see *jQuery user interface* at <http://go.microsoft.com/fwlink/?LinkID=311877> (this URL redirects to a third-party website).

There are five key tasks to perform when using functionality contained in the jQuery UI library and CSS:

1. Download the required jQuery library and theme from <http://go.microsoft.com/fwlink/?LinkID=311877> (this URL redirects to a third-party website).
2. Add the jQuery UI code, CSS, and images to your project.
3. Add jQuery and jQuery UI to your pages.
4. Create a JavaScript code file that calls the appropriate jQuery UI function, applies the style sheet, and creates your view.

5. In your list schema file, modify the JSLink element to point to your new JavaScript code file.

### Downloading the jQuery files

On the jQuery website, click the **Download** link, which enables you to select the functions and theme that you want to use in your SharePoint application. Select only the functions and themes that you actually require, because the downloader then generates a custom jQuery file containing only the code, style sheet, and images that you need.

### Adding the jQuery UI files to your project

After you download your custom jQuery file, you can extract the files contained in the download and place them in your application folders. Then in Visual Studio, you can use Solution Explorer to add the files to your project.

### Adding jQuery and jQueryUI to your pages

You can use a **CustomAction** element to load the jQuery and jQueryUI libraries in your application. Set the **Location** attribute to **ScriptLink**, and set the **ScriptSrc** attribute to the relative path of the two files.

The following XML example shows how to create the **CustomAction** element for jQuery files stored in the Scripts folder.

#### CustomAction Element

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction
 ScriptSrc="~Site/Scripts/jquery-1.9.1.min.js"
 Location="ScriptLink"
 Sequence="10">
 </CustomAction>
 <CustomAction
 ScriptSrc="~Site/Scripts/jquery-ui-1.10.3.custom.js"
 Location="ScriptLink"
 Sequence="20">
 </CustomAction>
</Elements>
```

### Creating a JavaScript code file

In your code file, create an anonymous function that loads your style sheet and uses the properties of the list context to generate your output, and then in the **document.ready** function, call the appropriate method of the jQuery UI object.

The following code example shows how to use the jQuery UI accordion to display a list in a collapsible manner. The accordion code expects an **h3** element followed by a **div** element, so the JavaScript generates this for each item in the list.

## Using the Accordion

```
// Function to process an accordion item.
window.CD = window.CD || {};
window.CD.accordionItem = {
 customItemHtml: function (ctx) {
 var accordionItemHtml = "<h3>" + ctx.CurrentItem.CustomerName + "</h3>";
 accordionItemHtml += "<div>" + ctx.CurrentItem.CallDetails + "</div>";
 return accordionItemHtml;
 }
};

(function () {
 // Load the jQuery CSS file.
 var appWebUrl = decodeURIComponent(getQueryStringParameter("SPAppWebUrl"));
 var CSSFolder = appWebUrl + "/SiteAssets/";
 loadcssfile(CSSFolder + "jquery-ui-1.10.3.custom.css", "css");

 // Initialize the variable that store the objects.
 var overrideCtx = {};
 overrideCtx.Templates = {};
 overrideCtx.Templates.Header = "<div id=\"\"accordion\"\">";
 overrideCtx.Templates.Item = window.CD.accordionItem.customItemHtml;
 overrideCtx.Templates.Footer = "</div>";

 overrideCtx.BaseViewID = 1;
 overrideCtx.ListTemplateType = 11000;

 // Register the template overrides.
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();

$(document).ready(function () {
 $("#accordion").find('#scriptBodyWPQ1').remove();
 $("#accordion").accordion({ heightStyle: "content" });
});
```



**Note:** The `accordion.find.remove` method call is required to remove an extraneous open tag that SharePoint creates.

## Modifying the JSLink element

After you create your JavaScript file, in the **Schema** file for your list, modify the **JSLink** element of the relevant **View** element to reference the file in the same way that you would for any other client-side rendering.

## Lab B: Using jQuery to Customize the SharePoint List User Interface

### Scenario

You have a list of customers with details about each one. When this list uses the default list view, it is not easy to read the information and you can foresee that as more data is added to the list, the worse the problem will become. You decide to use the accordion functionality available in jQuery UI to display the list and use the style sheet to apply custom styling to the list.

### Objectives

After completing this lab, you will be able to:

- Create a custom list view by using jQuery UI.
- Apply a custom style sheet from jQuery UI.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-14
- User name: CONTOSO\administrator
- Password: Pa\$\$w0rd

### Exercise 1: Creating a Custom List View

#### Scenario

In this exercise, you will use Visual Studio to write custom client-side code to display the list using the accordion functionality and reference this custom code from the default list view. When this is working, you will add code to load the jQuery UI style sheet to further enhance the view.

The main tasks for this exercise are as follows:

1. Create the Custom List View by Using jQuery UI
2. Apply a Custom Style Sheet from jQuery UI

#### ► Task 1: Create the Custom List View by Using jQuery UI

- Start the 20488B-LON-SP-14 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio, and then open **CustomerInfo.sln** from the **E:\Labfiles\Starter\CustomerInfo** folder.
- Run the app and review the customer information list.
- Add **jquery-1.7.1.js** and **jquery-ui.js** from the **E:\Labfiles\Starter\CustomerInfo\Scripts** folder to the **Scripts** folder in the project.
- Add a new module named **SiteAssets** to the project.
- Add **jquery-ui.css** from the **E:\Labfiles\Starter\SiteAssets** folder to the **SiteAssets** module in the project.
- Add a new folder named **Images** to the **SiteAssets** module in the project.
- Add all the files from the **E:\Labfiles\Starter\SiteAssets\images** folder to the **Images** folder in the project.
- Add a new module named **AddjQuery** to the project.

- Open **Elements.xml** from the module and add two **CustomAction** elements to the **Elements** element to load the **jquery-1.7.1.js** and **jquery-ui.js** script files from the **Scripts** folder.
- Save the **Elements.xml** file.
- Add a new JavaScript file named **CustomizeView.js** to the **Scripts** folder in the project.
- Add code to this file to process the accordion item, passing the **CustomerName** item from the list in an **h3** tag and the **Details** item from the list in a **div** tag.
- Add an anonymous function to the script file to create the view and register the template override.
- Add a **(document).ready** function to the script file to remove the extraneous **scriptBodyWPQ1** tag, and then call the **accordion** method.
- Open the schema file for the **CustomerInformation** list and locate the default view element.
- Modify the **JSLink** element to reference your code file.
- Run the app.
- When Internet Explorer loads, verify that the list now displays using the accordion theme.
- Click **Sara Davis** to expand her information.
- Close Internet Explorer.

### ► Task 2: Apply a Custom Style Sheet from jQuery UI

- Add the contents of the **CodeSnippets.txt** file in the E:\Labfiles\Starter\CustomerInfo folder to the end of the **CustomizeView.js** code file.
- At the beginning of the anonymous function in your code file, add the following lines of code to construct the path for the jQuery UI style sheet:

```
var appWebUrl = decodeURIComponent(getQueryStringParameter("SPAppWebUrl"));
var scriptFolder = appWebUrl + "/SiteAssets/";
Add the following line of code to load the style sheet:
loadcssfile(scriptFolder + "jquery-ui.css", "css");
```

- Retract the app from Internet Explorer.
- Run the app.
- When Internet Explorer loads, verify that the list now displays using the accordion style sheet.
- Click **Sara Davis** to expand her information.
- Close Internet Explorer.
- Deploy the app.
- From the Start screen, open Internet Explorer and browse to the URL where the app is deployed.
- Verify that the Customer Information list displays using the accordion theme and style sheet.
- Close Internet Explorer.
- Close Visual Studio.

**Results:** After completing this exercise, you should have implemented and tested a custom list view.

## Module Review and Takeaways

In this module, you have learned how to customize the user interface element in SharePoint 2013, from creating custom actions to add functionality to SharePoint elements, to displaying helpful information to your users, to customizing the list user interface.

### Review Question(s)

**Question:** You want to add a custom button to the ribbon in a sandboxed solution, but you can't find the **Ribbon Custom Item** in the **Add New Item** dialog box. What is the problem?

### Test Your Knowledge

Question	
You want to provide the user with additional information about an element in the SharePoint user interface without interrupting their work. Which functionality should you use?	
Select the correct answer.	
<input type="checkbox"/>	Status bar message
<input type="checkbox"/>	Notifications
<input type="checkbox"/>	Dialog box
<input type="checkbox"/>	Callouts

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
You can only use client-side rendering to customize list views.	

# Module 15

## Working with Branding and Navigation

### Contents:

Module Overview	15-1
Lesson 1: Creating and Applying Themes	15-2
Lesson 2: Branding and Designing Publishing Sites	15-9
Lab A: Branding and Designing Publishing Sites	15-18
Lesson 3: Tailoring Content to Platforms and Devices	15-24
Lesson 4: Configuring and Customizing Navigation	15-31
Lab B: Configuring Farm-Wide Navigation	15-41
Module Review and Takeaways	15-46

## Module Overview

SharePoint Server 2013 introduces many changes in the areas of branding, design, and navigation. The site design process for publishing sites has been completely overhauled to enable web designers to create SharePoint sites by using standard web technologies, such as HTML, CSS, and JavaScript. The new publishing infrastructure enables you to adapt your sites for users of different devices, by associating master pages with device channels. Image renditions enable you to reuse images efficiently at multiple resolutions. Finally, the new managed navigation model enables you to drive site navigation through managed metadata term sets, which creates opportunities to populate publishing pages with dynamic, search-driven content. In this module, you will learn how to leverage and customize these new features from a developer perspective.

### Objectives

After completing this module, you will be able to:

- Create and apply themes to SharePoint sites.
- Create publishing site design assets such as master pages and page layouts.
- Use device channels and image renditions to adapt content for different devices.
- Configure and customize the navigation experience for publishing sites.

## Lesson 1

# Creating and Applying Themes

In SharePoint, themes have long provided a way for site owners to customize the look and feel of their sites without resorting to extensive editing of master pages and CSS files. SharePoint 2013 introduces an entirely new theming model that makes this process easier and more intuitive. In this lesson, you will learn about the different components of the SharePoint 2013 theme model. You will learn how administrators can mix and match theme components to find the right look and feel for their sites, and you will learn how to create and deploy custom theme components to provide a genuinely unique look and feel.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe the components of the theme model in SharePoint 2013.
- Create custom color palettes.
- Create custom font schemes.
- Deploy and apply theme components programmatically.

### The SharePoint 2013 Theme Model

SharePoint 2013 includes an entirely new theme model. In place of the Microsoft PowerPoint-based .thmx files used in SharePoint 2010, SharePoint 2013 introduces the concept of a *composed look*. A composed look consists of the following resources:

- *A site master page*. The master page uses HTML to define the layout of the page as a whole, with placeholders for dynamic content. The master page also includes links to CSS files and JavaScript files that define the appearance and behavior of the site.
- *A color palette*. The color palette defines the combination of colors that are used in the site. Color palettes are defined in XML-based .spcolor files. When you use a color palette as part of a composed look, the .spcolor file overrides colors specified in the CSS files associated with the master page.
- *A font scheme*. The font scheme defines the collection of fonts that are used in the site, such as for headings, titles, and body text. Font schemes are defined in XML-based .spfont files. When you use a font scheme as part of a composed look, the .spfont file overrides fonts specified in the CSS files associated with the master page.
- *A background image*. The background image is used as the page background on all pages in the site.

You will explore all of these items in more detail later in this lesson.



**Note:** The terms “themes,” “designs,” and “composed looks” are often used interchangeably in product documentation.

- New theme model in SharePoint 2013
- Composed looks consist of:
  - A master page
  - A color palette (.spcolor file)
  - A font scheme (.spfont file)
  - A background image



To create a new composed look, you add a new item to the **Composed Looks** gallery in the root site for the site collection. You must provide a name for the composed look and specify site-relative URLs for the master page, color palette, font scheme, and background image files. You can create all these items yourself, or you can mix and match existing items. SharePoint includes 7 built-in font schemes and 32 color palettes that you can use in your own composed looks. These are stored in the **Themes** gallery in the root site for the site collection.

In contrast to many branding and navigation features, which are only available on SharePoint publishing sites, the SharePoint 2013 theme model is available on all SharePoint sites. Themes are the primary tool for branding non-publishing SharePoint sites.

## Demonstration: Building a Composed Look

The instructor will now demonstrate how site administrators can build a composed look interactively from the browser UI. The administrator can select from available background images, color palettes, font themes, and master pages. SharePoint enables the administrator to preview the effects of every change.

### Demonstration Steps

- Start the 20488B-LON-SP-15 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Internet Explorer, and browse to **http://team.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the **Settings** menu, click **Change the look**.
- On the **Change the look** page, scroll through the available designs. Explain that these are the built-in composed looks that ship with SharePoint 2013.
- Click **Nature**.
- Explain that the preview image is generated by the master page preview file. You can create a preview file, with a .preview file name extension, for every master page you add to the master page gallery.
- On the left of the page, under **Start over**, explain that the image is the background image for the composed look. Administrators can change the background image to an image in a library on the site, or upload a new image.
- Under **Colors**, click the drop-down arrow. Explain that these are the color palettes that are currently available on the site.
- Hover over several of the color palettes in the list, and draw attention to how the preview image is updated accordingly.
- Click the second color palette from the top.
- Under **Site layout**, click the drop-down arrow. Explain that these are the master pages that are currently available for use with composed looks.
- Hover over **Oslo**, draw attention to how the preview image is updated, and then click **Seattle**.
- Under **Fonts**, click the drop-down arrow. Explain that these are the font themes that are currently available on the site.
- Click **Blueprint MT Pro**, and draw attention to how the preview image is updated.

- At the upper-right corner of the page, click **Try it out**.
- On the preview page, click **No, not quite there**.
- On the **Fonts** drop-down menu, click **Century Gothic**, and then click **Try it out**.
- On the preview page, click **Yes, keep it**.
- On the **Site Settings** page, click the site logo to return to the home page.
- Point out how the new composed look is applied to every aspect of the home page.
- On the **Settings** menu, click **Site settings**.
- Under **Web Designer Galleries**, click **Composed looks**.
- On the **Composed Looks** page, click **new item**.
- Point out how you can create a composed look manually by specifying a master page URL, a theme (color palette) URL, a background image URL, and a font scheme URL.
- Close Internet Explorer.

## Creating Custom Color Palettes

In SharePoint 2013, color palettes are defined by XML-based .spcolor files. An .spcolor file defines colors for various elements on a SharePoint site by associating each element with a six-digit red green blue (RGB) or eight-digit alpha red green blue (ARGB) hexadecimal value.

The following code example shows an excerpt from an .spcolor file:

### Defining a Color Palette

```
<s:colorPalette isInverted="false"
 previewSlot1="BackgroundOverlay"
 previewSlot2="BodyText"
 previewSlot3="AccentText"
 xmlns:s="http://schemas.microsoft.com/sharepoint">
 <s:color name="BodyText" value="444444" />
 <s:color name="SubtleBodyText" value="777777" />
 <s:color name="StrongBodyText" value="262626" />
 <s:color name="DisabledText" value="B1B1B1" />
 <s:color name="SiteTitle" value="262626" />
 ...
</s:colorPalette>
```

- Create an XML-based .spcolor file
- Define RGB or ARGB color values for various site components

```
<s:colorPalette isInverted="false"
 previewSlot1="BackgroundOverlay"
 previewSlot2="BodyText"
 previewSlot3="AccentText"
 xmlns:s="...">
 <s:color name="BodyText" value="444444" />
 <s:color name="SubtleBodyText" value="777777" />
 <s:color name="StrongBodyText" value="262626" />
 <s:color name="DisabledText" value="B1B1B1" />
 <s:color name="SiteTitle" value="262626" />
 <s:color name="WebPartHeading" value="444444" />
 ...
</s:colorPalette>
```

Within the parent **colorPalette** element, you use a **color** element to define the color for each element. You use a six-digit RGB value to specify a simple color, or an eight-digit ARGB value if you want to include a degree of opacity in the color. In the parent **colorPalette** element, you can use the three **previewSlot** attributes to specify the color elements by name that you want to use to represent your color palette on the **Colors** menu when an administrator is building a composed look.

Creating color palettes offer many advantages over creating custom CSS files:

- Master pages are tightly bound to CSS files. By contrast, you can choose from multiple color palettes to customize the look of a site without editing the master page.

- Color palettes are often easier to create than complete CSS files. Color palettes reduce the need for you to identify CSS element classes and to order your CSS classes appropriately.
- Color palettes can be used as part of a composed look, which enables a flexible "mix and match" approach to styling a site.

Manually creating an .spcolor file would be a laborious process, because a typical .spcolor file includes approximately 90 individual **color** elements. To make the process easier, Microsoft has introduced a free downloadable utility—the SharePoint Color Palette Tool—that you can use to build color palettes interactively.



**Reference Links:** For more information and to download the SharePoint Color Palette Tool, see *SharePoint Color Palette Tool* at <http://go.microsoft.com/fwlink/?LinkID=311880>.

After you create an .spcolor file, upload it to the **Themes** gallery on the root site of your site collection to make it available for use in composed looks.

## Creating Custom Font Schemes

Font schemes are defined by XML-based .spfont files. An .spfont file defines the fonts that SharePoint should use for the following elements of a site:

- Title text
- Navigation links
- Small headings
- Headings
- Large headings
- Body text
- Large body text

- Create an XML-based .spfont file
- Define fonts for the following types of text:
  - Title text
  - Navigation links
  - Small headings
  - Headings
  - Large headings
  - Body text
  - Large body text

The .spfont file consists of a top-level **fontScheme** element that contains a collection of **fontSlot** elements. Each **fontSlot** element represents one of the preceding seven categories.

The following code example shows an excerpt from an .spfont file:

### Defining a Font Scheme

```
<s:fontScheme name="Bodoni"
 previewSlot1="title"
 previewSlot2="body"
 xmlns:s="http://schemas.microsoft.com/sharepoint/">
 <s:fontSlots>
 <s:fontSlot name="title">
 <s:latin typeface="Bodoni Book"
 eotsrc="/_layouts/15/fonts/BodoniBook.eot"
 woffsrc="/_layouts/15/fonts/BodoniBook.woff"
 ttfsrc="/_layouts/15/fonts/BodoniBook.ttf"
 svgsrcc="/_layouts/15/fonts/BodoniBook.svg"
 largeimgsrc="/_layouts/15/fonts/BodoniBookLarge.png"
 smallimgsrc="/_layouts/15/fonts/BodoniBookSmall.png" />
 <s:ea typeface="" />
 <s:cs typeface="Segoe UI Light" />
 <s:font script="Arab" typeface="Segoe UI Light" />
 <s:font script="Deva" typeface="Nirmala UI" />
 ...
 </s:fontSlot>
 <s:fontSlot name="navigation">...</s:fontSlot>
 <s:fontSlot name="small-heading">...</s:fontSlot>
 <s:fontSlot name="heading">...</s:fontSlot>
 <s:fontSlot name="large-heading">...</s:fontSlot>
 <s:fontSlot name="body">...</s:fontSlot>
 <s:fontSlot name="large-body">...</s:fontSlot>
 </s:fontSlots>
</s:fontScheme>
```

As you can see, the structure of an .spfont file is more complex than an .spcolor file. For each font slot, you must specify:

- An **s:latin** element. This defines the properties of the font using the **LatinFont** class.
- An **s:ea** element. This defines the properties of the font using the **EastAsianFont** class.
- An **s:cs** element. This defines the properties of the font using the **ComplexScriptFont** class.

Each of these elements must include a **typeface** element, which specifies the font family. For non-standard fonts, the elements can include additional resources required to define or preview the font, such as:

- An Embedded Open Type (EOT) font file.
- A Web Open Font Format (WOFF) file.
- A TrueType Font (TTF) file.
- A Scalable Vector Graphics (SVG) font file.
- Large and small preview images of the font text.

A **fontSlot** element can also include multiple optional **s:font** elements, which are used to map particular types of script, such as Arabic or Greek, to particular typefaces.

Because .spfont files can be complex to create manually, the best approach is often to copy an existing .spfont file and change only the properties you need. After you create an .spfont file, upload it to the **Themes** gallery on the root site of your site collection to make it available for use in composed looks.

## Deploying Custom Themes

SharePoint sites include various *galleries* to store components for theming. In SharePoint terminology, a gallery is just a list or library that is exposed through the **\_catalogs** virtual directory on each site. SharePoint 2013 includes the following galleries for theming components:

- *Master page gallery.* The master page gallery stores master pages, master page previews, and various resources for master pages, such as page layouts, display templates, JavaScript files, and CSS files. You can access the master page gallery for each site at the site-relative URL **\_catalogs/masterpage**.
- *Theme gallery.* The theme gallery stores color palettes (.spcolor files) and font schemes (.spfont files), as well as generated CSS output for each theme. You can access the theme gallery for each site at the site-relative URL **\_catalogs/theme**.
- *Design gallery.* The design gallery, also known as the Composed Looks list, stores composed looks. Composed looks are not files; each composed look is simply a list item that defines a title, a name, and URLs for the master page, the color palette, and the font scheme. You can access the design gallery for each site at the site-relative URL **\_catalogs/design**.

- Galleries:
  - Master page gallery (\_catalog/masterpage)
  - Theme gallery (\_catalog/theme)
  - Design gallery (\_catalog/design)
- To deploy components manually:
  - Upload master pages and preview files to the master page gallery
  - Upload .spcolor files and .spfont files to the 15 folder in the theme gallery



**Note:** Master pages and master page previews are covered later in this module.

### Deploying Theme Components Manually

In many scenarios, it is perfectly acceptable to deploy theme components manually by uploading each component to the relevant gallery on the SharePoint site:

- Upload custom master pages and the associated preview files to the master page gallery.
- Upload custom .spcolor files and .spfont files to the **15** folder in the theme gallery.
- Create new composed looks by adding a list item to the design gallery.

To make your theme components available across a site collection, upload each component to the relevant gallery on the root site of the site collection.



**Note:** When you use a color palette or a font scheme in a composed look, SharePoint generates CSS files from the source .spcolor and .spfont files. Within the theme gallery, the **15** folder stores the source .spcolor and .spfont files, and the **Themed** folder stores the corresponding CSS files generated by SharePoint. For this reason, you should always add your .spcolor and .spfont files to the **15** folder within the theme gallery.

## Deploying And Applying Theme Components Programmatically

Deploying theme components manually makes sense if you are providing components for administrators to choose from when they build their own composed looks. By contrast, if you want to apply a composed look to multiple sites in your organization, you should use a programmatic approach. You can use the following high-level process to deploy and apply a theme:

1. Create a new site-scoped Feature.
2. Within the Feature, use **Module** elements to deploy color palettes, font schemes, and master pages.
3. Add a feature receiver class to the Feature.
4. In the feature receiver class, apply the theme to the site when the feature is activated.

The following code example illustrates how to apply a theme from a feature receiver class:

### Applying Theme Components Programmatically

```
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
 using (var site = properties.Feature.Parent as SPSite)
 {
 using (var web = site.RootWeb)
 {
 // Get the color palette.
 SPFile fileColors = web.GetFile("_catalog/theme/15/contoso.spcolor");

 // Get the font scheme.
 SPFile fileFonts = web.GetFile("_catalog/theme/15/contoso.spfont");

 // Get the background image URI.
 var backgroundImage = new Uri("http://www.contoso.com/images/background.png");

 // Create a new SPTheme object.
 SPTheme theme = SPTheme.Open("Contoso Site Design", fileColors, fileFonts,
backgroundImage);

 // Apply the theme to the site.
 theme.ApplyTo(web, true);
 }
 }
}
```



**Additional Reading:** For more information about deploying theme components, see *How to: Deploy a custom theme in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=311881>. In particular, this topic describes how to update the theme named Current in the Composed Looks list.

## Lesson 2

# Branding and Designing Publishing Sites

In a SharePoint deployment, publishing sites are typically used for marketing and communication. You might use an Internet-facing publishing site to reach potential customers, or an intranet site to provide information to employees across the organization. In these types of sites, the branding and design of the site is usually a high priority, because the look and feel of a website is a major factor in how people perceive your organization.

SharePoint 2013 introduces a completely new model for branding and designing publishing sites. Web designers can now create assets for SharePoint sites, such as master pages and page layouts, using standard web technologies such as HTML, CSS, and JavaScript.

### Lesson Objectives

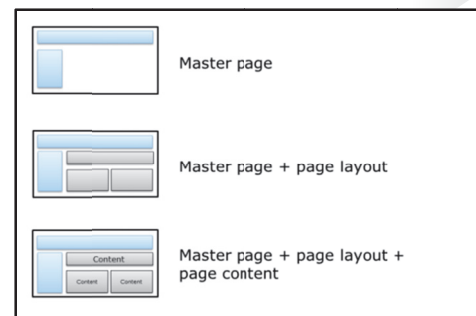
After completing this lesson, you will be able to:

- Describe the page model for SharePoint publishing sites.
- Explain the purpose and capabilities of the Design Manager.
- Create master pages for SharePoint sites.
- Create page layouts for SharePoint sites.
- Import and export design packages.

### The Publishing Site Page Model

In a SharePoint publishing site, the layout and content of publishing pages is determined by three components:

- *Master pages.* A master page defines the page components that are common to all pages in the publishing site, such as navigation menus, sign-in controls, and search boxes. The master page also defines the look and feel of the site as a whole, through links to CSS files and JavaScript files. Master pages include placeholders, into which pages that reference the master page can insert content.
- *Page layouts.* A page layout is effectively a template for a particular type of page. A page layout is an ASPX page that includes field controls to display different types of content, such as text or images. When a page is rendered, the page layout is inserted into one or more content placeholders in the master page.
- *Page content.* Content for publishing pages is stored as data in a SharePoint list, where each row of list data represents a publishing page. When a page is rendered, the content from each field in the list item is displayed by the corresponding field control in the page layout.



### Understanding Master Pages

A SharePoint master page is an ASP.NET master page. There are no differences between how master pages work in SharePoint and how master pages work in any other ASP.NET web application. A SharePoint master page contains the following types of content:

- HTML elements that define the overall structure of the page
- Various SharePoint and ASP.NET controls, such as the SharePoint **Welcome** menu control and the ASP.NET **ScriptManager** control
- Links to CSS style sheets and JavaScript files
- Several ASP.NET **ContentPlaceHolder** controls, into which page layouts or Web Part pages can insert content
- Default content for some of the **ContentPlaceHolder** controls

## Understanding Page Layouts

A page layout is an ASPX page. Because page layouts are designed to work with master pages, all the content in page layouts is contained within ASP.NET **Content** controls. Each **Content** control specifies a **ContentPlaceHolderID** value, which indicates the **ContentPlaceHolder** control in the master page into which the content should be inserted when the page is rendered.

Each page layout is based on a content type. The content type defines the fields that must be included on the page. For example, suppose you want to create a page layout that includes a title, two columns of text, and an image. The corresponding content type might include fields named **Title**, **Column1Text**, **Column2Text**, and **PageImage**. The page layout would then include field controls to render the values of each of these four fields.

The following code example shows how field controls are placed within content controls in a page layout:

### Content Controls and Field Controls

```
<asp:Content ContentPlaceHolderID="PlaceHolderPageTitle" runat="server">
 <SharePoint:FieldValue id="PageTitle" FieldName="Title" runat="server" />
</asp:Content>
```

When a user creates a new page based on a particular page layout, they are simply creating a new list item with the relevant content type. The user provides values for each of the fields defined by the content type. The page layout and the master page together determine how the content provided by the user is rendered.

## Design Manager and the Site Design Process

In previous versions of SharePoint, the site design process was challenging. SharePoint developers often lacked expertise in web design, whereas web designers often lacked the requisite technical knowledge of SharePoint. SharePoint 2013 introduces an entirely new site design process that enables a cleaner separation of design and development.

### Designing Site Assets With Standard Web Technologies

SharePoint 2013 enables you to develop site assets, such as master pages, page layouts, and display templates for search results, using standard web technologies. For example, to create a new SharePoint 2013 master page, you start by creating an HTML file, together with associated resources such as CSS files and JavaScript files. Essentially, this HTML file is a static version of your master page. It contains no SharePoint or ASP.NET markup at this stage, which means you can create your master page design in the

- Design site assets with standard web technologies:
  - HTML
  - CSS
  - JavaScript
- Use Design Manager to:
  - Convert HTML designs into SharePoint master pages
  - Create starter HTML files for page layouts
  - Generate markup snippets for SharePoint controls



web design tools of your choice. When you upload your HTML master page design to the master page gallery, you can use a new SharePoint 2013 component—the Design Manager—to convert your HTML design into a fully functional SharePoint master page.

## Working With The Design Manager

Design Manager is an integral part of SharePoint 2013. On a publishing site, you can access the Design Manager from the **Settings** menu on any SharePoint page. The Design Manager performs a variety of essential roles in the design process. For example, you can use the Design Manager to:

- Convert HTML files into SharePoint master pages.
- Generate markup snippets for various SharePoint components, such as navigation menus and sign-in controls, which designers can integrate into their master page designs.
- Generate starter HTML files for SharePoint page layouts, which designers can use as a basis for page designs.
- Preview master page designs with site content.
- Create and manage device channels, which enable you to apply different site designs to different consuming devices.



**Note:** Device channels are covered later in this module.

The core aim of Design Manager, and the new site design process in general, is to enable web designers to work with SharePoint sites without requiring an in-depth knowledge of SharePoint or ASP.NET development. Designers can now focus on creating compelling site designs without learning how to configure SharePoint placeholders and controls.



**Note:** Users must have at least the **Designers** permission level (or the **Design** permission in a custom permission level) to use Design Manager.

## Creating Master Pages

In SharePoint 2013, you can develop master pages as HTML files using the HTML editor of your choice. At various stages in the design process, you can use Design Manager to insert SharePoint functionality and preview how your master page will look with actual SharePoint content.

You can use the following high-level process to create a new master page:

1. Map the master page gallery as a network drive, so you can update files with ease during the design process.
2. Upload an HTML file, together with CSS style sheets and any required JavaScript resources, to the master page gallery.

- Use Design Manager to convert HTML files into master pages:
  1. Upload an HTML file to the master page gallery.
  2. Use Design Manager to convert the HTML file into a master page.
  3. Preview the master page in Design Manager.
  4. Use Design Manager to add snippets for SharePoint controls.
  5. Preview the master page in Design Manager.
  6. Edit the HTML and CSS files.
  7. Repeat steps 5–6 until the design is complete.

3. In Design Manager, convert the HTML file to a SharePoint master page. At this point, Design Manager will:
  - a. Add markup to your HTML file.
  - b. Generate a parallel .master file.
4. In Design Manager, preview the master page.
5. Use the Design Manager to add snippets for any SharePoint controls you require, such as the welcome menu and navigation controls.
6. In Design Manager, preview the master page.
7. In an HTML editor, make adjustments to the HTML file and CSS style sheets as required.
8. Repeat steps 6–7 until the design is final.



**Note:** If you prefer, you can start by using Design Manager to create a minimal master page, and then build your design around the HTML generated by SharePoint.

Site design is an iterative process, and you will typically repeat the cycle of editing HTML and previewing your changes in Design Manager multiple times. To gain an understanding of what Design Manager adds to your HTML master page, it is worth creating a simple HTML file, running the conversion process, and then viewing the changes (you will see an example of this in the next topic). Much of the additional content is commented out markup that is used by the conversion process. Most notably, Design Manager adds a **div** element with a **class** attribute value of **DefaultContentBlock**. This div element represents the area that page layouts or web part pages will fill. You should develop your master page design around this placeholder. After your design is complete, you should delete the **DefaultContentBlock** div before you use your master page.

When you use the Design Manager to convert an HTML file into a master page, SharePoint creates a relationship between your HTML file and the .master file it generates. Whenever you update the HTML file in the master page gallery, SharePoint will automatically make changes to the associated .master file to keep it synchronized.



**Additional Reading:** For more information about creating master pages, see *How to: Convert an HTML file into a master page in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=311882>.

## Demonstration: Developing Master Pages with Design Manager

The instructor will now demonstrate the following aspects of the master page development process:

- Mapping the master page gallery as a network drive.
- Creating a simple HTML file as the basis for a master page.
- Using Design Manager to convert an HTML file into a SharePoint master page.
- Using Design Manager to generate HTML snippets for SharePoint controls.
- Using Design Manager to preview changes to master the page.

## Demonstration Steps

- Connect to the 20488B-LON-SP-15 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open a File Explorer window, right-click **Computer**, and then click **Map network drive**.
- In the **Map Network Drive** dialog box, in the **Folder** box, type **http://publishing.contoso.com/\_catalogs/masterpage**, and then click **Finish**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- When the File Explorer window displays the contents of the master page gallery, close the window.
- On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
- On the **FILE** menu, point to **New**, and then click **File**.
- In the **New File** dialog box, click **HTML Page**, and then click **Open**.
- On the blank line after the opening **body** element, add the following markup:

```
<p>This is a very simple HTML file.</p>
```

- On the **FILE** menu, click **Save HTMLPage1.html As**.
- In the **Save File As** dialog box, under **Computer**, click **masterpage**.
- In the **File name** box, type **Contoso.html**, and then click **Save**.
- Close Visual Studio.
- On the Start screen, click **Internet Explorer**, and browse to **http://publishing.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- When the page loads, on the **Settings** menu, click **Design Manager**.
- On the list on the left side of the page, click **Edit Master Pages**.
- On the **Design Manager: Edit Master Pages** page, click **Convert an HTML file to a SharePoint master page**.
- In the **Select an Asset** dialog box, click **Contoso.html**, and then click **Insert**.
- On the list on the left side of the page, click **Edit Master Pages** again to refresh the list of pages.
- Without closing Internet Explorer, open a File Explorer window, and then click **masterpage**.
- Notice that the master page gallery now includes a file named **Contoso.master** in addition to the **Contoso.html** file.
- Right-click **Contoso.html**, point to **Open with**, and then click **Visual Studio 2012**.
- Review the content that SharePoint has added to the Contoso.html file, paying particular attention to the **ContentPlaceHolderMain** div toward the end of the file.
- Switch back to Internet Explorer.
- On the **Design Manager: Edit Master pages** page, click **Contoso**.
- On the master page preview, point out that the ribbon and the welcome menu are included on the page.

- At the top of the page, click **Snippets**.
- On the ribbon, in the **Navigation** group, click **Search Box**.
- On the right side of the page, briefly explore the options for customizing the search box.
- On the left side of the page, under the **HTML Snippet** box, click **Copy to Clipboard**.
- If the Internet Explorer dialog box appears, click **Allow access**.
- Switch back to Visual Studio.
- Add a new line immediately above the **DefaultContentBlock** div element, and then press Ctrl+V to paste the markup for a search box.
- On the **FILE** menu, click **Save All**, and then switch back to Internet Explorer.
- In Internet Explorer, on the master page preview page, click **Refresh**.
- Point out that the search box has been added to the page. Explain that you should use CSS to position the search box div where you want it to appear on the page.
- Close Internet Explorer and Visual Studio.

## Creating Page Layouts

A page layout is an ASP.NET content page. In other words, it is an ASPX file in which all content is contained with ASP.NET **Content** controls. Every **Content** control specifies a **ContentPlaceHolderID**, which indicates where the content should be inserted into the master page. Within these **Content** controls, a page layout uses field controls to render the contents of the page. You can use HTML and CSS to control how these field values are presented.

You can use the following high-level process to create a page layout:

1. Create a content type to underpin the page layout.
2. Create the page layout in Design Manager.
3. Edit the layout and style of the page layout HTML file.

### Creating Page Layout Content Types

Every page layout is underpinned by a content type. The content type determines the fields that the page layout should include. Typically, you add a field for each piece of content that you want to present on the page. For example, if you were creating a page layout for a pharmaceutical fact sheet, your fields might include the following:

- A product title
- A product byline
- A product image
- The revision number
- The publication date

1. Create a page layout content type:
  - Inherit from a page layout or publishing base type
  - Add a field for each individual piece of content
2. Use Design Manager to create the page layout:
  - ASPX and HTML files are created
  - Edit the HTML file
  - ASPX file is automatically synchronized
3. Add style sheet links within the **PlaceHolderAdditionalPageHead** control

- A short description
- A long description
- Body text column one
- Body text column two
- Footnotes
- Margin notes

Each of these fields has a corresponding publishing field type, such as **HTML**, **Image**, or **SummaryLinks**. Typically, when you create a content type for a page layout, you should inherit from one of the existing page layout content types and add or remove fields as required. You can also use the **Page** content type in the **Publishing Content Types** group as a base content type.



**Note:** You can create multiple page layouts based on the same content type. You can also create page layouts based on built-in page layout content types if they meet your needs.

### Creating a Page Layout In Design Manager

After you create or identify a suitable content type, you can use Design Manager to generate a starter page layout. When you use Design Manager to create a page layout, it will prompt you for two pieces of information:

- *The master page.* The **Content** controls in the page layout must match the **ContentPlaceholder** controls in the master page. Design Manager uses the master page you specify to work out what **Content** controls to add to the new page layout.
- *The content type.* Design Manager adds a SharePoint field control to the page layout for each field in the content type.

Design Manager adds two page layout files to the master page gallery: an .html file, and an .aspx file. The .html file is a static representation of the page layout, and the .aspx file is the actual page layout. This is similar to the coupling between the .html file and the .master file when you create a master page. Designers can edit the .html file to specify the presentation and style of the page layout. When you update the .html file in the master page gallery, SharePoint will automatically update the .aspx file to maintain synchronization.

### Styling a Page Layout

Because page layouts are ASP.NET content pages, all markup—including links to CSS files and JavaScript files—must be added to a **Content** control. SharePoint master pages provide a content placeholder control named **PlaceHolderAdditionalPageHead**, at the end of the **head** element in the master page. By adding CSS and JavaScript links in the **Content** control for **PlaceHolderAdditionalPageHead** in your page layout, you ensure that your links are added to the **head** element when the page is rendered. By contrast, if you add the links directly to the **head** element in your page layout HTML file, the links will be removed as part of the conversion process.

The following code example shows where to add CSS and JavaScript links in the page layout HTML file to ensure that they are added to the **head** element in rendered pages (the code to add is shown in black, existing code is shown in gray):

### Adding Resource Links to Page Layouts

```
<!--MS:<asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server">-->
<!--CS: Start Edit Mode Panel Snippet-->
<!--SPM:<%@Register Tagprefix="SharePoint" ... %>-->
<!--SPM:<%@Register Tagprefix="Publishing" ... %>-->
 <link href="ContosoPageLayout.css" type="text/css" ms-design-css-conversion="no" />
 <!--MS:<Publishing:EditModePanel runat="server" id="editmodestyles">-->
 <!--MS:<SharePoint:CssRegistration ...>-->
 <!--ME:</SharePoint:CssRegistration>-->
 <!--ME:</Publishing:EditModePanel>-->
<!--CE: End Edit Mode Panel Snippet-->
<!--ME:</asp:ContentPlaceHolder>-->
```

As illustrated by the code example, you should add your **link** elements after the **SPM** elements in **PlaceHolderAdditionalPageHead** comment section. The **PlaceHolderAdditionalPageHead** control adds content to the end of the **head** element in the rendered page. This means that your page layout CSS files appear after any other CSS files included in the master page, which in turn ensures that your styles override default SharePoint styles.

As an alternative, you can, of course, include styles for your page layout in the CSS file, or files, associated with the master page. However, this approach can become unwieldy if you have many page layouts, each with its own style requirements.



**Additional Reading:** For more information about creating page layouts, see *How to: Create a page layout in SharePoint 2013* at <http://go.microsoft.com/fwlink/?LinkID=311883>.

## Demonstration: Developing Page Layouts with Design Manager

The instructor will now demonstrate the process of creating a page layout by using Design Manager.

### Demonstration Steps

- Connect to the 20488B-LON-SP-15 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the Start screen, click **Internet Explorer**, and browse to **http://publishing.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the **Settings** menu, click **Design Manager**.
- On the list on the left side of the page, click **Edit Page Layouts**.
- On the **Design Manager: Edit Page Layouts** page, click **Create a page layout**.
- On the **Create a Page Layout** dialog box, in the **Name** box, type **FactSheet**.
- In the **Master Page** list, click **seattle**.
- In the **Content Type** list, click **Article Page**, and then click **OK**.
- Without closing Internet Explorer, open a File Explorer window and click **masterpage**.

- In the File Explorer window, point out that SharePoint has created files named **FactSheet.aspx** and **FactSheet.html**.
- Right-click **FactSheet.html**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- Briefly review the HTML content that SharePoint has generated for the page layout. Point out the commented out **asp:ContentPlaceHolder** elements, which represent the content placeholders in the master page. You add all markup for the page layout within these elements.
- Toward the end of the file, draw attention to how SharePoint has created a **div** element for each field control. Explain that you can edit the HTML and add CSS files to control how the field controls are presented.
- Close Visual Studio, File Explorer, and then Internet Explorer.

## Importing and Exporting Design Packages

SharePoint 2013 enables you to import and export design packages as .wsp files. This provides a convenient way to package and migrate site design assets between SharePoint deployments, such as from a development or staging environment to a production environment. It also creates opportunities for organizations to outsource site design, by commissioning a third party to create a design package.

- Export design assets as a .wsp file
- Contains all custom site design assets found within the site collection galleries:
  - Master pages
  - Page layouts
  - CSS and JavaScript files
  - Images and image renditions
  - Device channels and device channel mappings
- Import and apply to other site collections

### Exporting a Design Package

The high-level process for creating and exporting a design package is as follows:

1. Create site design assets, such as master pages, page layouts, theme components, and composed looks, on a SharePoint publishing site (for example, within a development environment).
2. Use the **Create Design Package** option in Design Manager, and specify a name for the design.
3. Download the .wsp file from the link provided in Design Manager and distribute it as required.

### Importing a Design Package

You can import and apply a design package in a single step. On the site collection to which you want to apply the design, use the **Import a complete design package** option in Design Manager to upload and import the .wsp file. When you import a design package, SharePoint will:

- Add all site assets within the package to the relevant site galleries.
- Apply the design settings, such as the current master page, that were applied in the source site.

## Lab A: Branding and Designing Publishing Sites

### Scenario

The management team at Contoso wants to develop an Internet-facing corporate website using SharePoint 2013. The team has asked you to lead the creation of a prototype publishing site. As a starting point in this process, you will work with web designers to develop and test a master page design for the new site.



**Note:** The HTML and CSS resources provided for this lab are simple examples to help illustrate the iterative site design process. They are not intended to represent best practices for SharePoint site design.

### Objectives

After completing this lab, you will be able to:

- Create master pages for SharePoint 2013.
- Add SharePoint components, such as the site logo, the site title, and navigation controls, to a master page.
- Publish a master page and apply it to a SharePoint site.

Estimated Time: 30 minutes

- Virtual Machine: 20488B-LON-SP-15
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

### Exercise 1: Creating SharePoint Master Pages

#### Scenario

In this exercise, you will create a new SharePoint master page. First, you will map the master page gallery as a network drive. This makes it easier to make regular changes to design assets and to preview your changes in Design Manager. Next, you will use Design Manager to create a new minimal master page. You will then use HTML and CSS to build a design around the minimal master page.

The main tasks for this exercise are as follows:

1. Map the Master Page Gallery as a Network Drive
2. Create a New Minimal Master Page
3. Add HTML to the Master Page
4. Add CSS to the Master Page
5. Preview Your Master Page Changes

#### ► Task 1: Map the Master Page Gallery as a Network Drive

- Start the 20488B-LON-SP-15 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In File Explorer, map the master page gallery at **http://publishing.contoso.com/\_catalogs/masterpage** as a network drive. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.



- Verify that you are able to browse the contents of the master page gallery in File Explorer, and then close the File Explorer window.

### ► Task 2: Create a New Minimal Master Page

- On the Start screen, click Internet Explorer, and browse to **http://publishing.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Browse to Design Manager, and then use Design Manager to create a minimal master page named **ContosoPrototype1**.
- Preview the new master page, and select the page at **http://publishing.contoso.com/pages/default.aspx** as the content page for previewing.



**Note:** The preview now shows how the Default.aspx page would look if it was rendered in your new master page. As you edit the master page, you can return to this preview page to preview your changes.

### ► Task 3: Add HTML to the Master Page

- Without closing Internet Explorer, open a File Explorer window and browse to the network drive that you connected to the master page gallery.
- Use Visual Studio 2012 to open the **ContosoPrototype1.html** file.
- Within the **body** element, immediately before the **div** element with an **id** value of **s4-workspace**, add the following code:

```
<header>
 <div class="contentContainer">
 </div>
</header>
```



**Note:** When you type a **class** attribute, Visual Studio may display dialog boxes warning that it is unable to edit certain built-in CSS files. You are not trying to edit the CSS files, so you can safely close these dialog boxes and continue.

- Locate the **div** element with an **id** value of **s4-bodyContainer**.
- Enclose the contents of the **s4-bodyContainer div** element in a **section** element.
- Add an **id** attribute to the **section** element, and set the value of the **id** attribute to **content**.
- Immediately before the closing **</body>** tag, add the following code:

```
<footer>
 <div class="contentContainer">
 </div>
</footer>
```

- Save the file.

#### ► Task 4: Add CSS to the Master Page

- Use File Explorer to copy the **ContosoLayout.css** file from the **E:\Labfiles\Starter** folder to the master page gallery.
- On the taskbar, click **File Explorer**.
- Browse to **E:\Labfiles\Starter**, right-click **ContosoLayout.css**, and then click **Copy**.
- Under **Computer**, click **masterpage**.
- Right-click any white space within the list of files, and then click **Paste**.
- Switch back to Visual Studio.
- Within the **head** element, locate the following line of code:

```
<!--ME:</SharePoint:CssLink>-->
```

- On a new line immediately after the line of code you just located, add a **link** element that loads the **ContosoLayout.css** file.



**Note:** SharePoint uses the **CssLink** control to load default CSS style sheets. By adding your CSS links after the **CssLink** control, you ensure that your styles can override the default styles. Similarly, by adding your CSS links before the **PlaceHolderAdditionalPageHead** control, you ensure that page layouts can insert CSS links after your master page links and thereby override your styles.

- Save the file.

#### ► Task 5: Preview Your Master Page Changes

- Switch back to Internet Explorer.
- On the preview page, reload the page.
- Verify that the preview page reflects the changes you have made to the **ContosoPrototype1** master page.
- Leave Internet Explorer open. You will continue to work with Design Manager in the next exercise.

**Results:** After completing this exercise, you should have created a new SharePoint master page, added design elements, and previewed your design in Design Manager.

## Exercise 2: Building Master Page Functionality

### Scenario

In this exercise, you will start to build the functionality of your master page by adding standard SharePoint controls. You will add the site logo, the site title, and the global navigation links. After each change, you will use Design Manager to preview the additions.

The main tasks for this exercise are as follows:

1. Add the Site Logo to the Master Page
2. Add the Site Title to the Master Page
3. Add Global Navigation to the Master Page

### ► Task 1: Add the Site Logo to the Master Page

- Use Design Manager to generate an HTML snippet for a **Site Logo** control.
- Copy the HTML snippet to the clipboard, and then switch back to Visual Studio.
- Within the **header** element, within the **div** element with a **class** attribute value of **contentContainer**, paste the HTML snippet from the clipboard.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

- Within the code you just added, locate the **div** element with a **data-name** attribute value of **logo**.
- Amend the **div** tag to include an **id** attribute value of **logo**, as follows:

```
<div id="logo" data-name="SiteLogo">
```



**Note:** The custom CSS file, ContosoLayout.css, uses the **logo** ID to position the site logo.

- Save your work, and then switch back to Internet Explorer.
- Reload the master page preview page.
- Verify that the preview page now includes the site logo, positioned on the left of the site header.
- In a new Internet Explorer tab, browse to **http://publishing.contoso.com**.
- In the site settings for the publishing.contoso.com site collection, set the site logo to the image at **E:\Labfiles\Starter\ContosoLogoHorizontal.png**.
- Switch back to the tab that shows a preview of your master page.
- Reload the master page preview page, and verify that the page now displays the updated site logo.

### ► Task 2: Add the Site Title to the Master Page

- Use Design Manager to generate an HTML snippet for a **Site Title** control.
- Copy the HTML snippet to the clipboard, and then switch back to Visual Studio.
- Within the **header** element, within the **div** element with a **class** attribute value of **contentContainer**, paste the HTML snippet from the clipboard.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

- Within the code you just added, locate the **div** element with a **data-name** attribute value of **SiteTitle**.
- Amend the **div** tag to include an **id** attribute value of **mainTitle**, as follows:

```
<div id="mainTitle" data-name="SiteTitle">
```



**Note:** The custom CSS file, ContosoLayout.css, uses the **mainTitle** ID to style and position the site title.

- Save your work, and then switch back to Internet Explorer.

- Reload the master page preview page.
- Verify that the preview page now includes the site title, displayed centrally in the header.

### ► Task 3: Add Global Navigation to the Master Page

- Use Design Manager to generate an HTML snippet for a **Top Navigation** component.
- Copy the HTML snippet to the clipboard, and then switch back to Visual Studio.
- Within the **header** element, within the **div** element with a **class** attribute value of **contentContainer**, paste the HTML snippet from the clipboard.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

- Notice that the code you just added includes a **div** element with a **class** value that includes **ms-core-listMenu-horizontalBox**. The ContosoLayouts.css file overrides this class to style and position the navigation controls.
- Save your work, and then switch back to Internet Explorer.
- Reload the master page preview page.
- Verify that the preview page now includes global navigation links, displayed in white at the lower-right of the header.

**Results:** After completing this exercise, you should have added a site logo, a site title, and global navigation links to a custom SharePoint master page.

## Exercise 3: Publishing and Applying Design Assets

### Scenario

In this exercise, you will publish the custom master page and CSS files. You will then apply the master page to the site.

The main tasks for this exercise are as follows:

1. Publish the Draft Assets
2. Apply the Master Page to the Site

### ► Task 1: Publish the Draft Assets

- In Internet Explorer, browse to the master page gallery of the **publishing.contoso.com** site collection.



**Note:** To check in and publish files, you must use the browser UI rather than the mapped network drive.

- Publish a major version of the **ContosoPrototype1.html** file.



**Note:** When you publish a master page .html file, SharePoint automatically publishes the associated .master file.

- Publish a major version of the **ContosoLayout.css** file.
- ▶ **Task 2: Apply the Master Page to the Site**
- In the site settings for the publishing.contoso.com site collection, set the site master page to **ContosoPrototype1**.
- Set the system master page to **ContosoPrototype1**.
- Browse to the site home page, and verify that the new master page renders correctly.
- Close all windows.

**Results:** After completing this exercise, you should have published and applied a new site master page and associated resources.

## Lesson 3

# Tailoring Content to Platforms and Devices

People use an increasingly diverse array of devices to browse websites and access web content. Designers of public-facing websites must consider how their designs will render on a range of screen sizes and across devices with a range of capabilities.

In many cases, you will want to tailor site design assets to particular devices or classes of device. For example, you would typically want to display less content on a mobile phone display than on a full-size computer monitor. On touch-screen devices, you may want to display links as larger touch targets that are easier for a user to click. You may also need to adapt content for devices that do not support well-known web plug-ins.

SharePoint 2013 introduces a new feature, device channels, which enable you to assign master pages and associated design assets to particular devices or classes of device.

## Lesson Objectives

After completing this lesson, you will be able to:

- Create and configure device channels to target individual devices or broader classes of device.
- Use device channel panels in page layouts to customize how page content is presented.
- Create, configure, and use image renditions.

## Understanding Device Channels

In SharePoint 2013, device channels are a technology that enables you to associate site design assets with different types of device. Device channels work by examining the *user-agent string* of incoming HTTP requests to identify the type of device that submitted the request.

### Understanding User-Agent Strings

When a user browses to a website, the HTTP GET request includes a header named **User-Agent**. This header, usually referred to as the user-agent string, contains a set of tokens that provide information about the client browser and device. For example, if you browse to a website from a Windows desktop with Internet Explorer 10, your user-agent string will resemble the following:

```
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
```

In contrast, if you browse to a website from a Nokia Lumia 900 mobile phone running Windows Phone 7.5, your user-agent string will resemble the following:

```
Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0; NOKIA; Lumia 900)
```

The user-agent string includes a *device substring* that identifies the platform or device that submitted the request. In the first example, the device substring is **Windows NT 6.1**, which indicates that the client device is a computer running Windows 7 or Windows Server 2008 R2. In the second example, the device

- Enable you to map site assets to classes of device
- Use device inclusion rules to match user-agent substrings, for example:
  - Windows Phone OS 7.5
  - Windows Phone OS
  - Android 4.2.2
  - Android
- SharePoint evaluates device channels in order
- Map master pages to each device channel

substring is **Windows Phone OS 7.5**, which indicates that the device is a mobile phone running Windows 7.5.



**Additional Reading:** For a more detailed analysis of user-agent strings, see *Understanding user-agent strings* at <http://go.microsoft.com/fwlink/?LinkID=311884>.

## Device Inclusion Rules and Device Channel Ordering

When you create a device channel, you specify one or more *device inclusion rules*. Each device inclusion rule is simply a user-agent token, or part of a user-agent token. For example:

- Specify **Windows Phone OS 7.5** to match Windows Phone 7.5 devices.
- Specify **Windows Phone OS** to match any Windows Phone device.
- Specify **Android 4.2.2** to match Android devices running version 4.2.2.
- Specify **Android** to match any Android device.

When a user requests a page, SharePoint reads the user-agent string for the request and attempts to match device inclusion rules. Device channels are evaluated in order, and you can change the order of your device channels in Design Manager. If the user-agent string matches a device inclusion rule for a device channel, SharePoint will apply that design channel. If the user-agent string does not match, SharePoint will continue working down the list of device channels until a match is found. The last item in the list of device channels is the default device channel, which matches all devices.

As a general rule, you should order your device channels so that those with the most specific device inclusion rules are evaluated first.

## Associating Design Assets With Device Channels

You can assign a master page to each device channel on your SharePoint publishing site. Each master page, in turn, can link to associated design assets, such as CSS and JavaScript files. This gives you complete control over how your site is rendered for each device channel.



**Additional Reading:** For more information about device channels, see *SharePoint 2013 Design Manager device channels* at <http://go.microsoft.com/fwlink/?LinkID=311885>.

## Demonstration: Creating and Configuring a Device Channel

The instructor will now demonstrate how to create and configure a device channel. You will also see how to assign master pages to device channels, and how to preview device channels by specifying a **DeviceChannel** query string.

### Demonstration Steps

- Connect to the 20488B-LON-SP-15 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Internet Explorer and browse to **http://publishing.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the **Settings** menu, click **Design Manager**.

- On the list on the left side of the page, click **Manage Device Channels**.
- On the **Design Manager: Manage Device Channels** page, click **Create a channel**.
- In the **Device Channels – New Item** dialog box, in the **Name** box, type **Windows Phone Devices**.
- In the **Alias** box, type **WindowsPhoneGeneric**.



**Note:** Explain that the alias is used to reference the device channel in various scenarios, such as when you create device channel panels.

- In the **Description** box, type **A device channel for all Windows Phone devices**.
- In the **Device Inclusion Rules** box, type **Windows Phone OS**.
- Select **Active**, and then click **Save**.
- On the list on the left side of the page, click **Publish and Apply Design**.
- On the **Design Manager: Publish and Apply Design** page, click **Assign master pages to your site based on device channel**.
- In the **Site Master Page Settings** dialog box, in the **Site Master Page** section, in the **Windows Phone Devices** drop-down menu, click **oslo**, and then click **OK**.
- On the top navigation menu, click **Contoso Publishing** to return to the site home page.
- Append **?DeviceChannel=WindowsPhoneGeneric** to the page URL, and then press Enter.



**Note:** Explain that the **DeviceChannel** query string provides a way of testing device channels. When you specify a **DeviceChannel** query string, the device channel is applied regardless of device inclusion rules.

- Point out that the page reloads using the alternative master page.

## Using Device Channel Panels

A device channel panel is a SharePoint control that renders its contents only to specified device channels. Device channel panels are defined by the **DeviceChannelPanel** class in the **Microsoft.SharePoint.Publishing.WebControls** namespace.

Device channel panels are typically used within page layouts to tailor page content to particular classes of devices. For example, suppose your page layout content type contains fields named **ShortDescription** and **LongDescription**. You might want to display the content of the **ShortDescription** field on mobile devices and the content of the **LongDescription** field on desktop devices. You could accomplish this in the page layout design by enclosing the field controls for **ShortDescription** and **LongDescription** in device channel panels and targeting each panel at relevant device channels.

- Display content only to specified device channels
- Typically used within page layouts
- Generate HTML snippet in Design Manager
- Set **IncludedChannels** property to a comma-separated list of device channel aliases
- Can include HTML elements and SharePoint components
- Cannot include Web Parts



The **DeviceChannelPanel** control includes a property named **IncludedChannels**. This is a comma-separated list of device channel aliases. The device channel panel will only render its contents if one of the device channels specified in this property is currently being applied to the request.

You can use Design Manager to generate an HTML snippet for a device channel panel, and then paste the HTML snippet into your page layout HTML file. You can include both HTML elements and SharePoint components in a device channel panel; however, you cannot use a device channel panel to selectively display Web Parts.

## Understanding Image Renditions

When you add images to websites, you may often use the same image at different resolutions in different contexts. For example, you might display an image at full resolution on one page, but as a thumbnail on another page. In the case of background images or site logos, you might display smaller versions of the same image on mobile or tablet devices.

Typically, site designers use CSS to control the size and resolution at which images are rendered. However, this is not an optimal approach, because the browser must download the full size image and then rescale it locally. Image renditions provide a more efficient alternative, by creating versions (renditions) of an image at various resolutions. When a page designer includes an image in a page, he or she can specify which rendition of the image to include. As a result, the browser downloads an appropriately-sized version of the image, rather than downloading a large image and resizing it.

- Create renditions of images at specific resolutions:
  - Optimizes page load times
  - Reduces bandwidth consumption
  - Requires BLOB cache
- To create a rendition, specify
  - Name
  - Width
  - Height
- Renditions generated automatically for every image on the site
- Customize how renditions are generated for individual images

### Image Rendition Prerequisites

Image renditions are only available when:

- You are using a publishing site collection.
- The binary large object (BLOB) cache is enabled on each front-end web server.

In a production environment, BLOB cache settings are typically configured by farm administrators or server administrators. However, in a development environment, you may need to configure BLOB cache settings yourself by editing the **BlobCache** element in the **Web.config** file.



**Reference Links:** For guidance on how to enable and configure the BLOB cache, see *Configure cache settings for a web application in SharePoint Server 2013* at <http://go.microsoft.com/fwlink/?LinkID=311886>.

### Defining Image Renditions

You can define image renditions in the site settings for any SharePoint 2013 publishing site. To define an image rendition, you must specify the following properties:

- *Name*. This is a friendly name that is used to identify the rendition in previews.
- *Width*. The width of the image rendition in pixels.
- *Height*. The height of the image rendition in pixels.

An ID field value is also assigned automatically when you create a new rendition.

After you define an image rendition, SharePoint will automatically create renditions for all images in asset libraries on the site. Image renditions are often created in conjunction with page layouts and display templates. For example, suppose you design a page layout that includes an image field. To display pages as you intended, you might require that the image field displays the image with a specified resolution and aspect ratio. By creating an image rendition with the required width and height, you make it easy for page authors to add images in a suitable format.

To display an image using a particular rendition, you append a **RenditionID** query string value to the URL of the image. The **RenditionID** query string identifies the ID field value of the rendition you want to use. For example, if your image file is stored at the following URL:

```
http://publishing.contoso.com/SiteAssets/Logo.png
```

To display the image rendition with an ID value of 1, you would use the following URL:

```
http://publishing.contoso.com/SiteAssets/Logo.png?RenditionID=1
```

### Refining Renditions For Individual Images

By default, SharePoint uses the following process to generate renditions for an image:

- It resamples the entire image at the rendition resolution.
- If the aspect ratio of the image does not match the aspect ratio of the rendition, the image will be cropped as part of the resampling process.

You can customize the way renditions are generated for individual images. For example, suppose you have an image measuring 2,000 x 2,000 pixels, and a rendition measuring 500 x 500 pixels. By default, SharePoint will generate the rendition by scaling the entire image down to 500 x 500 pixels. However, you could instead choose to show the upper-left quarter of the image at full resolution, or crop the image before scaling. In each case, the rendition will display the image with a resolution of 500 x 500 pixels, but you get to choose what part of the image is represented by those 500 x 500 pixels. The next topic demonstrates how to do this.

## Demonstration: Creating and Configuring an Image Rendition

The instructor will now demonstrate how to work with image renditions. First, you will see how to enable the BLOB cache in a development environment by editing the Web.config file. Next, you will see how to define image renditions by specifying a name, a width, and a height. Finally, you will see how to customize how individual images are presented in each rendition.



**Note:** In production environments, you should avoid manually editing Web.config files, because it becomes hard to keep changes synchronized across multiple web servers. Instead, you should use the **SPWebConfigModification** class to make the changes programmatically, typically within a feature receiver class. For more information, see *How to: Add and Remove Web.config Settings Programmatically* at <http://go.microsoft.com/fwlink/?LinkID=311887>. The article was written for SharePoint 2010, but the process remains unchanged in SharePoint 2013.

### Demonstration Steps

- Connect to the 20488B-LON-SP-15 virtual machine.
- If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.

- Open a File Explorer window and browse to **C:\inetpub\wwwroot\wss\VirtualDirectories\80**.
- Right-click **web.config**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- In the **web.config** file, locate the **BlobCache** element.
- Review the **BlobCache** element attributes:
  - The **location** attribute specifies the file system location where files will be cached.
  - The **path** attribute specifies the file types that should be cached, in the form of a regular expression.
  - The **maxSize** attribute specifies the maximum size of the cache in gigabytes (GB).
  - The **enabled** attribute specifies whether the BLOB cache is enabled.



**Note:** The BLOB cache is already enabled in the virtual machine to save time during the demonstration.

- Close Visual Studio.
- Open Internet Explorer and browse to **http://publishing.contoso.com**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- On the **Settings** menu, click **Site settings**.
- On the **Site Settings** page, under **Look and Feel**, click **Image Renditions**.
- On the **Image Renditions** page, click **Add new item**.
- On the **New Image Rendition** page, in the **Name** box, type **Landscape Large**.
- In the **Width** box, type **500**.
- In the **Height** box, type **400**, and then click **Save**.
- On the left side of the page, click **Site Contents**.
- On the **Site Contents** page, click **Site Collection Images**.
- On the **Site Collection Images** page, click **new item**.
- In the **Add a document** dialog box, browse to **E:\Democode\VanGogh-starry\_night.jpg**, click **Open**, and then click **OK**.
- In the **Site Collection Images – VanGogh-starry\_night.jpg** dialog box click **Save**.
- On the **Site Collection Images** page, click **All Assets**.
- On the **All Assets** page, select the **VanGogh-starry\_night** row (click in the column with a checkmark header).
- On the ribbon, on the **DESIGN** tab, click **Edit Renditions**.



**Note:** Because you are viewing a newly-uploaded image, SharePoint may take approximately a minute to generate the renditions.

- Scroll down the page and briefly review how the image appears in each rendition.
- Under the **Landscape Large** rendition, click **Click to change**.

- Click **OK** in the **Message from Webpage** dialog box.
- Drag the image handles to resize the selected area. Notice how the resolution stays the same, at 500 x 400 pixels, but you are effectively zooming in on the selected area.
- Drag the selected area around, and notice how the preview image is updated.



**Note:** The key point here is that the dimensions of the rendition do not change; in this case, they are fixed at 500 x 400 pixels. However, for individual images, you can configure whether to scale the whole image down to the size of the rendition, or whether to crop the image to size, or a combination of the two.

- Click **Cancel**, and then close Internet Explorer.

## Lesson 4

# Configuring and Customizing Navigation

SharePoint 2013 includes a sophisticated navigation infrastructure. SharePoint navigation providers can efficiently and dynamically create a navigation structure that reflects the site collections, sites, libraries, and lists in your deployment. When items are added or removed, the navigation providers automatically reflect the changes. SharePoint navigation providers also dynamically trim navigation links to remove any nodes that the current user does not have permission to visit.

For team sites and other intranet-facing sites, the out-of-the-box navigation experience meets the needs of most users. However, for publishing sites, you will probably want to customize the navigation experience to some extent. This customization could range from simply styling the built-in navigation controls, to creating your own custom navigation providers to completely overhaul the navigation system.

### Lesson Objectives

After completing this lesson, you will be able to:

- Describe the core navigation features in SharePoint Server 2013.
- Explain the SharePoint 2013 navigation architecture.
- Describe the main SharePoint navigation providers and their capabilities.
- Explain how managed navigation works in SharePoint 2013.
- Use server-side and client-side code to configure site navigation.
- Identify suitable approaches for customizing the navigation experience.

### Navigation in SharePoint Server 2013

SharePoint Server 2013 provides a wide variety of features and capabilities that you can use to implement navigation, and users can often navigate around a SharePoint site in a variety of ways. SharePoint sites typically include the following types of navigation functionality:

- *Global navigation.* Global navigation is implemented in the site master page, and provides a way for users to browse between the sites in a site collection. Global navigation is typically implemented as a "top link bar" across the top of the page.
- *Current navigation.* Current navigation provides a way for users to browse the contents of the current SharePoint site. Current navigation is typically implemented in the master page as a Quick Launch navigation menu, a tree view control, or both. In built-in master pages, the current navigation is displayed in a sidebar on the left of the page.
- *Breadcrumb trails.* Breadcrumb trails display a dynamically-generated set of links that represent the path through the site hierarchy from the root node to the current location. Breadcrumb trails are used widely within page layouts and other SharePoint pages.
- *Metadata navigation.* Metadata navigation enables users to navigate through specific lists or libraries by applying filters based on metadata values. You can configure metadata navigation through the

- Navigation in SharePoint
  - Global navigation
  - Current navigation
  - Breadcrumb trails
  - Metadata navigation
- Security trimming
- Types of navigation:
  - Structural
  - Managed

Metadata Navigation Settings page on each list. By default, the metadata navigation controls are displayed in the sidebar on the left side of the page.

In addition to these core types of navigation, users can often configure enhanced or customized navigation by adding navigation-based Web Parts to pages.

### Security Trimming

Navigation links in SharePoint are security trimmed. This means that SharePoint will only display links to items that the current user has permission to view. Security trimming results in a cleaner user interface and an improved user experience, because each user will only see the navigation items that are relevant to them and will not be presented with navigation links that display authorization errors when clicked.

### Structural and Managed Navigation

On publishing sites, you can configure global and current navigation to work in one of two ways:

- *Structural navigation.* This is the traditional approach to navigation. Navigation nodes are generated from the sites, subsites, and pages in the site collection.
- *Managed navigation.* This is a newer and more flexible approach to navigation. Navigation nodes are generated from a managed metadata term set. The term set hierarchy is used to build a friendly URL, which is mapped to a SharePoint page. You can use this approach to display static content or to drive dynamic search-driven content.

## Understanding the SharePoint Navigation Architecture

The navigation architecture in SharePoint 2013 is based on the ASP.NET navigation model. However, SharePoint extends many of the ASP.NET navigation classes to provide additional functionality, such as security trimming and audience targeting. Navigation in SharePoint 2013 consists of up to three logical layers: site map providers (also known as navigation providers), navigation controls, and data source controls.

- Site map providers:
  - Also known as navigation providers
  - Provide a collection of site map nodes
  - Perform security trimming
- Navigation controls:
  - Render site map nodes in various ways
  - AspMenu
  - SPTreeView
  - ListSiteMapPath
- SiteMapDataSource controls:
  - Act as an intermediary between providers and controls
  - Configure starting node, maximum node depth, and more

### Site Map Providers

Almost all websites have a hierarchical navigation structure. Any location within the website may have a parent, and it may have one or more children. The ASP.NET navigation model uses the **SiteMap** class to represent this hierarchical navigation structure. Individual pages within the website are represented by **SiteMapNode** objects, which can have ancestors (parent nodes) and descendants (child nodes). These collections of site map nodes are made available to navigation controls by classes that implement the abstract **SiteMapProvider** class.

SharePoint 2013 includes many site map provider classes, almost all of which derive from the abstract ASP.NET **SiteMapProvider** class. The role of every site map provider is the same: to expose a collection of site map nodes to consumer controls. Different site map provider classes generate site map nodes from different types of data source, such as content databases, XML files, or managed metadata term sets. The next topic describes the most common SharePoint site map providers in more detail.

One of the most important roles performed by SharePoint site map providers is security trimming. In this process, any nodes that a user does not have permission to visit are removed from the collection of site map nodes. As a result, users are only presented with navigation links that are relevant to them and that they have permission to visit.

Site map providers are registered and configured in the Web.config file for each web application (or web application zone).

## Navigation Controls

Users navigate through a SharePoint site by using a variety of navigation controls. A navigation control is simply a control that, directly or indirectly, renders nodes from a site map provider. Master pages almost always include one or more navigation controls, typically for the global navigation at the top of the page and the current navigation on the left side of the page. Other navigation controls, such as breadcrumb trails, may be configured within page layouts, application pages, or Web Part pages.

You can use any suitable data-bound control to display navigation items. The controls most commonly used in SharePoint sites are as follows:

- *AspMenu*. This is a highly-configurable ASP.NET control that enables you to display hierarchical data in a variety of ways. In the built-in master pages, the **AspMenu** control provides the top navigation bar and the Quick Launch navigation menu.
- *SPTreeView*. This is a SharePoint control that displays data, typically, the contents of an individual site, in a hierarchical tree structure.
- *SiteMapPath*. This is an ASP.NET control that displays the path from the root node to the current node as a breadcrumb trail. The **SiteMapPath** control is used extensively in page layouts and built-in SharePoint pages.
- *ListSiteMapPath*. This is a SharePoint control that extends the **SiteMapPath** control to support multiple site map providers. This enables you to consolidate global and current navigation into a single breadcrumb trail. Built-in SharePoint master pages use the **ListSiteMapPath** control to provide a pop-up breadcrumb trail from any page in the site collection.

Although it is not strictly a navigation control, you should also be aware of the **SPNavigationManager** control. In out-of-the-box SharePoint sites, the left side of the page includes a Quick Launch navigation menu and a tree view control that you can show or hide through the site settings. The **SPNavigationManager** control provides a wrapper for these controls, and it is the **SPNavigationManager** that enables you to control the visibility of the child controls.

## Data Source Controls

Some navigation controls, such as **SiteMapPath** and **ListSiteMapPath**, consume data directly from site map providers. Others, most notably the **AspMenu**, require an intermediary data source control.

When you use an **AspMenu**, you data-bind the **AspMenu** control to a **SiteMapDataSource** control. The **SiteMapDataSource** control, in turn, connects to a site map provider. You use the **SiteMapDataSource** to specify which site map nodes you want to retrieve from the site map provider and make available to the navigation control. For example, you can use the **SiteMapDataSource** control to specify which node represents the starting point for your navigation.

The following code example shows how **AspMenu** controls are used in conjunction with **SiteMapDataSource** controls:

### Using SiteMapDataSource Controls with AspMenu Controls

```
<asp:SiteMapDataSource
 ShowStartingNode="False"
 SiteMapProvider="SPNavigationProvider"
 id="topSiteMap"
 runat="server"
 StartingNodeUrl="sid:1002"/>

<SharePoint:AspMenu
 ID="TopNavigationMenu"
 runat="server"
 EnableViewState="false"
 DataSourceID="topSiteMap"
 UseSimpleRendering="true"
 UseSeparateCss="false"
 Orientation="Horizontal"
 StaticDisplayLevels="2"
 AdjustForShowStartingNode="true"
 MaximumDynamicDisplayLevels="2"
 SkipLinkText="" />
```

In this example, which was taken from the built-in seattle.master master page, the **AspMenu** control specifies the **SiteMapDataSource** control as its data source. In turn, the **SiteMapDataSource** identifies the site map provider named **SPNavigationProvider** (as defined in the web.config file) as its data provider.



**Note:** The **SPTreeView** control is typically data-bound to a **SPHierarchyDataSourceControl** control. The **SPHierarchyDataSourceControl** does not connect to a site map provider. Instead, it generates a hierarchical set of child nodes dynamically from a specified starting point, such as a specific site or list.

## SharePoint 2013 Site Map Providers

SharePoint 2013 provides many built-in site map providers. Before you consider creating a custom site map provider, you should evaluate whether any of the built-in providers meet your requirements. The following table lists providers SharePoint 2013 uses extensively.

- SPSiteMapProvider
- SPContentMapProvider
- SPXmlContentMapProvider
- SPNavigationProvider
- PortalSiteMapProvider
- TaxonomySiteMapProvider



Provider	Details
SPSiteMapProvider	<ul style="list-style-type: none"> <li>Provides site map nodes for the global part of the breadcrumb trail (in other words, the path from the root node to the current <b>SPWeb</b> instance).</li> <li>Typically used as a site map provider for breadcrumb trails.</li> </ul>
SPContentMapProvider	<ul style="list-style-type: none"> <li>Provides site map nodes for the local part of the breadcrumb trail (in other words, the path from the current <b>SPWeb</b> instance to the current item).</li> <li>Typically used as a site map provider for breadcrumb trails.</li> </ul>
SPXmlContentMapProvider	<ul style="list-style-type: none"> <li>Provides site map nodes based on an XML content source.</li> <li>Used by SharePoint to provide breadcrumb navigation between application pages in the <b>_layouts</b> folder.</li> <li>Can be used to provide navigation across site collection boundaries.</li> </ul>
SPNavigationProvider	<ul style="list-style-type: none"> <li>Provides global and local site map nodes.</li> <li>Used by SharePoint to populate the top navigation controls and the Quick Launch navigation menu in non-publishing sites.</li> <li>Typically used in non-publishing sites.</li> <li>Can be used as a base class for custom navigation providers.</li> </ul>
PortalSiteMapProvider	<ul style="list-style-type: none"> <li>Provides global, local, or combined global and local site map nodes.</li> <li>Used by SharePoint to populate the top navigation controls and the Quick Launch navigation menu in publishing sites.</li> <li>Uses cached objects, rather than actual <b>SPWeb</b> objects, to generate navigation nodes</li> <li>Effective site map provider for a wide range of applications.</li> <li>Not available in SharePoint Foundation.</li> </ul>
TaxonomySiteMapProvider	<ul style="list-style-type: none"> <li>Provides site map nodes generated from a managed metadata term set.</li> <li>Used by SharePoint when you configure a site to use managed navigation (rather than structural navigation).</li> </ul>

Although these are the most commonly used site map providers, SharePoint includes many more obscure site map providers for specific scenarios. The Web.config file for an out-of-the-box SharePoint deployment registers site map providers based on a range of classes, such as **AdministrationQuickLaunchProvider**, **SharedServicesQuickLaunchProvider**, **SiteDirectoryCategoryProvider**, **MySitePersonalQuickLaunchProvider**, **SwitchableSiteMapProvider**, **MySiteMapProvider**, **MySiteLeftNavProvider**, **MySiteSitesPageStaticProvider**, **EduTopNavProvider**, and **EduQuickLaunchProvider**. However, the six providers described in the previous table will meet your requirements for most SharePoint customization scenarios.

## Understanding Managed Navigation

When you configure a publishing site to use managed navigation, rather than structural navigation, you are configuring the site to use a **TaxonomySiteMapProvider** instead of a **PortalSiteMapProvider**. The **TaxonomySiteMapProvider** instance uses a managed metadata term set to generate site map nodes. For manageability, you should define this navigation term set within a site collection-specific term set group. By default, the site collection administrator is the owner of the navigation term set.

- Relies on `TaxonomySiteMapProvider`
- Uses a navigation term set to generate site map nodes
- Friendly URLs:
  - URL based on navigation term
  - Each term is associated with a physical URL
  - Decouples term set structure from physical site structure
- Search-driven content:
  - Associate multiple terms with the same physical page
  - Use Content Search Web Part to display content from search index based on navigation terms

### Friendly URLs

Each term in a navigation term set is associated with a SharePoint page URL. However, the physical page URL is hidden from the end user. Instead, the site displays a friendly URL that is based on the associated managed metadata term. For example, suppose you define the following term set structure for the current navigation in your site:

- Division
  - North
  - South
  - East
  - West
- Department
  - Research
  - Legal
  - HR
  - IT

End users would browse the site using these terms. However, SharePoint would render the actual URLs associated with each term. This enables you to decouple the navigation structure provided by the term set from the physical structure of the site. The following table illustrates how friendly URLs might correspond to structural URLs.

Friendly URL	Physical URL
<a href="http://publishing.contoso.com/Division">http://publishing.contoso.com/Division</a>	<a href="http://publishing.contoso.com/Pages/Division.aspx">http://publishing.contoso.com/Pages/Division.aspx</a>
<a href="http://publishing.contoso.com/Division/North">http://publishing.contoso.com/Division/North</a>	<a href="http://publishing.contoso.com/Pages/North.aspx">http://publishing.contoso.com/Pages/North.aspx</a>
<a href="http://publishing.contoso.com/Department">http://publishing.contoso.com/Department</a>	<a href="http://publishing.contoso.com/Pages/Department.aspx">http://publishing.contoso.com/Pages/Department.aspx</a>
<a href="http://publishing.contoso.com/Department/IT">http://publishing.contoso.com/Department/IT</a>	<a href="http://publishing.contoso.com/Pages/IT.aspx">http://publishing.contoso.com/Pages/IT.aspx</a>

When you set up managed navigation for a publishing site, you can configure the site to create friendly URLs whenever a user adds a page to the site.

## Search-Driven Content

You can map multiple terms in a term set to the same physical page. When combined with the new Content Search Web Part (CSWP), this enables you to generate content dynamically from the search index when a user browses your site.

The CSWP is a highly-configurable Web Part that displays content from the search index based on a search query. When you configure a CSWP, you can specify how the search query that is used to retrieve content is generated. Notably, you can configure the CSWP to build a search query based on the current navigation terms.

For example, suppose you add the following mappings to your navigation term set:

- Map every top-level term to the Category.aspx page.
- Map every second-level term to the Topic.aspx page.

Now suppose that you add a CSWP to the Topic.aspx page, and configure the CSWP to build a search query from the current navigation terms. When a user browses the site, the following process occurs:

1. The user browses to <http://publishing.contoso.com/Division/North>.
2. The SharePoint site renders the Topic.aspx page.
3. The CSWP on the Topic.aspx page uses the navigation terms **Division** and **North** to construct a search query, for example to retrieve news content about the North division of Contoso.
4. The user sees news and information relating to the North division.

You can extend this process to support more sophisticated scenarios, such as Product Catalog functionality. In this scenario, a publishing site uses search-driven navigation to enable users to browse indexed catalog content from another site collection.



**Note:** Product Catalog functionality and configuring managed metadata term sets programmatically are covered in course 20489: *Developing Microsoft SharePoint Server 2013 Advanced Solutions*.

## Configuring Navigation Programmatically

The SharePoint 2013 object model includes various classes that you can use to configure and manage navigation settings for SharePoint sites. This is useful if you need to script the deployment of a site, or to ensure that navigation settings are applied in a repeatable way to multiple sites.

The most important navigation classes are as follows:

- *The PublishingWeb class.* This class provides a wrapper for **SPWeb** instances that have the publishing feature enabled. The **PublishingWeb** class provides programmatic access to various aspects of the publishing site, such as navigation settings, site variations, and page layouts.
- *The PortalNavigation class.* This class enables you to adjust how navigation nodes are rendered on a specific web. For example, you can use the **PortalNavigation** map to specify the subsites or pages

- Configuring structural navigation:
  - **PublishingWeb** class
  - **PortalNavigation** class
  - Server-side object model only
- Configuring managed navigation:
  - **WebNavigationSettings** class
  - **TaxonomyNavigation** class
  - Server-side or client-side

that should be included in the global or current navigation. You access a **PortalNavigation** object through the **PublishingWeb.Navigation** property.

- *The TaxonomyNavigation class.* This static class enables you to perform various tasks relating to managed navigation, such as retrieving and editing the navigation term set for a specific site.
- *The WebNavigationSettings class.* This class enables you to configure the global and current navigation for a specific site. Most notably, the **WebNavigationSettings** class enables you to switch between structural navigation and managed navigation.

Generally speaking, you use the **PublishingWeb** class and the **PortalNavigation** class if you want to configure structural navigation programmatically, and you use the **WebNavigationSettings** class and the **TaxonomyNavigation** class if you want to configure managed navigation programmatically. You can use **WebNavigationSettings** and **TaxonomyNavigation** from both server-side and client-side code, whereas the **PortalNavigation** class is only available in the server-side object model.

The following code example shows how to configure a site to use managed navigation in server-side code:

### Configuring a Site to Use Managed Navigation

```
var site = SPContext.Current.Site;
var web = SPContext.Current.Web;

// Get the managed metadata term store.
TaxonomySession session = new TaxonomySession(SPContext.Current.Site);
TermStore store = session.TermStores["Contoso Managed Metadata"];

// Create a new term set to use for navigation.
Guid navTermSetGuid = Guid.NewGuid();
Group group = store.GetSiteCollectionGroup(site, true);
TermSet termSet = group.CreateTermSet("Contoso Publishing Navigation", navTermSetGuid);
NavigationTermSet navTermSet = NavigationTermSet.GetAsResolvedByWeb(termSet, web,
 StandardNavigationProviderNames.GlobalNavigationTaxonomyProvider);
navTermSet.IsNavigationTermSet = true;
navTermSet.TargetUrlForChildTerms.Value = "~site/Pages/Category.aspx";
NavigationTerm termDivision = navTermSet.CreateTerm("Division", NavigationLinkType.FriendlyUrl);
termDivision.TargetUrlForChildTerms.Value = "~site/Pages/Topic.aspx";
NavigationTerm termDivisionNorth = termDivision.CreateTerm("North",
 NavigationLinkType.FriendlyUrl);
NavigationTerm termDivisionSouth = termDivision.CreateTerm("South",
 NavigationLinkType.FriendlyUrl);
NavigationTerm termDivisionEast = termDivision.CreateTerm("East",
 NavigationLinkType.FriendlyUrl);
NavigationTerm termDivisionWest = termDivision.CreateTerm("West",
 NavigationLinkType.FriendlyUrl);
store.CommitAll();

// Get the navigation settings for the current site.
WebNavigationSettings settings = new WebNavigationSettings(web);

// Clear any existing settings.
settings.ResetToDefaults();

// Configure the site to use managed navigation for the global navigation,
// and specify the navigation term set.
settings.GlobalNavigation.Source = StandardNavigationSource.TaxonomyProvider;
settings.GlobalNavigation.TermStoreId = store.Id;
settings.GlobalNavigation.TermSetId = termSet.Id;

// Configure the site to use managed navigation for the current navigation,
// and specify the navigation term set.
settings.CurrentNavigation.Source = StandardNavigationSource.TaxonomyProvider;
settings.CurrentNavigation.TermStoreId = store.Id;
settings.CurrentNavigation.TermSetId = termSet.Id;
```

```
// Update the navigation settings, and clear the site from the taxonomy navigation cache.
settings.Update(session);
TaxonomyNavigation.FlushSiteFromCache(site);
```

## Customizing the Navigation Experience

You can customize the SharePoint 2013 navigation experience in many different ways. How you approach the customization depends on what you hope to achieve. For example:

- *You want to change the way site map nodes are presented.* In this scenario, you should customize or replace the navigation control. There is no need to edit the site map provider or the intermediary data source.
- *You want to customize which nodes are displayed.* There are various ways in which you can achieve this. You can configure the intermediary **SiteMapDataSource** control to filter the nodes it retrieves from the site map provider, or to change the starting point for the current navigation. If you are using an **AspMenu** control, you can configure the static and dynamic node depths that the control displays. You can also programmatically edit the collection of global and current navigation nodes for the site.
- *You want to display site map nodes from an alternative data source.* In this scenario, you may need to implement a custom site map provider. Depending on your requirements, you can use one of the ASP.NET or SharePoint site map provider classes or create your own derived class.

Scenario	Possible Approaches
You want to change the way navigation nodes are presented	<ul style="list-style-type: none"> <li>• Configure the navigation control</li> <li>• Use a different navigation control</li> </ul>
You want to customize which nodes are displayed	<ul style="list-style-type: none"> <li>• Configure the SiteMapDataSource control</li> <li>• Configure the navigation control</li> <li>• Edit the node collections programmatically</li> </ul>
You want to display site map nodes from an alternative data source	<ul style="list-style-type: none"> <li>• Leverage an alternative site map provider</li> <li>• Create a custom site map provider</li> </ul>

## Customizing Navigation Controls

If you just want to change the look and feel of your site navigation, rather than changing the actual navigation structure of the site, you should customize or replace the navigation controls. The **AspMenu** control, which is used by default to provide the top link bar and the Quick Launch menu, is highly customizable. You can achieve a broad variety of navigation styles by changing the control properties and apply CSS to the control output. If the **AspMenu** control does not meet your requirements, you can use any hierarchical data-bound control to drive navigation on your site.

## Creating Custom Navigation Providers

If you want to generate site map nodes from an alternative data source, or you want a high level of control over how site map nodes are generated, you may need to create a custom navigation provider. Wherever possible, you should use an existing ASP.NET or SharePoint site map provider as the basis for your provider. For example:

- If you want to provide a fairly static collection of navigation links that are not constrained by site collection boundaries, consider using the **SpXmlContentMapProvider** class. This approach requires you to deploy your navigation links to the file system as an XML-based .sitemap file.
- If you want to create a navigation provider that retrieves navigation links from an entirely different source, such as a line-of-business database, consider creating a site map provider class that derives from the abstract ASP.NET **SiteMapProvider** class. You will need to implement your own security trimming functionality if required.

- If you want to leverage the functionality (such as security trimming) of SharePoint site map providers, but you need to customize how nodes are generated, consider creating a site map provider class that derives from the SharePoint **SPNavigationProvider** class.

## Lab B: Configuring Farm-Wide Navigation

### Scenario

Contoso consists of four geographical divisions. The marketing team at Contoso wants to create a publishing site collection for each division. The team also wants users to be able to navigate quickly between these divisional site collections, and your task is to implement this navigation. Because the default SharePoint navigation providers do not support navigation across site collection boundaries, you must create a custom navigation provider. You will then add custom navigation controls that display these divisional links to the footer of your prototype master page.

### Objectives

After completing this lab, you will be able to:

- Create a custom site map provider.
- Add custom navigation controls to a SharePoint master page.

Estimated Time: 15 minutes

- Virtual Machine: 20488B-LON-SP-15
- User name: CONTOSO\Administrator
- Password: Pa\$\$w0rd

### Exercise 1: Creating a Custom Site Map Provider

#### Scenario

In this exercise, you will create a SharePoint solution that deploys a custom XML-based site map file to the **\_layouts** virtual directory. You will also use a feature receiver class to register a new instance of the **SPXmlSiteMapProvider** class in the Web.config file for the publishing site. This site map provider will consume navigation data from your custom site map file.

The main tasks for this exercise are as follows:

1. Create an Empty SharePoint 2013 Project
2. Create an XML Site Map
3. Add a Site Map Provider to the Web.config File
4. Deploy and Test the Solution

#### ► Task 1: Create an Empty SharePoint 2013 Project

- Start the 20488B-LON-SP-15 virtual machine.
- Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open Visual Studio 2012, and create a new project with the following properties:
  - Use the **SharePoint 2013 – Empty Project** template.
  - Set the Visual Studio solution name to **CrossSiteNavigation**.
  - Create the solution in the **E:\Labfiles\Starter** folder.
  - Use the site at **http://publishing.contoso.com** for debugging.
  - Specify the farm solution deployment option.

### ► Task 2: Create an XML Site Map

- Add a **Layouts** mapped folder to the project.
- Within the **Layouts** mapped folder, in the **CrossSiteNavigation** folder, add a new XML file named **Contoso.sitemap**.
- In the **Contoso.sitemap** file, provide a custom navigation structure as follows:

```
<siteMap>
 <siteMapNode title="Root" url="http://publishing.contoso.com">
 <siteMapNode title="North" url="http://north.contoso.com" />
 <siteMapNode title="South" url="http://south.contoso.com" />
 <siteMapNode title="East" url="http://east.contoso.com" />
 <siteMapNode title="West" url="http://west.contoso.com" />
 </siteMapNode>
</siteMap>
```

- Save your work and then close the **Contoso.sitemap** file.

### ► Task 3: Add a Site Map Provider to the Web.config File

- Add a new Feature to the project, and rename the Feature node in Solution Explorer to **CrossSiteNavigation**.
- Set the title of the Feature to **Cross-Site Navigation**.
- Set the description of the Feature to **Configures a site map provider for cross-site navigation**.
- Set the scope of the Feature to **WebApplication**.
- Add an event receiver class to the Feature.
- In the class file, add a **using** statement that imports the **Microsoft.SharePoint.Administration** namespace.
- In the **FeatureActivated** method, use the **SPWebConfigModification** element to add the following element to the providers collection in the Web.config file:

```
<add name='ContosoCrossSiteProvider'
 siteMapFile='_layouts/15/CrossSiteNavigation/Contoso.sitemap'
 type='Microsoft.SharePoint.Navigation.SPXmlContentMapProvider, Microsoft.SharePoint,
Version=15.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c' />
```



**Note:** This element creates a new instance of the **SPXmlContentMapProvider** class, and configures the instance to retrieve navigation data from your custom site map file.



**Note:** The XML path to the **providers** element in the Web.config file is **configuration/system.web/siteMap/providers**.

- In the **FeatureDeactivating** method, remove your modification from the Web.config file.
- Build the solution and resolve any errors.



#### ► Task 4: Deploy and Test the Solution

- On the **DEBUG** menu, click **Start Without Debugging**.



**Note:** If you need to troubleshoot your feature receiver code, attach the Visual Studio debugger to the w3wp.exe process with the user name **CONTOSO\SPFarm**. If multiple w3wp.exe processes are running under **CONTOSO\SPFarm**, attach the debugger to all of them.

- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Open a File Explorer window and browse to the C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS folder.
- Verify that the **LAYOUTS** folder includes a subfolder named **CrossSiteNavigation**.
- Verify that the **CrossSiteNavigation** folder includes a file named **Contoso.sitemap**.
- Close the File Explorer window.
- In Visual Studio, on the **FILE** menu, point to **Open**, and then click **File**.
- In the Open File dialog box, browse to the following directory:
  - C:\inetpub\wwwroot\wss\VirtualDirectories\80
- Click **web.config**, and then click **Open**.
- In the Web.config file, in the configuration element, in the **system.web** element, in the **siteMap** element, in the **providers** element, verify that there is an **add** element with a name attribute value of **ContosoCrossSiteProvider**.
- Close Visual Studio.

**Results:** After completing this lab, you should have deployed a custom site map file and registered a custom site map provider.

## Exercise 2: Adding Custom Navigation Controls to a Master Page

### Scenario

In this exercise, you will modify the prototype master page that you created in the previous lab. Within the master page footer, you will configure a **SiteMapDataSource** instance to consume data from your custom site map provider. You will then configure an **AspMenu** control to display navigation links from the data source.

To preserve the relationship between the master page .html file and the .master file, you will use Design Manager to convert your ASP.NET markup into HTML snippets that you can add to the master page.

The main tasks for this exercise are as follows:

1. Add a Data Source to the Master Page
2. Add a Custom Navigation Control to the Master Page

### ► Task 1: Add a Data Source to the Master Page

- Open Internet Explorer and browse to <http://publishing.contoso.com>.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- Use Design Manager to generate an HTML snippet for the following ASP.NET markup:

```
<asp:SiteMapDataSource
 ID="bottomSiteMap"
 ShowStartingNode="false"
 SiteMapProvider="ContosoCrossSiteProvider"
 runat="server" />
```



**Note:** This ASP.NET element creates a **SiteMapDataSource** instance that retrieves navigation data from the site map provider you registered in the previous exercise.

- Use Visual Studio 2012 to open the **ContosoPrototype1.html** file from the network drive you mapped to the master page gallery in the previous lab.
- In the **ContosoPrototype1.html** file, within the **footer** element, within the **div** element, add the HTML snippet.
- Save the file and switch back to Internet Explorer.
- Refresh the master page preview page, and verify that the preview displays without errors.



**Note:** There are no new visible page components at this stage. However, the page preview will display an error if your ASP.NET markup contains errors.

### ► Task 2: Add a Custom Navigation Control to the Master Page

- In Internet Explorer, use Design Manager to generate an HTML snippet for the following ASP.NET markup:

```
<SharePoint:AspMenu
 ID="FooterNavigationMenu"
 runat="server"
 DataSourceID="bottomSiteMap"
 UseSimpleRendering="true"
 UseSeparateCss="false"
 Orientation="Horizontal"
 StaticDisplayLevels="1"
 MaximumDynamicDisplayLevels="0"
 SkipLinkText="" />
```



**Note:** This ASP.NET element creates an **AspMenu** instance that displays navigation data provided by the data source that you added in the previous task.

- Switch back to Visual Studio.
- In the **ContosoPrototype1.html** file, within the **footer** element, within the **div** element, add the new HTML snippet below the existing code.
- Save the file and switch back to Internet Explorer.

- Refresh the master page preview page.
- Verify that the master page preview now displays four navigation links labeled **North**, **South**, **East**, and **West** in the page footer.

**Results:** After completing this exercise, you should have added custom navigation controls to a SharePoint master page.

## Module Review and Takeaways

In this module, you learned about many aspects of branding and navigation in SharePoint 2013. First, you learned about how to create and apply themes, which you can use with any SharePoint site. You learned about the new publishing site design process, and how you can build master pages and page layouts using HTML and CSS. You learned about how to create and configure device channels and image renditions. Finally, you learned about the navigation architecture in SharePoint 2013 and how you can customize the navigation experience for your users.

### Review Question(s)

#### Test Your Knowledge

Question	
Which of the following items do you not deploy in a composed look?	
Select the correct answer.	
<input type="checkbox"/>	A master page.
<input type="checkbox"/>	A color palette.
<input type="checkbox"/>	A font scheme.
<input type="checkbox"/>	A background image.
<input type="checkbox"/>	A CSS file.

#### Test Your Knowledge

Question	
You want to create a new page layout. What should you do first?	
Select the correct answer.	
<input type="checkbox"/>	Create a new master page in Design Manager.
<input type="checkbox"/>	Create or identify a page layout content type.
<input type="checkbox"/>	Create custom field controls.
<input type="checkbox"/>	Create a new page layout in Design Manager.
<input type="checkbox"/>	Create a new page layout in HTML.

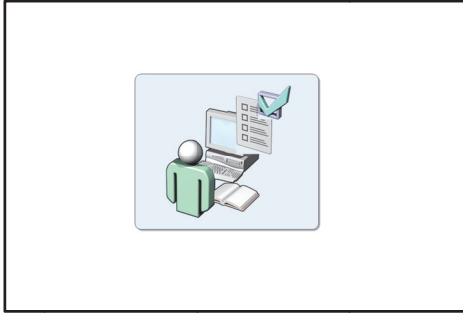
Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false: Image renditions are defined independently on each individual image on a site collection.	

### Test Your Knowledge

Question	
Which of the following navigation providers supplies site map nodes for the local part of the breadcrumb trail in SharePoint sites?	
Select the correct answer.	
	SPSiteMapProvider
	SPContentMapProvider
	SPXmlContentMapProvider
	SPNavigationProvider
	PortalSiteMapProvider

## Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

# Module 1: SharePoint as a Developer Platform

## Lab: Comparing Web Parts and App Parts

### Exercise 1: Creating and Deploying a SharePoint Web Part

#### ► Task 1: Create a Visual Web Part Project in Visual Studio 2012

1. Start the 20488B-LON-SP-01 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start page, click **New Project**.
5. In the **New Project** dialog box, in the left navigation pane, under **Templates**, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
6. In the center pane, click **SharePoint 2013 - Visual Web Part**.
7. In the **Name** box, type **GreetingWebPartProject**.
8. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
9. In the **SharePoint Customization Wizard** dialog box, under **What site do you want to use for debugging**, type **http://team.contoso.com**, and then click **Validate**.
10. In the **Microsoft Visual Studio** dialog box, verify that it reports **Connection successful**, and then click **OK**.
11. Click **Deploy as a farm solution**, and then click **Finish**.



**Note:** A specific issue prevents the running of sandboxed solution code on Windows Server 2012 servers that are domain controllers, so in this case, you will use a farm solution to deploy the Web Part. In a real world scenario, you should configure your development environment to support sandboxed solutions, but that is impractical in the classroom environment.

#### ► Task 2: Configure the Web Part Solution

1. In Solution Explorer, right-click **VisualWebPart1**, and then click **Rename**.
2. Type **GreetingWebPart**, and then press Enter.
3. Expand the **GreetingWebPart** node, right-click **VisualWebPart1.ascx**, and then click **Rename**.
4. Type **GreetingWebPart**, and then press Enter.
5. Right-click **VisualWebPart1.webpart**, and then click **Rename**.
6. Type **GreetingWebPart**, and then press Enter.
7. Review the contents of the **GreetingWebPart** node. Notice that the node contains:
  - An ASCX file and a corresponding code-behind file.
  - A .webpart file.
  - An Elements.xml file.
8. Double-click **GreetingWebPart.webpart** and review the contents of the file.

9. Edit the contents of the **property** element with the **name="Title"** attribute as follows:

```
<property name="Title" type="string">Greet Users</property>
```

10. Edit the contents of the **property** element with the **name="Description"** attribute as follows:

```
<property name="Description" type="string">Welcomes the current user with a friendly message</property>
```

11. Click **Save**, and then close the **GreetingWebPart.webpart** tab.
12. In Solution Explorer, double-click **Elements.xml**, and then review the contents of the file.



**Note:** The Elements.xml file deploys the GreetingWebPart.webpart file to the Web Part Gallery on the destination site. The list value **113** indicates a list of type Web Part Gallery, and the URL of **\_catalog/wp** is the site-relative URL for the Web Part Gallery.

13. Edit the **Property** element with the **Name="Group"** attribute as follows:

```
<Property Name="Group" Value="Contoso Web Parts" />
```

14. Click **Save**, and then close the **Elements.xml** tab.
15. In Solution Explorer, expand **Features**, right-click **Feature1**, and then click **Rename**.
16. Type **GreetingWebPart**, and then press Enter.
17. Under **Features**, double-click **GreetingWebPart**.
18. Notice that the Feature is scoped at the **Site** level.
19. In the **Title** box, type **Greet Users Web Part**.
20. In the **Description** box, type **Adds the Greet Users web part to the gallery**.
21. In the **Items in the Feature** pane, expand **Files**. Notice that the Feature includes two files: the **Elements.xml** file and the **GreetingWebPart.webpart** file.
22. On the **Manifest** tab, notice how the Feature contains an element manifest and an element file.
23. Click **Save**, and then close the **GreetingWebPart.feature** tab.
24. In Solution Explorer, double-click **Package**.
25. Notice that the solution package will deploy both the Feature and the **GreetingWebPart** project item, which is essentially the assembly for the ASCX control.
26. On the **Manifest** tab, review the solution manifest file.
27. Close the **Package.package** tab.

### ► Task 3: Create the Web Part User Interface

1. In Solution Explorer, double-click **GreetingWebPart.ascx**.
2. At the bottom of the design pane, click **Design**.
3. Click **Toolbox**, expand **HTML**, and then drag and drop a **Div** element into the dotted rectangle on the design pane.
4. In the **Properties** pane, in the **(id)** row, type **greeting** and then press Enter.





**Note:** Enclosing your controls in a named **div** element is not essential. However, it can be useful if you want to style your controls with CSS.

5. Click **Toolbox**, expand **Standard** (if it is not already expanded), and drag a **Label** control onto the **div** element on the design surface.
6. In the **Properties** pane, in the **Text** row, delete the text **Label**.
7. In the **(ID)** row, type **lblGreeting**, and then press Enter.
8. At the bottom of the design pane, click **Source**.
9. Verify that the page contains the following markup beneath the ASP.NET directives:

```
<asp:Label ID="lblGreeting" runat="server"></asp:Label>
<div id="greeting">
</div>
```

#### ► Task 4: Add Code to Greet the Current User

1. At the bottom of the source pane, click **Design**.
2. Right-click the design surface, and then click **View Code**.
3. In the class declaration, double-click **VisualWebPart1** to select the text, and then type **GreetingWebPart**.
4. Press CTRL+., and then click **Rename 'VisualWebPart1' to 'GreetingWebPart'**.



**Note:** This operation changes the name of the class everywhere it appears in the solution, such as in the **type** element in the Web Parts control definition file.

5. At the top of the code file, immediately under the last **using** directive, type the following code, and then press Enter:

```
using Microsoft.SharePoint;
```

6. Within the **Page\_Load** method, add the following code, and then press Enter:

```
var user = SPContext.Current.Web.CurrentUser;
var greeting = String.Format("Welcome to the Contoso intranet portal, {0}!", user.Name);
lblGreeting.Text = greeting;
```

7. On the **BUILD** menu, click **Build Solution**.
8. Verify that your code builds without errors.

#### ► Task 5: Test the Web Part

1. On the **DEBUG** menu, click **Start Without Debugging**.



**Note:** In the classroom environment, the virtual machine performs better if you test the Web Part without attaching the debugger.

2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. When the **Contoso Team Site** page loads, on the ribbon, on the **PAGE** tab, click **Edit**.

4. On the **INSERT** tab, click **Web Part**.
5. In the **Categories** list, click **Contoso Web Parts**.
6. In the **Parts** list, click **Greet Users**, and then click **Add**.
7. Verify that the **Greet Users** Web Part greets the current user by name.
8. On the ribbon, on the **Save** menu, click **Stop Editing**.
9. In the **Save Changes** dialog box, click **Discard changes**.
10. Close Internet Explorer.

**Results:** After completing this exercise, you should have created and deployed a Web Part that greets the current user.

## Exercise 2: Creating and Deploying a SharePoint App Part

### ► Task 1: Create a SharePoint App Project in Visual Studio 2012

1. In Visual Studio 2012, on the **FILE** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, in the left navigation pane, under **Templates**, expand **Visual C#**, expand **Office/SharePoint**, and then click **Apps**.
3. In the center pane, click **App for SharePoint 2013**.
4. In the **Name** box, type **GreetingAppProject**.
5. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
6. In the **New app for SharePoint** dialog box, under **What is the name of your app for SharePoint**, type **Greet Users**.
7. Under **What SharePoint site do you want to use for debugging your app**, type **http://dev.contoso.com**, and then click **Validate**.
8. In the **Microsoft Visual Studio** dialog box, click **OK**.
9. In the **New app for SharePoint** dialog box, under **How do you want to host your app for SharePoint**, click **SharePoint-hosted**, and then click **Finish**.

### ► Task 2: Review the Contents of the App Project

1. In Solution Explorer, expand the **Content** node. Notice that the node contains a CSS file (App.css), together with an element manifest that deploys the CSS file (Elements.xml).
2. In Solution Explorer, expand the **Images** node. Notice that the node contains an image (AppIcon.png), together with an element manifest that deploys the image (Elements.xml).



**Note:** The AppIcon.png image is the default icon that represents your app on the Site Contents page in SharePoint. In most cases, you would replace this file with your own icon or logo.

3. In Solution Explorer, expand the **Pages** node. Notice that the node contains an ASPX page (Default.aspx), together with an element manifest that deploys the page (Elements.xml).



**Note:** The Default.aspx file is the default page for the app. The default page is displayed when a user first launches a full-page app. You can edit this page or make a different page your default page.

4. In Solution Explorer, expand the **Scripts** node. Notice that the node contains several JavaScript files, including the jQuery library, together with an element manifest that deploys the JavaScript files (Elements.xml).
5. In Solution Explorer, expand **Features**, right-click **Feature1**, and then click **Rename**.
6. Type **GreetingResources**, and then press Enter.
7. Double-click **GreetingResources**. Notice that the Feature deploys the files from the **Pages**, **Scripts**, **Content**, and **Images** nodes.



**Note:** The Feature provisions all of these items to the app web—not the host web—when a user installs the app.

8. In Solution Explorer, under **Scripts**, double-click **App.js**, and review the contents of the file.



**Note:** App.js is the default code file for the app. By default, the App.js file includes a method that attempts to get the name of the current user. The success callback function uses jQuery to inject a greeting message into the Default.aspx page.

9. In Solution Explorer, double-click **AppManifest.xml**. Notice that the app manifest defines various app properties, including the icon and the start page.
10. Close all open tabs.

### ► Task 3: Add a Client Web Part to the App

1. In Solution Explorer, right-click the **GreetingAppProject** project node, point to **Add**, and then click **New Item**.
2. In the **Add New Item - GreetingAppProject** dialog box, click **Client Web Part (Host Web)**.
3. In the **Name** box, type **GreetUserPart**, and then click **Add**.
4. In the **Create Client Web Part** dialog box, click **Finish**.
5. On the **GreetUserPart.aspx** page, on a new line immediately after the final closing **</script>** tag, type the following code:

```
<script type="text/javascript" src="../../Scripts/Greeting.js"></script>
```



**Note:** You will add the Greeting.js file to your project shortly.

6. On a new line immediately after the opening **<body>** tag, type the following code:

```
<div>
 <p id="greeting">
 Loading your app...
 </p>
</div>
```

7. Save and close the **GreetUserPart.aspx** page.
8. In Solution Explorer, expand **GreetUserPart**, and then double-click **Elements.xml**.
9. In the **Elements.xml** file, in the **ClientWebpart** element, change the value of the **Title** attribute to **Greet Users**.
10. In the **ClientWebpart** element, change the value of the **Description** attribute to **Welcomes the current user with a friendly message**.
11. Save and close the **Elements.xml** file.

#### ► Task 4: Add Code to Greet the Current User

1. In Solution Explorer, right-click **Scripts**, point to **Add**, and then click **Existing Item**.
2. In the **Add Existing Item - GreetingAppProject** dialog box, browse to **E:\Labfiles\Starter\Snippets**, click **Greeting.js**, and then click **Add**.
3. Double-click **Greeting.js**, and then review the contents of the file. The code in the file performs the following actions:
  - a. It retrieves the name of the current user.
  - b. It creates a personalized greeting for the current user, and inserts it into the HTML element with the **id** value of **greeting**.



**Note:** You do not need to understand how the JavaScript code works in detail at this stage. You will learn more about working with the client-side object model later in this course.

#### ► Task 5: Test the App Part

1. Click **Start**.
2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. When the **Greet Users** page loads, click **Contoso Development Site**.
4. On the ribbon, on the **PAGE** tab, click **Edit**.
5. On the **INSERT** tab, click **App Part**.
6. In the **Parts** list, click **Greet Users**, and then click **Add**.
7. Verify that the **Greet User** app part welcomes the current user by name.
8. On the ribbon, on the **Save** menu, click **Stop Editing**.
9. In the **Save Changes** dialog box, click **Discard changes**.
10. Close Internet Explorer.

---

**Results:** After completing this exercise, you should have created and deployed an app part that greets the current user.



## Module 2: Working with SharePoint Objects

# Lab A: Working with Sites and Webs

### Exercise 1: Working with Sites and Webs in Managed Code

#### ► Task 1: Create Local Variables

1. Start the 20488B-LON-SP-02 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password Pa\$\$w0rd.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start page, click **Open Project**.
5. In the **Open Project** dialog box, browse to **E:\Labfiles\Starter\TitleChecker\_LabA**, click **TitleChecker.sln**, and then click **Open**.
6. In Solution Explorer, expand **TitleCheckerWebPart**, and then double-click **TitleCheckerWebPart.ascx**.
7. Review the contents of the page. Notice that the user control contains:
  - A list box control named **lstWebs**.
  - A panel control named **pnlUpdateControls**, which contains a text box named **txtTitle** and a button named **btnUpdate**.
  - A panel control named **pnlResult**, which contains a literal control named **litResult**.
8. Right-click the design surface, and then click **View Code**.
9. Review the code-behind file. Notice that the code-behind file contains:
  - A method that handles the **SelectedIndexChanged** event of the **lstWeb** control.
  - A method that handles the **Click** event of the **btnUpdate** control.
10. On the **VIEW** menu, click **Task List**.
11. In the **Task List** pane, on the drop-down menu, click **Comments**.
12. In the **Task List** pane, double-click **TODO: Ex 1 Task 1 Add variables to track the web part state**.
13. Add the following code:

```
Guid selectedSiteGuid = Guid.Empty;
bool siteUpdated = false;
```

#### ► Task 2: Display a List of Webs in the Current Site Collection

1. In the **Task List** window, double-click **TODO: Ex 1 Task 2 Hide the update controls by default**.
2. Add the following code:

```
pnlUpdateControls.Visible = false;
pnlResult.Visible = false;
```
3. In the **Task List** window, double-click **TODO: Ex 1 Task 2 Populate the ListBox with the list of webs in the current site**.

4. Add the following code:

```
var site = SPContext.Current.Site;
lstWebs.Items.Clear();
foreach (SPWeb web in site.AllWebs)
{
 try
 {
 lstWebs.Items.Add(new ListItem(web.Title, web.ID.ToString()));
 }
 finally
 {
 web.Dispose();
 }
}
```

► **Task 3: Respond to a User Selecting a Web**

1. In the **Task List** window, double-click **TODO: Ex 1 Task 3 Get the GUID of the selected list item**.
2. Add the following code:

```
selectedSiteGuid = new Guid(lstWebs.SelectedValue);
```

3. In the **Task List** window, double-click **TODO: Ex 1 Task 3 Set the title text box to the title of the selected site**.
4. Add the following code:

```
txtTitle.Text = lstWebs.SelectedItem.Text;
```

5. In the **Task List** window, double-click **TODO: Ex 1 Task 3 If the user has selected an item, reselect the item in the list and display the update controls**.
6. Add the following code:

```
if (!selectedSiteGuid.Equals(Guid.Empty))
{
 lstWebs.Items.FindByValue(selectedSiteGuid.ToString()).Selected = true;
 pnlUpdateControls.Visible = true;
}
```

► **Task 4: Update Web Titles When the User Clicks a Button**

1. In the **Task List** window, double-click **TODO: Ex 1 Task 4 Get the GUID of the selected list site**.
2. Add the following code:

```
selectedSiteGuid = new Guid(lstWebs.SelectedValue);
```

3. In the **Task List** window, double-click **TODO: Ex 1 Task 4 Get the new title for the selected site**.
4. Add the following code:

```
string newTitle = txtTitle.Text;
```

5. In the **Task List** window, double-click **TODO: Ex 1 Task 4 Update the title of the selected site and display a confirmation message**.



6. Add the following code:

```
if(!String.IsNullOrEmpty(newTitle) && !selectedSiteGuid.Equals(Guid.Empty))
{
 using (SPWeb web = SPContext.Current.Site.OpenWeb(selectedSiteGuid))
 {
 web.Title = newTitle;
 web.Update();
 litResult.Text = String.Format("The title of the site at <i>{0}</i> has been changed to <i>{1}</i>.", web.Url, newTitle);
 }
 siteUpdated = true;
}
```

7. In the **Task List** window, double-click **TODO: Ex 1 Task 4 If a site has been updated, clear the selected item and display the results label, otherwise hide the results label.**
8. Add the following code:

```
if (siteUpdated)
{
 lstWebs.SelectedIndex = -1;
 selectedSiteGuid = Guid.Empty;
 pnlResult.Visible = true;
}
```

#### ► Task 5: Test the Web Part

1. On the **DEBUG** menu, click **Start Without Debugging**.
2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. In Internet Explorer, after the page finishes loading, on the ribbon, on the **PAGE** tab, click **Edit Page**.
4. In the **Left** zone, click **Add a Web Part**.
5. In the **Categories** list, click **Contoso Web Parts**.
6. In the **Parts** list, click **Title Checker**, and then click **Add**.
7. On the ribbon, click **Stop Editing**.
8. In the **Title Checker** Web Part, in the list box, select any web.
9. Verify that the Web Part displays a text box containing the title of the selected web, together with an **Update** button.
10. In the text box, type a new title, and then click **Update**.
11. Verify that:
  - a. The Web Part displays a message confirming the action taken.
  - b. The update controls (the text box and the **Update** button) are no longer visible.
  - c. The new web title is reflected in the list of webs.
12. Close all open windows.

**Results:** After completing this exercise, you should have developed and tested a visual Web Part that enables users to select and update web titles.

## Exercise 2: Working with Sites and Webs in Windows PowerShell.

### ► Task 1: Create a Windows PowerShell Script to Retrieve and Update Web Titles

1. On the Start screen, type **Windows PowerShell ISE**, and then click **Windows PowerShell ISE**.
2. To clear the window when the script runs, in the script editor pane (the pane in the upper-left with line numbers), add the following code:

```
cls
```

3. To load the SharePoint snap-in for Windows PowerShell, add the following code:

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

4. To get a site collection URL from the user, add the following code:

```
$siteUrl = Read-Host "Please provide the site URL"
```

5. To create an **SPSite** object from the URL provided by the user, add the following code:

```
$site = Get-SPSite -Identity $siteUrl -ErrorAction Ignore
if($site) {
 Write-Host "Found site: $($site.ID)"
}
else {
 Write-Host "Unable to find site at $siteUrl"
 break
}
```

6. To enumerate the webs within the selected site collection, add the following code:

```
Write-Host "The site contains the following webs: "
foreach($web in $site.AllWebs)
{
 Write-Host
 Write-Host $web.Title
 $web.Dispose()
}
```

7. To give users the opportunity to change the title of each web, within the **foreach** block, on a new line immediately before the **\$web.Dispose()** statement, add the following code:

```
$choice = Read-Host "Edit this web? (Y/N) (N is default)"
if($choice -eq "y")
{
 # Update the site title.
 $newTitle = Read-Host "Please provide the new web title, or press Enter to cancel"
 if($newTitle) {
 Write-Host "Updating web title..." -ForegroundColor Gray
 $web.Title = $newTitle
 $web.Update()
 Write-Host "Web title updated." -ForegroundColor Green
 }
}
```

8. Click **Save**, and save the script as **E:\Labfiles\Starter\TitleUpdater.ps1**.

## ► Task 2: Test the Windows PowerShell Script

1. In the Windows PowerShell ISE window, click **Run Script**.



**Note:** The **Run Script** command is represented by a green arrow icon on the toolbar.

2. In the command window, when you are prompted to provide a site URL, type **http://projects.contoso.com**, and then press Enter.
3. When the command window displays the title of the first web, press Enter repeatedly. Verify that the script moves on to the next web every time you press Enter.
4. Continue to press Enter until the script finishes running, and then click **Run Script** again.
5. When you are prompted to provide a site URL, type **http://projects.contoso.com**, and then press Enter.
6. When the command window displays the title of the first web, type **y**, and then press Enter.
7. Type a new title, and then press Enter. Verify that the command window displays a confirmation message before moving on to the next web in the collection.
8. For each remaining web in the collection, you can either type **y** to edit the web title or press Enter to skip the web.

**Results:** After completing this exercise, you should have created a Windows PowerShell script that enables users to update the title of each web within a site collection.

# Lab B: Working with Execution Contexts

## Exercise 1: Running Code with Elevated Privileges

### ► Task 1: Test the Web Part as a Site Member

1. On the Start screen, type **Internet Explorer**, and then press Enter.
2. In the address bar, type **http://projects.contoso.com**, and then press Enter.
3. When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.
4. Notice that you are denied access to the home page, despite being a site member.



**Note:** Only users who are granted the **ManageWeb** permission can enumerate the webs in a site collection. This permission is granted by the **Full Control** permission level.

5. Close Internet Explorer.

### ► Task 2: Modify Code to Run with Elevated Privileges

1. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
2. On the Start page, click **Open Project**.
3. In the **Open Project** dialog box, browse to **E:\Labfiles\Starter\TitleChecker\_LabB**, click **TitleChecker.sln**, and then click **Open**.
4. In Solution Explorer, expand **TitleCheckerWebPart**, expand **TitleCheckerWebPart.ascx**, and then double-click **TitleCheckerWebPart.ascx.cs**.
5. If the **Task List** pane is not already visible, on the **VIEW** menu, click **Task List**.
6. In the **Task List** pane, on the drop-down menu, click **Comments**.
7. In the **Task List** pane, double-click **TODO: Ex 1 Task 1 Declare a GUID variable**.
8. Add the following code:

```
Guid siteCollID = Guid.Empty;
```

9. In the **Task List** pane, double-click **TODO: Ex 1 Task 1 Populate the ListBox with the list of webs in the current site**.
10. Select the following code:

```
var site = SPContext.Current.Site;
lstWebs.Items.Clear();
foreach(SPWeb web in site.AllWebs)
{
 try
 {
 lstWebs.Items.Add(new ListItem(web.Title, web.ID.ToString()));
 }
 finally
 {
 web.Dispose();
 }
}
```

11. Right-click the selection, point to **Refactor**, and then click **Extract Method**.

12. In the **Extract Method** dialog box, in the **New method name** box, type **PopulateWebsList**, and then click **OK**.

13. Delete the following code:

```
PopulateWebsList();
```

14. Add the following code in place of the method call you just deleted:

```
siteCollID = SPContext.Current.Site.ID;
var populateWebsList = new SPSecurity.CodeToRunElevated(PopulateWebsList);
SPSecurity.RunWithElevatedPrivileges(populateWebsList);
```

15. In the **Task List** pane, double-click **TODO: Ex 1 Task 1 Populate the ListBox with the list of webs in the current site**.

16. Locate the **PopulateWebsList** method. Within the method, delete the following line of code:

```
var site = SPContext.Current.Site;
```

17. In place of the statement you just deleted, add the following code:

```
using (var site = new SPSite(siteCollID))
{
```

18. On a new line immediately before the closing curly bracket of the **PopulateWebsList** method, add another closing curly bracket to close the **using** block:

```
}
```

19. On the **BUILD** menu, click **Rebuild Solution**. Verify that the solution builds without errors.

20. On the **BUILD** menu, click **Deploy Solution**.

### ► Task 3: Test the Web Part as a Site Member

1. On the Start screen, type **Internet Explorer**, and then press Enter.

2. In the address bar, type **http://projects.contoso.com**, and then press Enter.

3. When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.

4. In Internet Explorer, when the home page loads, verify that the **Title Checker** Web Part now displays the list of webs correctly.

5. In the **Title Checker** Web Part, in the list box, select any web.

6. In the text box, type a new title, and then click **Update**.

7. Notice that nothing happens. This is because the call to **SPWeb.Update** in the **btnUpdate\_Click** method is throwing an exception of type **UnauthorizedAccessException**.

8. Close Internet Explorer.

**Results:** After completing this exercise, you should have configured the Title Checker Web Part to use elevated privileges to populate the list of webs.

## Exercise 2: Adapting Content for Different User Permissions

### ► Task 1: Modify the Web Part to Hide Controls from Unauthorized Users

1. In Visual Studio, in Solution Explorer, double-click **TitleCheckerWebPart.ascx**.
2. If you are not already in source view, at the bottom of the design pane, click **Source**.
3. On a new line, immediately before the opening tag of the **pnlUpdateControls** element, add the following code:

```
<SharePoint:SPSecurityTrimmedControl runat="server" PermissionsString="ManageWeb">
```



**Note:** Visual Studio will automatically generate a closing tag for you. Delete this closing tag.

4. On a new line, immediately after the closing tag of the **pnlUpdateControls** element, add the following code:

```
</SharePoint:SPSecurityTrimmedControl>
```

5. On the **BUILD** menu, click **Rebuild Solution**.
6. On the **DEBUG** menu, click **Start Without Debugging**.
7. When you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
8. In Internet Explorer, when the page loads, in the **Title Checker** Web Part, in the list box, select any web.
9. In the text box, type a new title, and then click **Update**.
10. Verify that the Web Part updates the title.



**Note:** Because you are testing the Web Part as an administrator, the user experience should be unchanged at this point.

11. Close Internet Explorer.

### ► Task 2: Test the Web Part As a Site Member

1. On the Start screen, type **Internet Explorer**, and then press Enter.
2. In the address bar, type **http://projects.contoso.com**, and then press Enter.
3. When you are prompted for credentials, log on as **CONTOSO\lukas** with the password **Pa\$\$w0rd**.
4. In Internet Explorer, when the home page loads, verify that the **Title Checker** Web Part now displays the list of webs correctly.
5. In the **Title Checker** Web Part, in the list box, select any web.
6. Verify that the Web Part does not display the update controls.



**Note:** Because you are testing the Web Part as a regular site member, the **SPSecurityTrimmedControl** prevents ASP.NET from rendering the update controls.

---

**Results:** After completing this exercise, you should have configured the Title Checker Web Part to use a security-trimmed control to hide the update controls from users with insufficient permissions.





## Module 3: Working with Lists and Libraries

# Lab A: Querying and Retrieving List Data

### Exercise 1: Querying List Items

#### ► Task 1: Review the Starter Code

1. Start the 20488B-LON-SP-03 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password Pa\$\$w0rd.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start page, click **Open Project**.
5. In the **Open Project** dialog box, browse to **E:\Labfiles\Starter\ExpenseChecker\_LabA**, click **ExpenseChecker.sln**, and then click **Open**.
6. In Solution Explorer, expand **ExpenseCheckerWebPart**, and then double-click **ExpenseCheckerWebPart.ascx**.
7. Review the contents of the page. Notice that the user control contains an ASP.NET **ListView** control with the following row headers:
  - a. Unnamed row
  - b. Requestor
  - c. Category
  - d. Description
  - e. Amount
8. Notice that the page also includes buttons named **btnApprove** and **btnReject**.
9. Right-click the design surface, and then click **View Code**.
10. Review the code-behind file. Notice that the **btnApprove\_Click** and the **btnReject\_Click** both invoke a method named **UpdateItems**, which retrieves the collection of selected items from the **ListView** control.

#### ► Task 2: Create a Query Object

1. On the **VIEW** menu, click **Task List**.
2. In the **Task List** pane, on the drop-down menu, click **Comments**.
3. In the **Task List** pane, double-click **TODO: Ex 1 Task 2 Create and configure a query object**.
4. Add the following code, and then press Enter:

```
SPQuery query = new SPQuery();
```

5. Add the following code, and then press Enter:

```
query.Query = @"
 <Where>
 <And>
 <Leq>
 <FieldRef Name=""CapExAmount""></FieldRef>
 <Value Type=""Currency"">500.00</Value>
 </Leq>
 <Eq>
 <FieldRef Name=""CapExStatus""></FieldRef>
 <Value Type=""Choice"">Pending</Value>
 </Eq>
 </And>
 </Where>
";
```

6. Add the following code:

```
query.ViewFields = @"
 <FieldRef Name=""UniqueId"" />
 <FieldRef Name=""CapExRequestor"" />
 <FieldRef Name=""CapExCategory"" />
 <FieldRef Name=""CapExDescription"" />
 <FieldRef Name=""CapExAmount"" />
";
```

### ► Task 3: Query the Expenditure Requests List

1. In the **Task List** pane, double-click **TODO: Ex 1 Task 3 Query the list and bind the results to the lstExpenses control**.
2. Add the following code:

```
var web = SPContext.Current.Web;
var list = web.Lists["Expenditure Requests"];
var items = list.GetItems(query);
lstExpenses.DataSource = items.GetDataTable();
```

3. `lstExpenses.DataBind();`

### ► Task 4: Bind List Item Fields to the ListView Control

1. In Solution Explorer, double-click **ExpenseCheckerWebPart.ascx**.
2. Within the **asp:ListView** element, within the **ItemTemplate** element, locate the **asp:HiddenField** element with an **ID** value of **hiddenID**.
3. Amend the **asp:HiddenField** element to the following:

```
<asp:HiddenField ID="hiddenID" runat="server" Value='<%= Eval("UniqueId") %>' />
```

4. Locate the **asp:Label** element with an **ID** value of **lblRequestor**.
5. Amend the **asp:Label** element to the following:

```
<asp:Label ID="lblRequestor" runat="server" Text='<%= Eval("CapExRequestor") %>' />
```

6. Locate the **asp:Label** element with an **ID** value of **lblCategory**.

- Amend the **asp:Label** element to the following:

```
<asp:Label ID="lblCategory" runat="server" Text='<%=# Eval("CapExCategory") %>' />
```

- Locate the **asp:Label** element with an **ID** value of **lblDescription**.

- Amend the **asp:Label** element to the following:

```
<asp:Label ID="lblDescription" runat="server" Text='<%=# Eval("CapExDescription") %>' />
```

- Locate the **asp:Label** element with an **ID** value of **lblAmount**.

- Amend the **asp:Label** element to the following:

```
<asp:Label ID="lblAmount" runat="server" Text='<%=# Eval("CapExAmount", "{0:C2}") %>' />
```

### ► Task 5: Test the Web Part

- On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
- On the **DEBUG** menu, click **Start Without Debugging**.
- If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
- In Internet Explorer, when the page loads, on the ribbon, click **EDIT**.
- On the **INSERT** tab, click **Web Part**.
- In the **Categories** list, click **Contoso Web Parts**.
- In the **Parts** list, click **Expense Checker**, and then click **Add**.
- On the ribbon, click **SAVE**.
- Verify that the **Expense Checker** Web Part displays several list items with the following information:
  - The requestor name
  - The expense category
  - The expense description
  - The amount
- Close Internet Explorer.

**Results:** After completing this exercise, you should have created a visual Web Part that retrieves and displays data from a SharePoint list.

## Exercise 2: Updating List Items

### ► Task 1: Update the Status of Selected List Items

1. In the **Task List** pane, double-click **TODO: Ex 2 Task 1 Update the status of the list items**.
2. Add the following code:

```
var web = SPContext.Current.Web;
var list = web.Lists["Expenditure Requests"];
foreach (var selectedItem in selectedItems)
{
 // Get the unique identifier for each list item.
 var hiddenField = selectedItem.FindControl("hiddenID") as HiddenField;
 Guid itemID;
 if(Guid.TryParse(hiddenField.Value, out itemID))
 {
 // Retrieve the list item and update the status.
 SPListItem item = list.GetItemById(itemID);
 item["Request Status"] = status;
 item.Update();
 }
}
```

### ► Task 2: Test the Web Part

1. On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
2. On the **DEBUG** menu, click **Start Without Debugging**.
3. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
4. In Internet Explorer, when the page loads, on the **Expense Checker** Web Part, select three or four items, and then click **Approve**. Verify that the items are no longer displayed in the Web Part.
5. On the **Expense Checker** Web Part, select another three or four items, and then click **Reject**. Verify that the items are no longer displayed in the Web Part.
6. On the Quick Launch navigation menu, click **Expenditure Requests**.
7. Verify that the **Expenditure Requests** list includes the items with statuses of **Approved** and **Rejected** that you amended in the Web Part.
8. Close Internet Explorer.
9. Close Visual Studio.

**Results:** After completing this exercise, you should have configured the Expense Checker Web Part to enable users to approve or reject multiple expense items simultaneously.

# Lab B: Working With Large Lists

## Exercise 1: Using the ContentIterator Class

### ► Task 1: Add Assembly References and Using Statements

1. Connect to the 20488B-LON-SP-03 virtual machine.
2. If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start page, click **Open Project**.
5. In the **Open Project** dialog box, browse to **E:\Labfiles\Starter\ExpenseChecker\_LabB**, click **ExpenseChecker.sln**, and then click **Open**.
6. In Solution Explorer, right-click **References**, and then click **Add Reference**.
7. In the **Reference Manager - ExpenseChecker** dialog box, in the search box, type **Microsoft.Office.Server**.
8. In the center pane, select **Microsoft.Office.Server** (ensure the check box is selected), and then click **OK**.
9. In Solution Explorer, expand **ExpenseCheckerWebPart**, expand **ExpenseCheckerWebPart.ascx**, and then double-click **ExpenseCheckerWebPart.ascx.cs**.
10. At the top of the code file, on a new line immediately below the existing **using** statements, add the following code:

```
using Microsoft.Office.Server.Utilities;
using System.Data;
```

### ► Task 2: Create a Data Table to Store Query Results

1. In the **Task List** pane, double-click **TODO: Ex 1 Task 2 Declare a DataTable variable**.
2. Add the following code:

```
DataTable dtable;
```

3. In the **Task List** pane, double-click **TODO: Ex 1 Task 2 Instantiate the DataTable and add columns**.
4. Add the following code:

```
dtable = new DataTable();
dtable.Columns.Add("UniqueId");
dtable.Columns.Add("CapExRequestor");
dtable.Columns.Add("CapExCategory");
dtable.Columns.Add("CapExDescription");
```

5. `dtable.Columns.Add("CapExAmount", typeof(Decimal));`

### ► Task 3: Create the ProcessItem Method

1. In the **Task List** pane, double-click **TODO: Ex 1 Task 3 Add the ProcessItem method**.
2. Add the following code:

```
private void ProcessItem(SPListItem item)
{
 string uniqueId = item["UniqueId"].ToString();
 var field = item.Fields["Requested By"] as SPFieldUser;
 var fieldValue = field.GetFieldValue(item["Requested By"].ToString()) as SPFieldUserValue;
 string username = fieldValue.User.Name;
 string category = item["Category"].ToString();
 string description = item["Description"].ToString();
 decimal amount = Decimal.Parse(item["Amount"].ToString());
 dtable.Rows.Add(uniqueId, username, category, description, amount);
}
```

#### ► Task 4: Create the ProcessError Method

1. In the **Task List** pane, double-click **TODO: Ex 1 Task 4 Add the ProcessError method**.
2. Add the following code:

```
private bool ProcessError(SPListItem item, Exception e)
{
 throw new Exception("An error occurred during item processing", e);
}
```

#### ► Task 5: Configure a ContentIterator Instance

1. In the **Task List** pane, double-click **TODO: Ex 1 Task 5 Create and configure a ContentIterator object**.
2. Select and delete the following code:

```
var items = list.GetItems(query);
lstExpenses.DataSource = items.GetDataTable();
lstExpenses.DataBind();
```

3. Add the following code:

```
ContentIterator iterator = new ContentIterator();
iterator.ProcessListItems(list, query, ProcessItem, ProcessError);
lstExpenses.DataSource = dtable;
lstExpenses.DataBind();
```

#### ► Task 6: Test the Web Part

1. On the **BUILD** menu, click **Rebuild Solution**, and verify that your code builds without errors.
2. On the **DEBUG** menu, click **Start Without Debugging**.
3. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
4. In Internet Explorer, when the page loads, verify that the Expense Checker Web Part behaves as before.
5. Close Internet Explorer and close Visual Studio.

**Results:** After completing this exercise, you should have modified the Expense Checker Web Part to use the **ContentIterator** class.

# Module 4: Designing and Managing Features and Solutions

## Lab: Working With Features and Solutions

### Exercise 1: Configuring SharePoint Features

#### ► Task 1: Create a new SharePoint project in Visual Studio

1. Start the 20488B-LON-SP-04 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
4. On the Start Page, click **New Project**.
5. In the **New Project** dialog box, in the navigation pane, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
6. In the center pane, click **SharePoint 2013 – Empty Project**.
7. In the **Name** box, type **ContractorAgreements**.
8. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
9. In the **SharePoint Customization Wizard** dialog box, under **What site do you want to use for debugging**, type **http://team.contoso.com**.
10. Select **Deploy as a farm solution**, and then click **Finish**.



**Note:** In a production environment, you would deploy these components in a sandboxed solution. However, SharePoint 2013 will not allow you to execute sandboxed code on a domain controller. As such, you must use a farm solution in the classroom environment.

#### ► Task 2: Add site columns to the solution

1. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
2. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.
3. In the **Name** box, type **ContosoManager**, and then click **Add**.
4. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="ContosoManager"
 DisplayName="Contoso Manager"
 Type="User"
 UserSelectionMode="0"
 Required="TRUE"
 Group="Contoso Site Columns">
</Field>
```

5. Click **Save**, and then close the Elements.xml tab.
6. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
7. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.

8. In the **Name** box, type **ContosoTeam**, and then click **Add**.
9. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="ContosoTeam"
 DisplayName="Contoso Team"
 Type="Choice"
 Required="TRUE"
 Group="Contoso Site Columns">
 <CHOICES>
 <CHOICE>Research</CHOICE>
 <CHOICE>Production</CHOICE>
 <CHOICE>Audit</CHOICE>
 <CHOICE>IT</CHOICE>
 </CHOICES>
</Field>
```

10. Click **Save**, and then close the Elements.xml tab.
11. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
12. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.
13. In the **Name** box, type **DailyRate**, and then click **Add**.
14. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="DailyRate"
 DisplayName="Daily Rate"
 Type="Currency"
 LCID="1033"
 Decimals="0"
 Required="TRUE"
 Group="Contoso Site Columns">
</Field>
```

15. Click **Save**, and then close the Elements.xml tab.
16. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
17. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.
18. In the **Name** box, type **AgreementStartDate**, and then click **Add**.
19. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="AgreementStartDate"
 DisplayName="Agreement Start Date"
 Type="DateTime"
 Format="DateOnly"
 Required="TRUE"
 Group="Contoso Site Columns">
</Field>
```



20. Click **Save**, and then close the Elements.xml tab.
21. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
22. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.
23. In the **Name** box, type **AgreementEndDate**, and then click **Add**.
24. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="AgreementEndDate"
 DisplayName="Agreement End Date"
 Type="DateTime"
 Format="DateOnly"
 Required="TRUE"
 Group="Contoso Site Columns">
</Field>
```


25. Click **Save**, and then close the Elements.xml tab.
26. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
27. In the **Add New Item - ContractorAgreements** dialog box, click **Site Column**.
28. In the **Name** box, type **SecurityCleared**, and then click **Add**.
29. In the Elements.xml file, edit the **Field** element to resemble the following (leave the ID attribute unchanged):

```
<Field
 ID="{...}"
 Name="SecurityCleared"
 DisplayName="Security Cleared"
 Type="Boolean"
 Required="TRUE"
 Group="Contoso Site Columns">
</Field>
```

30. Click **Save**, and then close the Elements.xml tab.

### ► Task 3: Create and configure a Feature

1. In Solution Explorer, under **Features**, right-click **Feature1**, and then click **Delete**.
2. In the **Microsoft Visual Studio** dialog box, click **OK**.

 **Note:** Visual Studio creates a Feature automatically when you add declarative items to a SharePoint project. However, in this exercise you will create a new Feature to help you to gain an understanding of the complete process.

3. Right-click **Features**, and then click **Add Feature**.
4. In Solution Explorer, right-click **Feature1**, and then click **Rename**.
5. Type **ContractingResources**, and then press Enter.
6. On the **ContractingResources.feature** tab, in the **Title** box, type **Contractor Management Resources**.

7. In the **Description** box, type **Provisions site columns and content types for managing contractor agreements**.
8. In the **Scope** drop-down list box, click **Site**.
9. Click the double right arrow button (>>) to add all the site columns to the Feature.
10. Click **Save**.
11. On the Manifest tab, review the Feature manifest file that Visual Studio has generated for you, and then close the **ContractingResources.feature** tab.


**Results:** After completing this exercise, you should have configured a Site-scoped Feature to deploy several site columns.

## Exercise 2: Creating Feature Receiver Classes

### ► Task 1: Create a Feature receiver class

1. In Solution Explorer, under **Features**, right-click **ContractingResources**, and then click **Add Event Receiver**.
2. In the **ContractingResourcesEventReceiver** class, immediately after the opening brace, add the following line of code:

```
public static readonly SPContentTypeId ctid = new
SPContentTypeId("0x010100C3316E15A95F420F8187FBBE1B9636F9");
```

 **Note:** You are creating a static content type ID to make it easy to identify and remove your content type at a later date. You will learn more about content type IDs later in this course.

### ► Task 2: Create a content type programmatically

1. Uncomment the **FeatureActivated** method.

 **Note:** If you are running short on time, you can copy the complete code for the **FeatureActivated** method from the file **E:\Labfiles\Snippets\FeatureActivated.txt**.

2. Within the **FeatureActivated** method, add the following code to create the Contractor Agreement content type:

```
var site = properties.Feature.Parent as SPSite;
var web = site.RootWeb;
SPContentType contractingCT = web.ContentTypes[ctid];
if (contractingCT == null)
{
 contractingCT = new SPContentType(ctid, web.ContentTypes, "Contractor Agreement");
 web.ContentTypes.Add(contractingCT);
}
```

3. Add the following code to configure the properties of the content type:

```
contractingCT.Description = "A contractual agreement between Contoso Pharmaceuticals and an
independent contractor";
contractingCT.Group = "Contoso Content Types";
```

4. Add the following code to add the built-in Full Name column to the content type:

```
SPField fldFullName = web.AvailableFields["Full Name"];
SPFieldLink fldLinkFullName = new SPFieldLink(fldFullName);
if (contractingCT.FieldLinks[fldLinkFullName.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkFullName);
}
```

5. Add the following code to add the Contoso Manager column to the content type:

```
SPField fldContosoManager = web.AvailableFields["Contoso Manager"];
SPFieldLink fldLinkContosoManager = new SPFieldLink(fldContosoManager);
if (contractingCT.FieldLinks[fldLinkContosoManager.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkContosoManager);
}
```

6. Add the following code to add the Contoso Team column to the content type:

```
SPField fldContosoTeam = web.AvailableFields["Contoso Team"];
SPFieldLink fldLinkContosoTeam = new SPFieldLink(fldContosoTeam);
if (contractingCT.FieldLinks[fldLinkContosoTeam.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkContosoTeam);
}
```

7. Add the following code to add the Daily Rate column to the content type:

```
SPField fldDailyRate = web.AvailableFields["Daily Rate"];
SPFieldLink fldLinkDailyRate = new SPFieldLink(fldDailyRate);
if (contractingCT.FieldLinks[fldLinkDailyRate.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkDailyRate);
}
```

8. Add the following code to add the Agreement Start Date column to the content type:

```
SPField fldStartDate = web.AvailableFields["Agreement Start Date"];
SPFieldLink fldLinkStartDate = new SPFieldLink(fldStartDate);
if (contractingCT.FieldLinks[fldLinkStartDate.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkStartDate);
}
```

9. Add the following code to add the Agreement End Date column to the content type:

```
SPField fldEndDate = web.AvailableFields["Agreement End Date"];
SPFieldLink fldLinkEndDate = new SPFieldLink(fldEndDate);
if (contractingCT.FieldLinks[fldLinkEndDate.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkEndDate);
}
```

10. Add the following code to add the built-in Security Cleared column to the content type:

```
SPField fldSecurityCleared = web.AvailableFields["Security Cleared"];
SPFieldLink fldLinkSecurityCleared = new SPFieldLink(fldSecurityCleared);
if (contractingCT.FieldLinks[fldLinkSecurityCleared.Id] == null)
{
 contractingCT.FieldLinks.Add(fldLinkSecurityCleared);
}
```

11. Add the following code to persist your changes to the content database:

```
contractingCT.Update(true);
```

► **Task 3: Remove a content type programmatically**

1. Uncomment the **FeatureDeactivating** method.



**Note:** If you are running short on time, you can copy the complete code for the **FeatureDeactivating** method from the file **E:\Labfiles\Snippets\FeatureDeactivating.txt**.

2. Add the following code to the **FeatureDeactivating** method to remove the Contractor Agreement content type:

```
var site = properties.Feature.Parent as SPSite;
var web = site.RootWeb;
SPContentType contractingCT = web.ContentTypes[ctid];
if (contractingCT != null)
{
 web.ContentTypes.Delete(ctid);
}
```

3. Click **Save**.

► **Task 4: Test the Feature receiver**

1. On the **BUILD** menu, click **Rebuild Solution**.
2. On the **DEBUG** menu, click **Start Without Debugging**.
3. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
4. After the page finishes loading, on the **Settings** menu, click **Site settings**.
5. On the **Site Settings** page, under **Web Designer Galleries**, click **Site content types**.
6. On the **Site Content Types** page, under **Contoso Content Types**, verify that a content type named **Contractor Agreement** is listed.
7. Click **Contractor Agreement**.
8. Under columns, verify that the following site columns are listed:
  - a. Full Name
  - b. Contoso Manager
  - c. Contoso Team
  - d. Daily Rate
  - e. Agreement Start Date
  - f. Agreement End Date
  - g. Security Cleared
9. On the **Settings** menu, click **Site settings**.
10. On the **Site Settings** page, under **Site Collection Administration**, click **Site collection features**.
11. On the **Site Collection Features** page, in the **Contractor Management Resources** row, click **Deactivate**.

12. If a warning message appears, click **Deactivate this feature**.
13. On the **Settings** menu, click **Site settings**.
14. On the **Site Settings** page, under **Web Designer Galleries**, click **Site content types**.
15. Verify that the **Contractor Agreement** content type is no longer displayed.
16. On the **Settings** menu, click **Site settings**.
17. On the **Site Settings** page, under **Site Collection Administration**, click **Site collection features**.
18. On the **Site Collection Features** page, in the **Contractor Management Resources** row, click **Activate**.



**Note:** You will want the Feature to be active so that Visual Studio can retrieve the custom site columns from the site in the next exercise.

19. Close Internet Explorer and close Visual Studio.



**Note:** You will reopen Visual Studio in the next task. Closing and reopening Visual Studio forces Visual Studio to retrieve the current list of content types from the SharePoint site.

**Results:** After completing this exercise, you should have created and tested a Feature receiver class that creates a new content type.

### Exercise 3: Creating Features with Dependencies

#### ► Task 1: Add a list template and a list instance to the solution

1. On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
2. On the Start Page, under **Recent**, click **ContractorAgreements**.
3. In Solution Explorer, right-click the **ContractorAgreements** project node, point to **Add**, and then click **New Item**.
4. In the **Add New Item – ContractorAgreements** dialog box, click **List**.
5. In the **Name** box, type **ContractorAgreementsList**, and then click **Add**.
6. In the **SharePoint Customization Wizard** dialog box, under **What name do you want to display for your list**, type **Contractor Agreements**.
7. Under **Create a customizable list template and a list instance of it**, in the drop-down menu, click **Document Library**.
8. Click **Next**, and then click **Finish**.
9. On the **Columns** tab, scroll to the bottom of the page and click **Content Types**.
10. In the **Content Type Settings** dialog box, select the **Document** row and press **Delete**.
11. Select the **Folder** row and press **Delete**.
12. Type **Contractor Agreement**, and then click **OK**.

13. On the **List** tab, in the **Description** box, type **Use this list to manage contracts between Contoso Pharmaceuticals and external contractors.**

14. Click **Save All**, and then close the **Contractor Agreements** tab.

► **Task 2: Configure a Feature with dependencies**

1. In Solution Explorer, right click **Features**, and then click **Add Feature**.

2. In Solution Explorer, right-click **Feature1**, and then click **Rename**.

3. Type **ContractorAgreementsList**, and press Enter.

4. On the **ContractorAgreementsList.feature** tab, in the **Title** box, type **Contractor Agreements List**.

5. In the **Description** box, type **Provisions a Contractor Agreements list at the Web scope**.

6. Make sure the **Scope** is set to **Web**.

7. In the **Items in the Solution** list box, select **ContractorAgreementsListInstance** and **ContractorAgreementsList**, and then click the right arrow button (>) to move the items into the current Feature.

8. Scroll to the bottom of the design pane, expand **Feature Activation Dependencies**, and then click **Add**.

9. In the **Add Feature Activation Dependencies** dialog box, click **Contractor Management Resources**, and then click **Add**.

10. On the **Manifest** tab, review the Feature manifest file that Visual Studio has created for you.

11. Click **Save**, and then close the **ContractorAgreementsList.feature** tab.

► **Task 3: Test the Feature dependency**

1. On the **PROJECT** menu, click **ContractorAgreements Properties**.

2. On the list on the left of the page, click **SharePoint**.

3. In the **Active Deployment Configuration** drop-down menu, click **No Activation**.

4. Click **Save**, and then close the **ContractorAgreements** tab.

5. On the **BUILD** menu, click **Rebuild Solution**.

6. On the **DEBUG** menu, click **Start Without Debugging**.

7. If you are prompted for credentials, log in as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.

8. In Internet Explorer, after the page finishes loading, on the **Settings** menu, click **Site settings**.

9. On the **Site Settings** page, under **Site Actions**, click **Manage site features**.

10. On the **Site Features** page, in the **Contractor Agreements List** row, click **Activate**.

11. Verify that SharePoint displays an error message stating that you must activate the Contractor Management Resources feature before trying again.

12. On the **Settings** menu, click **Site settings**.

13. Under **Site Collection Administration**, click **Site collection features**.

14. On the **Site Collection Features** page, in the **Contractor Management Resources** row, click **Activate**.

15. On the **Settings** menu, click **Site settings**.

16. On the **Site Settings** page, under **Site Actions**, click **Manage site features**.
17. On the **Site Features** page, in the **Contractor Agreements List** row, click **Activate**.
18. On the Quick Launch navigation menu, under **Recent**, click **Contractor Agreements**.
19. On the **Contractor Agreements** page, click **new item**.
20. In the **Add a document** dialog box, browse to **E:\Labfiles\Starter**, click **SJAgreement.docx**, click **Open**, and then click **OK**.
21. In the **Contractor Agreements – SJAgreement.docx** dialog box, in the **Title** box, type **SJ Work for Hire Agreement**.
22. In the **Full Name** box, type **Sanjay Jacob**.
23. In the **Contoso Manager** box, type **Carol**, and then click **Carol Troup**.
24. In the **Contoso Team** drop-down list box, click **Research**.
25. In the **Daily Rate** box, type **800**.
26. In the **Agreement Start Date** box, type **10/1/2013**.
27. In the **Agreement End Date** box, type **10/1/2014**.
28. Select **Security Cleared**, and then click **Save**.
29. Close all open windows.

**Results:** After completing this exercise, you should have configured a Web-scoped Feature that includes a dependency on a Site-scoped Feature.





## Module 5: Working with Server-Side Code

# Lab: Working with Server-Side Code

### Exercise 1: Developing an Event Receiver

#### ► Task 1: Create a New Solution

1. Start the 20488B-LON-SP-05 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. In Visual Studio, on the **FILE** menu, point to **New**, and then click **Project**.
5. In the **New Project** dialog box, under **Templates**, expand **Visual C#**, expand **Office/SharePoint**, click **SharePoint Solutions**, and then click **SharePoint 2013 - Empty Project**.
6. In the **Name** box, type **ExpensesEventReceiver**, in the **Location** box, type **E:\Labfiles\Starter\**, and then click **OK**.
7. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://sales.contoso.com**, click **Deploy as a farm solution**, and then click **Finish**.

#### ► Task 2: Create an Event Receiver

1. In Solution Explorer, right-click **ExpensesEventReceiver**, point to **Add**, and then click **New Item**.
2. In the **Add New Item - ExpensesEventReceiver** dialog box, click **Event Receiver**, in the **Name** box, type **ContosoExpensesEventReceiver**, and then click **Add**.
3. In the **SharePoint Customization Wizard** dialog box, in the **What type of event receiver do you want** list, click **List Item Events**.
4. In the **What item should be the event source** list, click **Custom List**.
5. In the **Handle the following events** list, select the **An item is being added**, **An item is being updated**, and **An item is being deleted** check boxes, and then click **Finish**.
6. Add the following code on a new line after the closing brace of the **ItemDeleting** method.

```
private void UpdatePropertyBag(SPWeb web, double change)
{
 string keyName = "ContosoDepartmentalExpenseTotal";
 double currentValue = 0;
 if(web.Properties[keyName] != null)
 {
 currentValue = double.Parse(web.Properties[keyName]);
 }
 else
 {
 web.Properties.Add(keyName, "");
 }
 currentValue += change;
 web.Properties[keyName] = currentValue.ToString();
 web.Properties.Update();
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex1.txt file.

7. Modify the **ItemAdding** method to resemble the following code example.

```
public override void ItemAdding(SPItemEventProperties properties)
{
 double invoiceValue = double.Parse(properties.AfterProperties["InvoiceTotal"] as string);
 UpdatePropertyBag(properties.Web, invoiceValue);
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex1.txt file.

8. Modify the **ItemUpdating** method to resemble the following code example.

```
public override void ItemUpdating(SPItemEventProperties properties)
{
 double previousInvoiceValue = (double)properties.ListItem["InvoiceTotal"];
 double newInvoiceValue = double.Parse(properties.AfterProperties["InvoiceTotal"] as string);
 double change = newInvoiceValue - previousInvoiceValue;
 UpdatePropertyBag(properties.Web, change);
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex1.txt file.

9. Modify the **ItemDeleting** method to resemble the following code example.

```
public override void ItemDeleting(SPItemEventProperties properties)
{
 double invoiceValue = double.Parse(properties.BeforeProperties["InvoiceTotal"] as string);
 UpdatePropertyBag(properties.Web, -invoiceValue);
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex1.txt file.

10. On the **BUILD** menu, click **Build Solution**. Correct any errors.

### ► Task 3: Create a Feature and Deploy the Event Receiver

1. In Solution Explorer, double-click **Elements.xml**.
2. Modify the **Receivers** element to resemble the following code sample.

```
<Receivers ListUrl="Lists/Contoso%20Expenses">
```

3. Click **Save All**.
4. On the BUILD menu, click **Deploy ExpensesEventReceiver**.
5. Close Visual Studio.

**Results:** After completing this exercise, you should have developed and deployed an event receiver.

## Exercise 2: Updating a Web Part

### ► Task 1: Modify a Web Part to Retrieve a Value from the Site Property Bag

1. On the taskbar, click **File Explorer**.
2. In File Explorer, browse to **E:\Labfiles\Starter\ExpensesWebPart**, and then double-click **ExpensesWebPart.sln**.
3. In the **How do you want to open this type of file (.sln)?** dialog box, click **Visual Studio 2012**.
4. In Visual Studio, in Solution Explorer, double-click **ExpensesInformationWebPart**.
5. In the **RenderContents** method, add the following code shown in bold.

```
writer.Write("Please record your expenses in the departmental expenses list.");
SPWeb web = SPContext.Current.Web;
string keyName = "ContosoDepartmentalExpenseTotal";
if(web.Properties[keyName] != null)
{
 string expenseTotal = web.Properties[keyName];
 writer.Write("<p>Estimated departmental expense total is: " + expenseTotal + "</p>");
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex2.txt file.

6. On the **BUILD** menu, click **Build Solution**. Correct any errors.

### ► Task 2: Deploy the Web Part to the sales.contoso.com Site

1. On the **BUILD** menu, click **Deploy ExpensesWebPart**.
2. On the Start screen, click **Internet Explorer**.
3. In Internet Explorer, browse to **http://sales.contoso.com**.
4. On the **Contoso Team Site** site, on the ribbon, on the **PAGE** tab, click **Edit**.
5. On the **INSERT** tab, click **Web Part**.
6. In the **Categories** list, click **Custom**.
7. In the **Parts** list, click **ExpensesWebPart - ExpensesInformationWebPart**, and then click **Add**.
8. Click **SAVE**.

### ► Task 3: Test the Web Part and Event Receiver by Adding Items to the Contoso Expenses List

1. Verify that the **ExpensesWebPart - ExpensesInformationWebPart** Web Part does not currently display a total.
2. On the Quick Launch menu, click **Contoso Expenses**.
3. On the **Contoso Expenses** page, click **new item**.
4. On the **New Item** page, in the **Title** box, type **Professional Services**.
5. In the **Invoice Number** box, type **AA678**.
6. In the **Supplier Name** box, type **Fabrikam**.
7. In the **Invoice Total** box, type **150**.

8. In the **Invoice Date** box, type today's date, or use the date picker to select today's date, and then click **Save**.
9. On the Quick Launch menu, click **Home**.
10. Verify that the **ExpensesWebPart - ExpensesInformationWebPart** Web Part now displays an estimated total for departmental expenses.
11. Close Internet Explorer.
12. Close Visual Studio.

**Results:** After completing this exercise, you should have modified and deployed a Web Part, and tested the Web Part and event receiver.

### Exercise 3: Creating a Timer Job

#### ► Task 1: Create a New Solution

1. On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
2. In Visual Studio, on the **FILE** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, under **Templates**, expand **Visual C#**, click **Office/SharePoint**, and then click **SharePoint 2013 - Empty Project**.
4. In the **Name** box, type **ExpensesTimerJob**, in the **Location** box, type **E:\Labfiles\Starter\**, and then click **OK**.
5. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://managers.contoso.com**, click **Deploy as a farm solution**, and then click **Finish**.

#### ► Task 2: Create a Timer Job Definition

1. In Solution Explorer, right-click **ExpensesTimerJob**, point to **Add**, and then click **New Item**.
2. In the **Add New Item - ExpensesTimerJob** dialog box, click **Code**, and then click **Class**.
3. In the **Name** box, type **ContosoExpensesOverviewTimerJob**, and then click **Add**.
4. Add the following **using** statements to the list of **using** statements at the start of the file.

```
using Microsoft.SharePoint.Administration;
using Microsoft.SharePoint;
```

5. Modify the class declaration as shown in bold in the following code.

```
public class ContosoExpensesOverviewTimerJob : SPJobDefinition
```

6. Add the following code to the **ContosoExpensesOverviewTimerJob** class.

```
public ContosoExpensesOverviewTimerJob()
{
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

7. Add the following code to the **ContosoExpensesOverviewTimerJob** class.

```
public ContosoExpensesOverviewTimerJob(string name, SPWebApplication webApplication, SPSServer
server, SPJobLockType lockType)
 : base(name, webApplication, server, lockType)
{
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

8. Add the following code to the **ContosoExpensesOverviewTimerJob** class.

```
public override void Execute(Guid targetInstanceId)
{
 // Obtain a reference to the managers site collection.
 using (SPSite managerSite = new SPSite("http://managers.contoso.com"))
 {
 // Obtain a reference to the managers site.
 using (SPWeb managerWeb = managerSite.RootWeb)
 {
 // Obtain a reference to the expense overview list.
 SPList overviewList = managerWeb.Lists["Expenses Overview"];
 // Remove all existing items from the list.
 while (overviewList.Items.Count > 0)
 {
 overviewList.Items[0].Delete();
 overviewList.Update();
 }
 // Iterate through each site collection in the current web application.
 foreach (SPSite departmentSite in this.WebApplication.Sites)
 {
 using (SPWeb departmentWeb = departmentSite.RootWeb)
 {
 // Get the Contoso Expenses list, if one exists.
 SPList expensesList = departmentWeb.Lists.TryGetList("Contoso Expenses");
 if (expensesList != null)
 {
 // Calculate the total for the department.
 double departmentTotal = 0;
 foreach (SPListItem expense in departmentWeb.Lists["Contoso
Expenses"].Items)
 {
 departmentTotal += (double)expense["InvoiceTotal"];
 }
 // Use the site URL to determine the department name.
 Uri url = new Uri(departmentWeb.Url);
 string hostName = url.GetComponents(UriComponents.Host,
UriFormat.Unescaped);
 string[] hostNameComponents = hostName.Split('.');
 // Create a new item in the expense overview list.
 SPListItem overviewItem = overviewList.Items.Add();
 overviewItem["Title"] = hostNameComponents[0];
 overviewItem["Expense Total"] = departmentTotal;
 overviewItem.Update();
 overviewList.Update();
 }
 }
 departmentSite.Dispose();
 }
 }
 }
}
```



**Note:** The code for this step is available in E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

- On the **BUILD** menu, click **Build Solution**. Correct any errors.

### ► Task 3: Deploy the Timer Job

- In Solution Explorer, right-click **Features**, and then click **Add Feature**.
- In the Feature designer, in the **Title** box, type **Contoso Expenses Overview Timer Job Installer**.
- In the **Scope** list, click **Site**.
- In Solution Explorer, right-click **Feature1**, and then click **Rename**.
- Type **ContosoExpensesOverviewTimerJobInstaller**, and then press Enter.
- Right-click **ContosoExpensesOverviewTimerJobInstaller**, and then click **Add Event Receiver**.
- Add the following **using** statement to the list of **using** statements at the start of the file.

```
using Microsoft.SharePoint.Administration;
```

- Add the following code to the **ContosoExpensesOverviewTimerJobInstallerEventReceiver** class.

```
const string timerJobName = "ExpensesOverviewJob";
```

- Add the following code to the **ContosoExpensesOverviewTimerJobInstallerEventReceiver** class.

```
private void deleteJob(SPWebApplication webApplication)
{
 foreach(SPJobDefinition job in webApplication.JobDefinitions)
 {
 if(job.Name.Equals(timerJobName))
 {
 job.Delete();
 }
 }
}
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

- Select the **FeatureActivated** method, press CTRL+K, and then CTRL+U.
- Add the following code to the **FeatureActivated** method.

```
SPWebApplication webApplication = ((SPSite)properties.Feature.Parent).WebApplication;
deleteJob(webApplication);
ContosoExpensesOverviewTimerJob timerJob = new
ContosoExpensesOverviewTimerJob(timerJobName,webApplication, null, SPJobLockType.Job);
SPMinuteSchedule schedule = new SPMinuteSchedule();
schedule.BeginSecond = 1;
schedule.EndSecond = 5;
schedule.Interval = 2;
timerJob.Schedule = schedule;
timerJob.Update();
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

12. Select the **FeatureDeactivating** method, press CTRL+K, and then CTRL+U.
13. Add the following code to the **FeatureDeactivating** method.

```
SPWebApplication webApplication = ((SPSite)properties.Feature.Parent).WebApplication;
deleteJob(webApplication);
```



**Note:** The code for this step is available in the E:\Labfiles\Starter\LabCodeSnippets-Ex3.txt file.

14. On the **BUILD** menu, click **Deploy ExpensesTimerJob**.

► **Task 4: Test the Timer Job by Viewing the Expenses Overview List on the managers.contoso.com Site**

1. On the Start screen, click **Internet Explorer**.
2. In Internet Explorer, browse to **http://managers.contoso.com**.
3. On the **Contoso Team Site** site, click **Site Contents**.
4. On the **Site Contents** page, click **Expenses Overview**.
5. Verify that the **Expenses Overview** list contains a list item for each department. You may need to wait up to two minutes for the timer job to run and populate the list.
6. Close Internet Explorer.
7. Close Visual Studio.

**Results:** After completing this exercise, you should have developed, deployed, and tested a timer job.





## Module 6: Managing Identity and Permissions

# Lab A: Managing Permissions Programmatically in SharePoint 2013

### Exercise 1: Managing List Permissions Programmatically

#### ► Task 1: Add a Feature Receiver to the Solution

1. Start the 20488B-LON-SP-06 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
3. On the Start screen, click **Computer**.
4. In File Explorer, browse to **E:\Labfiles\LabA\Starter\FinancialsLibrary**.
5. Double-click **FinancialsLibrary.sln**.
6. If you are asked **How do you want to open this type of file (.sln)?** click **Visual Studio 2012**.
7. In Solution Explorer, expand **FinancialsLibrary**.
8. Double-click the **Financials** list.
9. In Solution Explorer, expand **Features**, and then double-click **FinancialsLibrary**.
10. Right-click **FinancialsLibrary**, and then click **Add Event Receiver**.

#### ► Task 2: Add Code that Breaks Permissions Inheritance

1. In the **FinancialsLibrary.EventReceiver.cs** code window, locate the following code:

```
//public override void FeatureActivated (SPFeatureReceiverProperties properties)
//{
//}
```

2. Replace the located code with the following code:

```
public override void FeatureActivated (SPFeatureReceiverProperties properties)
{
}
```

3. Place the cursor within the **FeatureActivated** method you just uncommented.
4. Type the following code:

```
SPWeb parentWeb = (SPWeb)properties.Feature.Parent;
```

5. Press Enter.
6. Type the following code:

```
SPDocumentLibrary financialsLibrary = (SPDocumentLibrary)parentWeb.Lists["Financials"];
```

7. Press Enter.

- Type the following code:

```
if (financialsLibrary != null)
{
}
```

- Place the cursor within the **if** statement you just created.
- Type the following code:

```
financialsLibrary.BreakRoleInheritance(false);
```

- Press Enter.
- Type the following code:

```
financialsLibrary.Update();
```

### ► Task 3: Grant Permissions to Site Owners

- Press Enter.
- Type the following code:

```
SPRoleAssignment ownersAssignment = new SPSRoleAssignment(parentWeb.AssociatedOwnerGroup);
```

- Press Enter.
- Type the following code:

```
ownersAssignment.RoleDefinitionBindings.Add(parentWeb.RoleDefinitions["Full Control"]);
```

- Press Enter.
- Type the following code:

```
financialsLibrary.RoleAssignments.Add(ownersAssignment);
```

- Press Enter.
- Type the following code:

```
financialsLibrary.Update();
```

### ► Task 4: Grant Permissions to Managers

- Press Enter.
- Type the following code:

```
parentWeb.AllUsers.Add("CONTOSO\Managers", "", "", "Managers AD DS Group");
```

- Press Enter.
- Type the following code:

```
SPRoleAssignment managersAssignment = new
SPRoleAssignment(parentWeb.AllUsers["CONTOSO\Managers"]);
```

- Press Enter.
- Type the following code:

```
managersAssignment.RoleDefinitionBindings.Add(parentWeb.RoleDefinitions["Contribute"]);
```

7. Press Enter.
8. Type the following code:

```
financialsLibrary.RoleAssignments.Add(managersAssignment);
```

9. Press Enter.
10. Type the following code:

```
financialsLibrary.Update();
```

### ► Task 5: Test the Financials Library Project

1. On the **DEBUG** menu, click **Start Without Debugging**.
2. In the **Windows Security** dialog box, in the **Username** box, type **Administrator**.
3. In the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
4. On the **Contoso Development Site** home page, in the Quick Launch menu, click **Site Contents**.
5. Under **Site Contents**, click the **Financials** list.
6. If a **Webpage Error** dialog box appears, click **No**.
7. On the ribbon, click the **Library** tab, and then click **Library Settings**.
8. Under **Permissions and Management**, click **Permissions for this document library**.
9. Note that the library does not inherit permissions from the parent site.
10. Note that the **Contoso Development Site Owners** group has **Full Control** permissions.
11. Note that the **Managers** group has **Contribute** permissions.
12. Close Internet Explorer.
13. Close Visual Studio.

**Results:** When you have completed this exercise, you will have built a complete SharePoint solution that includes a feature receiver. The feature receiver code will set permissions level on the Financials library.

# Lab B: Creating and Deploying a Custom Claims Provider

## Exercise 1: Creating a Custom Claims Provider

### ► Task 1: Create a New SharePoint Project in Visual Studio

1. Start the 20488B-LON-SP-06 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with password **Pa\$\$w0rd**.
3. On the Start screen, click **Visual Studio 2012**.
4. On the **File** menu, point to **New**, and then click **Project**.
5. In the hierarchy on the left, browse to **Templates|Visual C#|Office/SharePoint|SharePoint Solutions**.
6. In the list of templates, click **SharePoint 2013 - Empty Project**.
7. In the Name box, type **ContosoClaimsProvider**.
8. In the Location box, type **E:\Labfiles\LabB\Starter**, and then click **OK**.
9. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging?** box, type **http://dev.contoso.com**.
10. Click **Deploy as a farm solution**, and then click **Finish**.
11. In Solution Explorer, right-click **References**, and then click **Add Reference**.
12. In the hierarchy on the left, click **Extensions**.
13. In the list of extensions, select the **Microsoft.IdentityModel** check box, select the **Microsoft.SharePoint.Security** check box, and then click **OK**.

### ► Task 2: Add a Claims Provider Class to the Project

1. In Solution Explorer, right-click **ContosoClaimsProvider**, point to **Add**, and then click **Existing Item**.
2. In the **Add Existing Item - ContosoClaimsProvider** dialog box, browse to **E:\LabFiles\LabB\C# Code**.
3. Click **ContosoClaimsProvider.cs** and then click **Add**.
4. In Solution Explorer, double-click **ContosoClaimsProvider.cs**.
5. Locate the following line of code:

```
class LocationClaimsProvider
```

6. Replace the located code with the following code:

```
class LocationClaimsProvider : SPClaimProvider
```

7. On the **FILE** menu, click **Save All**.

### ► Task 3: Create Internal Properties for the Claims Provider

1. Locate the following line of code:

```
//Add internal and private properties here
```

2. Replace the located code with the following code:

```
internal static string ProviderDisplayName
{
 get { return "Location Claims Provider"; }
}
```

3. Press Enter.
4. Type the following code:

```
internal static string ProviderInternalName
{
 get { return "LocationClaimsProvider"; }
}
```

5. Press Enter.
6. Type the following code:

```
private static string LocationClaimType
{
 get { return "http://schema.contoso.com/location"; }
}
```

7. Press Enter.
8. Type the following code:

```
private static string LocationClaimValueType
{
 get { return Microsoft.IdentityModel.Claims.ClaimValueTypes.String; }
}
```

9. On the **FILE** menu, click **Save All**.

#### ► Task 4: Implement Claims Augmentation

1. In the **ContosoClaimsProvider.cs** code file, locate the **FillClaimsForEntity** method.



**Note:** The full code for this task is available in E:\Labfiles\LabB\Starter\LabCodeSnippets-Ex1.txt file.

2. Within the **FillClaimsForEntity** method, locate the following code:

```
throw new NotImplementedException();
```

3. Replace the located code with the following code:

```
if (entity == null)
{
 throw new ArgumentNullException("entity");
}
```

4. Press Enter.
5. Type the following code:

```
if (claims == null)
{
 throw new ArgumentNullException("claims");
}
```

6. Press Enter.

7. Type the following code:

```
string currentLocation = getLocation(entity);
```

8. Press Enter.

9. Type the following code:

```
SPClaim newClaim = CreateClaim(LocationClaimType, currentLocation, LocationClaimValueType);
```

10. Press Enter.

11. Type the following code:

```
claims.Add(newClaim);
```

12. Locate the following lines of code:

```
public override bool SupportsEntityInformation
{
 get { return false; }
}
```

13. Replace the located code with the following code:

```
public override bool SupportsEntityInformation
{
 get { return true; }
}
```

14. On the **FILE** menu, click **Save All**.

**Results:** A claims provider that can add claims to the user's security token based on their location.

## Exercise 2: Supporting Search and Resolve in a Claims Provider

### ► Task 1: Implement Search Functionality

1. In the **ContosoClaimsProvider.cs** code file, locate the **getPickerEntity** method.



**Note:** The full code for this task is available in E:\Labfiles\LabB\Starter\LabCodeSnippets-Ex2.txt file.

2. Within the **getPickerEntity** method, locate the following code:

```
throw new NotImplementedException();
```

3. Replace the located code with the following code:

```
PickerEntity newEntity = CreatePickerEntity();
```

4. Press Enter.

5. Type the following code:

```
newEntity.Claim = CreateClaim(LocationClaimType, ClaimValue, LocationClaimValueType);
```

6. Press Enter.

7. Type the following code:

```
newEntity.Description = ProviderDisplayName + ":" + ClaimValue;
```

8. Press Enter.

9. Type the following code:

```
newEntity.DisplayText = ClaimValue;
```

10. Press Enter.

11. Type the following code:

```
newEntity.EntityData[PeopleEditorEntityDataKeys.DisplayName] = ClaimValue;
```

12. Press Enter.

13. Type the following code:

```
newEntity.EntityType = SPClaimEntityTypes.FormsRole;
```

14. Press Enter.

15. Type the following code:

```
newEntity.IsResolved = true;
```

16. Press Enter.

17. Type the following code:

```
newEntity.EntityGroupName = "Location";
```

18. Press Enter.

19. Type the following code:

```
return newEntity;
```

20. Locate the **FillSearch** method.

21. Within the **FillSearch** method, locate the following code:

```
throw new NotImplementedException();
```

22. Replace the located code with the following code:

```
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
{
 return;
}
```

23. Press Enter.

24. Type the following code:

```
int locationNode = -1;
```

25. Press Enter.

26. Type the following code:

```
SPPProviderHierarchyNode matchesNode = null;
```

27. Press Enter.

28. Type the following code:

```
foreach (string location in possibleLocations)
{
}
```

29. Place the cursor within the **foreach** loop you just created.

30. Type the following code:

```
locationNode ++;
```

31. Press Enter.

32. Type the following code:

```
if (location.ToLower().StartsWith(searchPattern.ToLower()))
{
}
```

33. Place the cursor within the **if** statement you just created.

34. Type the following code:

```
PickerEntity newEntity = getPickerEntity(location);
```

35. Press Enter.

36. Type the following code:

```
if (!searchTree.HasChild(locationKeys[locationNode]))
{
}
else
{
}
```

37. Place the cursor within the **if** code block.

38. Type the following code:

```
matchesNode = new SPPProviderHierarchyNode(
 ProviderInternalName,
 locationLabels[locationNode],
 locationKeys[locationNode],
 true);
```

39. Press Enter.



40. Type the following code:

```
searchTree.AddChild(matchesNode);
```

41. Place the cursor within the **else** code block.

42. Type the following code:

```
matchesNode = searchTree.Children.Where(
 theNode => theNode.HierarchyNodeID == locationKeys[locationNode]).First();
```

43. Place the cursor after the end of the **else** code block.

44. Type the following code:

```
matchesNode.AddEntity(newEntity);
```

45. Locate the following code:

```
public override bool SupportsSearch
{
 get { return false; }
}
```

46. Replace the located code with the following code:

```
public override bool SupportsSearch
{
 get { return true; }
}
```

47. On the **FILE** menu, click **Save All**.

## ► Task 2: Implement Resolve Functionality

1. In the **ContosoClaimsProvider.cs** code file, locate the **FillResolve** method which currently throws a **NotImplementedException** exception.



**Note:** The full code for this task is available in E:\Labfiles\LabB\Starter\LabCodeSnippets-Ex2.txt file.

2. Within the **FillResolve** method, locate the following code:

```
throw new NotImplementedException();
```

3. Replace the located code with the following code:

```
if (!EntityTypesContain(entityTypes, SPClaimEntityTypes.FormsRole))
{
 return;
}
```

4. Press Enter.

5. Type the following code:

```
foreach (string location in possibleLocations)
{
}
```

6. Place the cursor within the **foreach** code block you just created.

7. Type the following code:

```
if (location.ToLower() == resolveInput.ToLower())
{
}
```

8. Place the cursor within the **if** code block you just created.
9. Type the following code:

```
PickerEntity newEntity = getPickerEntity(location);
```

10. Press Enter.
11. Type the following code:

```
resolved.Add(newEntity);
```

12. Locate the following code:

```
public override bool SupportsResolve
{
 get { return false; }
}
```

13. Replace the located code with the following code:

```
public override bool SupportsResolve
{
 get { return true; }
}
```

14. On the **FILE** menu, click **Save All**.

**Results:** A claims provider that can respond to user searches in the People Picker control.

### Exercise 3: Deploying and Testing a Claims Provider

#### ► Task 1: Create a Feature for the Solution

1. In Solution Explorer, right-click **Features**, and then click **Add Feature**.
2. In Solution Explorer, right-click **Feature1**, and then click **Rename**.
3. Type **ContosoClaimProviderFeature**, and then press Enter.
4. In the **ContosoClaimProviderFeature.feature** designer, in the **Title** box, type **Contoso Location Claim Provider**.
5. In the **Scope** drop-down list, click **Farm**.
6. On the **FILE** menu, click **Save All**.

#### ► Task 2: Create a Feature Receiver to Deploy the Claims Provider

1. In Solution Explorer, right click **ContosoClaimProviderFeature**, and then click **Add Event Receiver**.
2. In **ContosoClaimProviderFeature.EventReceiver.cs**, locate the following line of code:

```
using Microsoft.SharePoint;
```

3. Immediately after the located code, insert the following code:

```
using Microsoft.SharePoint.Administration.Claims;
```

4. Locate the following line of code:

```
[Guid("...")]
```

5. Immediately before the located line, insert the following line of code:

```
[Microsoft.SharePoint.Security.SharePointPermission(System.Security.Permissions.SecurityAction.Demand, ObjectModel = true)]
```

6. Locate the following line of code:

```
public class ContosoClaimProviderFeatureEventReceiver : SPFeatureReceiver
```

7. Replace the located code with the following code:

```
public class ContosoClaimProviderFeatureEventReceiver : SPClaimProviderFeatureReceiver
```

8. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class.

9. Type **override**, and then double click **ClaimProviderAssembly{get; }**.

10. Locate the following code:

```
get { throw new NotImplementedException(); }
```

11. Replace the located code with the following code:

```
get { return typeof(Contoso.Security.LocationClaimsProvider).Assembly.FullName; }
```

12. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class but outside any methods.

13. Type **override**, and then double-click **ClaimProviderDescription{get; }**.

14. Locate the following code:

```
get { throw new NotImplementedException(); }
```

15. Replace the located code with the following code:

```
get { return "A claim provider that certifies the user's location"; }
```

16. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class but outside any methods.

17. Type **override**, and then double-click **ClaimProviderDisplayName{get; }**.

18. Locate the following code:

```
get { throw new NotImplementedException(); }
```

19. Replace the located code with the following code:

```
get { return Contoso.Security.LocationClaimsProvider.ProviderDisplayName; }
```

20. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class but outside any methods.

21. Type **override**, and then double-click **ClaimProviderType{get; }**.

22. Locate the following code:

```
get { throw new NotImplementedException(); }
```

23. Replace the located code with the following code:

```
get { return typeof(Contoso.Security.LocationClaimsProvider).FullName; }
```

24. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class but outside any methods.

25. Type the following code:

```
private void ExecBaseFeatureActivated(SPFeatureReceiverProperties properties)
{
}
```

26. Place the cursor within the **ExecBaseFeatureActivated** method.

27. Type the following code:

```
base.FeatureActivated(properties);
```

28. Place the cursor within the **ContosoClaimProviderFeatureEventReceiver** class but outside any methods.

29. Type the following code:

```
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
}
```

30. Place the cursor within the **FeatureActivated** method.

31. Type the following code:

```
ExecBaseFeatureActivated(properties);
```

32. On the **FILE** menu, click **Save All**.

### ► Task 3: Test the Solution

1. On the **DEBUG** menu, click **Start Without Debugging**.
2. In the **Windows Security** dialog box, in the **Username** box, type **Administrator**.
3. In the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
4. In the top-right, click the **Settings** icon, and then click **Site settings**.
5. Under **Users and Permissions**, click **Site permissions**.
6. Click **Contoso Development Site Visitors**.
7. Click **New**.
8. Under **Add people to the Contoso Development Site Visitors group**, type **Europe**.
9. In the results, click **Europe**, and then click **Share**.
10. Close Internet Explorer.
11. On the **DEBUG** menu, click **Start Debugging**.
12. If a **Debugging Not Enabled** dialog box appears, click **OK**.
13. Before you log on to SharePoint, switch back to Visual Studio.
14. On the **DEBUG** menu, click **Attach to Process**.

15. Select the **Show processes from all users** checkbox.
16. Scroll to the bottom of the **Available processes** list.
17. Select the **w3wp.exe** process with the username **CONTOSO\SPFarm**.
18. Click **Attach** and then click **Attach** in the dialog.
19. In the **ContosoClaimsProvider.cs** code file, in the **FillClaimsForEntity** method, locate the following line of code:

```
claims.Add(newClaim);
```

20. Right-click the located code, click **Breakpoint**, and then click **Insert Breakpoint**.
21. Switch back to Internet Explorer.
22. In the **Windows Security** dialog, in the **User name** box, type **CONTOSO\Alexei**
23. In the **Password** box, type **Pa\$\$w0rd** and then click **OK**. Visual Studio interrupts execution at the breakpoint you just created.
24. In the **Immediate Window**, type **currentLocation** and then press Enter. Visual Studio shows that the **currentLocation** variable contains the string **Europe**.
25. On the **DEBUG** menu, click **Continue**. SharePoint displays the site, because Alexei Eremenko is located in Europe.
26. Close Internet Explorer.
27. Close Visual Studio.

**Results:** A completed SharePoint claims provider project.



# Module 7: Introducing Apps for SharePoint

## Lab: Creating a Site Suggestions App

### Exercise 1: Creating a New SharePoint App

#### ► Task 1: Create a New SharePoint Hosted App

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20488B-LON-SP-07**, and then click **Connect**.
2. To log on to **20488B-LON-SP-07**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **CONTOSO\Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then click the **Forward** button.
4. On the Start screen, click **Visual Studio 2012**.
5. On the **File** menu, point to **New**, and then click **Project**.
6. In the hierarchy on the left, browse to **Templates/Visual C#/Office/SharePoint/Apps**.
7. In the list of templates, click **App for SharePoint 2013**.
8. In the **Name** box, type **SiteSuggestionsApp**.
9. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
10. In the **New App for SharePoint** dialog box, in the **What is the name of your app for SharePoint** box, type **Site Suggestions**.
11. In the **What SharePoint site do you want to use for debugging your app** box, type **http://dev.contoso.com**.
12. In the **How do you want to host your app for SharePoint** list, click **SharePoint-hosted**, and then click **Finish**.



**Note:** Visual Studio creates the new project and displays the default.aspx code window.

#### ► Task 2: Add Site Columns to the App

1. In Solution Explorer, right-click **SiteSuggestionsApp**, point to **Add**, and then click **New Item**.
2. In the list of templates, click **Site Column**.
3. In the **Name** text box, type **SuggestionSubject**, and then click **Add**.
4. In Solution Explorer, right-click **SiteSuggestionsApp**, point to **Add**, and then click **New Item**.
5. In the list of templates, click **Site Column**.
6. In the **Name** box, type **Feedback**, and then click **Add**.
7. In Solution Explorer, double-click the **Feedback** site column.
8. In the **Elements.xml** code window, locate the following line of code:

```
type="Text"
```

9. Replace the located code with the following code:

```
type="Note"
```

10. On the **File** menu, click **Save All**.

► **Task 3: Create a Content Type for Suggestions**

1. In Solution Explorer, right-click **SiteSuggestionsApp**, point to **Add**, and then click **New Item**.
2. In the list of templates, click **Content Type**.
3. In the **Name** box, type **Suggestion**, and then click **Add**.
4. Ensure that **Item** is selected in the drop-down list, and then click **Finish**.
5. In the Suggestion designer, in the Display Name column, click **Click here to add a column**.
6. Type **Suggestion Subject**, and then click the **Suggestion Subject** column.
7. Type **Feedback**, and then click the **Feedback** column.
8. On the **File** menu, click **Save All**.

► **Task 4: Create the Suggestions List**

1. In Solution Explorer, right-click **SiteSuggestionsApp**, point to **Add**, and then click **New Item**.
2. In the list of templates, click **List**.
3. In the **Name** box, type **Suggestions**, and then click **Add**.
4. In the SharePoint Customization Wizard, click **Finish**.
5. In the Suggestions list designer, on the **Columns** tab, scroll to the bottom, and then click **Content Types**.
6. In the **Content Type Settings** dialog box, click **Click here to add a content type**.
7. Type **Suggestion**, and then click the **Suggestion** content type.
8. Right-click **Item**, and then click **Delete**.
9. Right-click **Folder**, click **Delete**, and then click **OK**.
10. In the Suggestions list designer, click the **List** tab.
11. In the **Description** box, type **Use this list to store site suggestions**.
12. On the **File** menu, click **Save All**.

**Results:** A simple SharePoint hosted app with custom site columns, a content type and a list instance.



## Exercise 2: Using the Client-Side Object Model

### ► Task 1: Add the Contoso Framework

1. In Solution Explorer, expand the **Scripts** folder.
2. Right-click **App.js**, click **Delete**, and then click **OK**.
3. Right-click the **Scripts** folder, click **Add**, and then click **Existing Item**.
4. Browse to **E:\Labfiles\JavaScript Code**.
5. Click **App.js**, and then click **Add**.

### ► Task 2: Build New Suggestion Functionality

1. On the Start screen, click **Computer**.
2. Browse to **E:\Labfiles\Web Page Markup**.
3. Double-click **CreateSuggestionUI.txt**.
4. From the **How do you want to open this type of file (.txt)?** dialog box, click **Notepad**.
5. In Notepad, on the **Edit** menu, click **Select All**.
6. On the **Edit** menu, click **Copy**.
7. Switch to Visual Studio.
8. In Solution Explorer, expand the **Pages** folder, and then double-click **default.aspx**.
9. Locate the last **</asp:Content>** tag in the default.aspx page.
10. Place the cursor immediately above the located tag.
11. On the **Edit** menu, click **Paste**.
12. Expand the **Scripts** folder, and then double-click **App.js**.
13. Locate the following line of code:

```
var user = context.get_web().get_currentUser();
```

14. Immediately after the located code, insert the following code:

```
var suggestionsList;
var allSuggestions;
var currentSuggestion;
```

15. Locate the following line of code

```
// Add code to add a new suggestion here
```

16. Replace the located code with the following code:

```
var itemCreateInfo = SP.ListItemCreationInformation();
```

17. Press Enter.

18. Type the following code:

```
currentSuggestion = suggestionsList.addItem(itemCreateInfo);
```

19. Press Enter.

20. Type the following code:

```
currentSuggestion.set_item('SuggestionSubject', $('#subject-input').val().toString());
```

21. Press Enter.

22. Type the following code:

```
currentSuggestion.set_item('Feedback', $('#feedback-input').val().toString());
```

23. Press Enter.

24. Type the following code:

```
currentSuggestion.update();
```

25. Press Enter.

26. Type the following code:

```
context.load(currentSuggestion);
```

27. On the **File** menu, click **Save All**.

### ► Task 3: Build the Display of Existing Suggestions

1. In Solution Explorer, double-click **Default.aspx**.
2. Locate the last `</asp:Content>` tag in the default.aspx page.
3. Place the cursor immediately above the located tag and then type the following code:

```
<p>Here are the current suggestions:</p>
```

4. Press Enter.

5. Type the following code:

```
<div id="suggestions-list"></div>
```

6. In Solution Explorer, double-click **App.js**.

7. In the App.js code file, locate the following line of code:

```
// Add code to query for suggestions here
```

8. Replace the located code with the following code:

```
suggestionsList = context.get_web().get_lists().getByTitle('Suggestions');
```

9. Press Enter.

10. Type the following code:

```
allSuggestions = suggestionsList.getItems(new SP.CamlQuery());
```

11. Press Enter.

12. Type the following code:

```
context.load(allSuggestions);
```

13. Locate the following line of code:

```
$(document).ready(function () {
```

14. Immediately after the located code, insert the following line of code:

```
Contoso.SuggestionsApp.get_suggestions();
```

15. On the **File** menu, click **Save All**.

#### ► **Task 4: Build the Suggestion Details Display**

1. Switch to File Explorer.
2. Browse to **E:\Labfiles\Web Page Markup**.
3. Double-click **DisplaySuggestionUI.txt**.
4. In Notepad, on the **Edit** menu, click **Select All**.
5. On the **Edit** menu, click **Copy**.
6. Switch to Visual Studio.
7. In Solution Explorer, expand the **Pages** folder, and then double-click **default.aspx**.
8. Locate the last **</asp:Content>** tag in the default.aspx page.
9. Place the cursor immediately above the located tag.
10. Click **Edit**, and then click **Paste**.
11. In Solution Explorer, double-click **App.js**.
12. In the **App.js** code window, locate the following line of code:

```
// Add code to query for a single suggestion here
```

13. Replace the located line of code with the following code:

```
currentSuggestion = suggestionsList.getItemById(suggestionId);
```

14. Press Enter.
15. Type the following code:

```
context.load(currentSuggestion);
```

16. Switch to File Explorer.
17. Browse to **E:\Labfiles\Style Code**.
18. Double-click **App Styles.txt**.
19. In Notepad, on the **Edit** menu, click **Select All**.
20. On the **Edit** menu, click **Copy**
21. Switch to Visual Studio.
22. In Solution Explorer, expand the **Content** folder, and then double-click **App.css**.
23. In the **App.css** code window, click **Edit**, and then click **Paste**.
24. Click **File**, and then click **Save All**.

► **Task 5: Run the App**

1. Click **Debug**, and then click **Start Debugging**.
2. If the **Windows Security** dialog box appears, in the **User name** text box, type **CONTOSO\Administrator**.
3. In the **Password** text box, type **Pa\$\$w0rd**, and then click **OK**.
4. In the **Suggestion Subject** text box, type **Test Suggestion**.
5. In the **Feedback** text box, type **This is to test the Site Suggestions app**, and then click **Submit Query**.
6. Under **Here are the current suggestions**, click **Test Suggestion**.



**Note:** The test suggestion item is displayed with the **Suggestion Subject** and **Feedback** fields.

7. Close all open windows.

**Results:** A simple SharePoint app in which users can create new suggestions with feedback for site improvements. The app will also enable users to browse suggestions from other users.

# Module 8: Client-Side SharePoint Development

## Lab A: Using the Client-Side Object Model for Managed Code

### Exercise 1: Create the Mileage Claim List

#### ► Task 1: Add Site Columns

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20488B-LON-SP-08**, and then click **Connect**.
2. To log on to **20488B-LON-SP-08**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **CONTOSO\Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then press Enter.
4. On the Start screen, click **Computer**.
5. In File Explorer, browse to **E:\Labfiles\LabA\Starter\MileageRecorder**.
6. Double-click **MileageRecorder.sln**.
7. If the **How do you want to open this type of file (.sln)?** dialog box appears, click **Visual Studio 2012**.
8. In Solution Explorer, in the **MileageRecorder SharePoint** project, right-click **SiteColumns**, click **Add**, and then click **New Item**.
9. In the **Add New Item** dialog box, select the **Site Column** template.
10. In the **Name** box, type **Destination**, and then click **Add**.
11. In the **Elements.xml** code file, locate the following code:

```
Required="FALSE"
```

12. Replace the located code with the following code:
13. Required="TRUE"
14. In Solution Explorer, right-click **SiteColumns**, click **Add**, and then click **New Item**.
15. In the **Add New Item** dialog box, select the **Site Column** template.
16. In the **Name** box, type **ReasonForTrip**, and then click **Add**.
17. In the **Elements.xml** code file, locate the following code:

```
Type="Text"
Required="FALSE"
```

18. Replace the located code with the following code

```
Type="Note"
```

19. Required="TRUE"
20. In Solution Explorer, right-click **SiteColumns**, click **Add**, and then click **New Item**.
21. In the **Add New Item** dialog box, select the **Site Column** template.
22. In the **Name** box, type **Miles**, and then click **Add**.

23. In the **Elements.xml** code file, locate the following code:

```
Type="Text"
Required="FALSE"
```

24. Replace the located code with the following code

```
Type="Integer"
```

25. Required="TRUE"
26. In Solution Explorer, right-click **SiteColumns**, click **Add**, and then click **New Item**.
27. In the **Add New Item** dialog box, select the **Site Column** template.
28. In the **Name** box, type **EngineSize**, and then click **Add**.
29. In the **Elements.xml** code file, locate the following code:

```
Type="Text"
Required="FALSE"
```

30. Replace the located code with the following code

```
Type="Integer"
```

31. Required="TRUE"
32. In Solution Explorer, right-click **SiteColumns**, click **Add**, and then click **New Item**.
33. In the **Add New Item** dialog box, select the **Site Column** template.
34. In the **Name** box, type **Amount**, and then click **Add**.
35. In the **Elements.xml** code file, locate the following code:

```
Type="Text"
Required="FALSE"
```

36. Replace the located code with the following code

```
Type="Currency"
```

37. Required="TRUE"
38. On the **File** menu, click **Save All**.

### ► Task 2: Add the Mileage Claim Content Type

1. In Solution Explorer, right-click **MileageRecorder**, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select the **Content Type** template.
3. In the **Name** box, type **MileageClaim**, and then click **Add**.
4. Click **Finish**
5. In the **MileageClaim** content type designer, click **Click here to add a column**.
6. Type **Destination**, and then click **Destination**.
7. Type **Reason For Trip**, and then click **Reason For Trip**.
8. Type **Miles**, and then click **Miles**.
9. Type **Engine Size**, and then click **Engine Size**.

10. Type **Amount**, and then click **Amount**.
11. In the **Mileage Claim** content type designer, click the **Content Type** tab.
12. In the **Description** box, type **This content type defines a Mileage Claim for the Mileage Recorder app**.
13. On the **File** menu, click **Save All**.

► **Task 3: Add the Claims List**

1. In Solution Explorer, right-click **MileageRecorder**, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select the **List** template.
3. In the **Name** box, type **Claims**, and then click **Add**.
4. In the **SharePoint Customization Wizard** dialog box, click **Finish**.
5. In the **Claims** list designer, scroll to the bottom of the **Columns** page, and then click **Content Types**.
6. In the **Content Type Settings** dialog box, right-click **Item**, and then click **Delete**.
7. Right-click **Folder**, and then click **Delete**.
8. Type **MileageClaim**, click **MileageClaim**, and then click **OK**.
9. In the **Claims** list designer, right-click the **Title** column, and then click **Delete**.
10. In the **Claims** list designer, click the **List** tab.
11. In the **Description** box, type **This list stores mileage claims**.
12. On the **File** menu, click **Save All**.

**Results:** A cloud-hosted SharePoint app with a list for storing mileage claims configured in the app web.

## Exercise 2: Add Mileage Claim Creation Code

► **Task 1: Store URLs for Later Use**

1. In Solution Explorer, expand the **MileageRecorderRemoteWeb** project, and then expand **Controllers**.
2. Double-click **HomeController.cs**.
3. In the **Index** action method, locate the following line of code:

```
return View();
```

4. Immediately before the located code, insert the following code:

```
if (Request.QueryString["SPAppWebUrl"] != null)
{
 Session["SPAppWebUrl"] = Request.QueryString["SPAppWebUrl"];
}
```

5. Press Enter.

6. Type the following code:

```
if (Request.QueryString["SPHostUr1"] != null)
{
 Session["SPHostUr1"] = Request.QueryString["SPHostUr1"];
}
```

7. On the **File** menu, click **Save All**.

### ► Task 2: Code the Amount Calculation

1. In Solution Explorer, expand **Models**, and then double-click **MileageClaim.cs**.
2. Place the cursor within the **MileageClaim** class code block.
3. Type the following code:

```
public void calculateAmount()
{
}
```

4. Place the cursor within the **calculateAmount** method you just created.
5. Type the following code:

```
double result;
double lowMileageRate = 0.6;
double highMileageRate = 0.5;
```

6. Press Enter.
7. Type the following code:

```
if (this.EngineSize < 1000)
{
 lowMileageRate = 0.3;
 highMileageRate = 0.25;
}
```

8. Press Enter.
9. Type the following code:

```
if (this.Miles < 100)
{
 result = this.Miles * lowMileageRate;
}
else
{
 result = this.Miles * highMileageRate;
}
```

10. Press Enter.
11. Type the following code:

```
this.Amount = result;
```

12. On the **File** menu, click **Save All**.



► **Task 3: Complete the Mileage Claim Create Action**

1. In Solution Explorer, in the **Controllers** folder, double-click **MileageClaimController.cs**.
2. Locate the following line of code:

```
//Insert mileage claim creation code here
```

3. Replace the located code with the following code:

```
claim.calculateAmount();
```

4. Press Enter.
5. Type the following code:

```
string appWebUrl = Session["SPAppWebUrl"].ToString();
```

6. Press Enter.
7. Type the following code:

```
using (ClientContext context = new ClientContext(appWebUrl))
{
}
```

8. Place the cursor within the **using** code block you just created.
9. Type the following code:

```
List claimsList = context.Web.Lists.GetByTitle("Claims");
context.Load(claimsList);
```

10. Press Enter.
11. Type the following code:

```
ListItemCreationInformation creationInfo = new ListItemCreationInformation();
```

12. Press Enter.
13. Type the following code:

```
ListItem newClaim = claimsList.AddItem(creationInfo);
```

14. Press Enter.
15. Type the following code:

```
newClaim["Destination"] = claim.Destination;
newClaim["ReasonForTrip"] = claim.ReasonForTrip;
newClaim["Miles"] = claim.Miles;
newClaim["EngineSize"] = claim.EngineSize;
newClaim["Amount"] = claim.Amount;
```

16. Press Enter.
17. Type the following code:

```
newClaim.Update();
context.ExecuteQuery();
```

- On the **File** menu, click **Save All**.

**Results:** A cloud-hosted app that can create items in a SharePoint list.

### Exercise 3: Display Mileage Claims on the App Start Page

#### ► Task 1: Complete the Mileage Claim Index Action

- In the **MileageClaimController.cs** code file, locate the **Index** action method.
- In the **Index** action method, locate the following line of code:

```
//Obtain all the mileage claims here
```

- Replace the located code with the following code:

```
List<MileageClaim> claimsToDisplay = new List<MileageClaim>();
```

- Press Enter.

- Type the following code:

```
string appWebUrl = Session["SPAppWebUrl"].ToString();
```

- Press Enter.

- Type the following code:

```
using (ClientContext context = new ClientContext(appWebUrl))
{
}
```

- Place the cursor within the **using** code block you just created.

- Type the following code:

```
List claimsList = context.Web.Lists.GetByTitle("Claims");
context.Load(claimsList);
```

- Press Enter.

- Type the following code:

```
CamlQuery camlQuery = new CamlQuery();
```

- Press Enter.

- Type the following code:

```
ListItemCollection allClaims = claimsList.GetItems(camlQuery);
```

- Press Enter.

- Type the following code:

```
context.Load(allClaims);
context.ExecuteQuery();
```

16. Press Enter.

17. Type the following code:

```
foreach (ListItem sharepointClaim in allClaims)
{
}
```

18. Place the cursor within the **foreach** loop you just created.

19. Type the following code:

```
MileageClaim currentClaim = new MileageClaim();
```

20. Press Enter.

21. Type the following code:

```
currentClaim.Destination = sharepointClaim["Destination"].ToString();
currentClaim.ReasonForTrip = sharepointClaim["ReasonForTrip"].ToString();
currentClaim.Miles = (int)sharepointClaim["Miles"];
currentClaim.EngineSize = (int)sharepointClaim["EngineSize"];
currentClaim.Amount = (double)sharepointClaim["Amount"];
```

22. Press Enter.

23. Type the following code:

```
claimsToDisplay.Add(currentClaim);
```

24. Locate the following line of code:

```
return PartialView("Index");
```

25. Replace the located code with the following line of code:

```
return PartialView("Index", claimsToDisplay);
```

26. On the **File** menu, click **Save All**.

### ► Task 2: Test the Mileage Recorder App

1. On the **DEBUG** menu, click **Start Debugging**.
2. In the **Security Alert** dialog box, click **Yes**.
3. In the **Security Warning** dialog box, click **Yes**.
4. In the **Windows Security** dialog box, in the **User name** box, type **CONTOSO\Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
5. In the **Do you trust Mileage Recorder?** page, click **Trust It**.
6. In the **Mileage Recorder App** start page, click **Make a Claim**.
7. In the **Destination** box, type **Las Vegas**.
8. In the **Reason for Trip** box, type **Attending SharePoint conference**.
9. In the **Miles** box, type **400**.
10. In the **Engine Size** box, type **2000**, and then click **Create**.
11. Close Internet Explorer.

12. Close Visual Studio.
13. Close File Explorer.

**Results:** A completed autohosted SharePoint app.

# Lab B: Using the REST API with JavaScript

## Exercise 1: Creating List Relationships

### ► Task 1: Add Site Columns

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20488B-LON-SP-08**, and then click **Connect**.
2. To log on to **20488B-LON-SP-08**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **CONTOSO\Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then press Enter.
4. On the Start screen, click **Computer**.
5. In File Explorer, browse to **E:\Labfiles\LabB\Starter**.
6. Double-click **SiteSuggestionsApp.sln**.
7. If the **How do you want to open this type of file (.sln)?** dialog box appears, click **Visual Studio 2012**.
8. In Solution Explorer, right-click **SiteSuggestionsApp**, click **Add**, and then click **New Item**.
9. In the **Add New Item** dialog box, select the **Site Column** template.
10. In the **Name** box, type **Positive**, and then click **Add**.
11. In the **Element.xml** code file, locate the following code:

```
DisplayName="Positive"
Type="Text"
Required="FALSE"
```

12. Replace the located code with the following code:

```
DisplayName="Positive?"
Type="Boolean"
Required="TRUE"
```

13. In Solution Explorer, right-click **SiteSuggestionsApp**, click **Add**, and then click **New Item**.
14. In the **Add New Item** dialog box, select the **Site Column** template.
15. In the **Name** box, type **SuggestionLookup**, and then click **Add**.
16. In the **Element.xml** code file, locate the following code:

```
DisplayName="Suggestion Lookup"
Type="Text"
Required="FALSE"
```

17. Replace the located code with the following code:

```
DisplayName="Suggestion Lookup"
Type="Lookup"
List="Lists/Suggestions"
ShowField="Subject"
Required="TRUE"
```

18. On the **File** menu, click **Save All**.

### ► Task 2: Add the Vote Content Type

1. In Solution Explorer, right-click **SiteSuggestionsApp**, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select the **Content Type** template.
3. In the **Name** box, type **Vote**, and then click **Add**.
4. Click **Finish**.
5. In the **Vote** content type designer, click **Click here to add a column**.
6. Type **Positive**, and then click **Positive?**.
7. Type **Suggestion**, and then click **Suggestion Lookup**.
8. In the **Vote** content type designer, click the **Content Type** tab.
9. In the **Description** box, type **This content type defines a Vote for the Site Suggestions app**.
10. On the **File** menu, click **Save All**.

### ► Task 3: Add the Votes List

1. In Solution Explorer, right-click **SiteSuggestionsApp**, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select the **List** template.
3. In the **Name** box, type **Votes**, and then click **Add**.
4. In the **SharePoint Customization Wizard** dialog box, click **Finish**.
5. In the **Votes** list designer, scroll to the bottom of the **Columns** page, and then click **Content Types**.
6. In the **Content Type Settings** dialog box, right-click **Item**, and then click **Delete**.
7. Right-click **Folder**, and then click **Delete**.
8. Type **Vote**, click **Vote**, and then click **OK**.
9. In the **Votes** list designer, right-click the **Title** column, and then click **Delete**.
10. In the **Votes** list designer, click the **List** tab.
11. In the **Description** box, type **This list stores votes linked to the corresponding suggestions**.
12. On the **File** menu, click **Save All**.

**Results:** A SharePoint-hosted app with a lookup site column that links items in the Suggestions list with items in the Votes list.

## Exercise 2: Add Vote Recording

### ► Task 1: Add the recordVote Function

1. In Solution Explorer, expand the **Scripts** folder, and then double-click **App.js**.
2. Place the cursor within the **Contoso.SuggestionsApp** function, but before the **return** statement and outside any of the child functions.
3. Type the following code:

```
var recordVote = function (positive) {
};
```

4. Place the cursor within the anonymous function you just created.
5. Type the following code:

```
var votesListURL = _spPageContextInfo.webServerRelativeUrl +
 "/_api/web/lists/getByTitle('Votes')/items";
```

6. Press Enter.
7. Type the following code:

```
var formDigest = $('#_REQUESTDIGEST').val();
```

8. Press Enter.
9. Type the following code:

```
$.ajax({
});
```

10. Place the cursor within the **ajax()** function you just created.
11. Type the following code:

```
url: votesListURL,
type: "POST",
```

12. Press Enter.
13. Type the following code:

```
data: JSON.stringify({
 '__metadata': { 'type': 'SP.Data.VotesListItem' },
 'Positive': positive,
 'SuggestionLookupId': currentSuggestion.get_item('ID')
}),
```

14. Press Enter.
15. Type the following code:

```
headers: {
 'accept': 'application/json;odata=verbose',
 'content-type': 'application/json;odata=verbose',
 'X-RequestDigest': formDigest
},
```

16. Press Enter.
17. Type the following code:

```
success: function () {
 alert("Thank you for your vote");
},
```

18. Press Enter.
19. Type the following code:

```
error: function(err) {
 alert("Could not register your vote. Please try again later.");
}
```

### ► Task 2: Call the recordVote Function

1. In Solution Explorer, expand **Pages**, and then double-click **Default.aspx**.
2. Locate the following code:

```
<a>Like
```

3. Replace the located code with the following code:

```
Like
```

4. Locate the following code:

```
<a>Dislike
```

5. Replace the located code with the following code:

```
Dislike
```

6. On the **File** menu, click **Save All**.

**Results:** A simple SharePoint hosted app in which users can add positive or negative votes to each site suggestion.

## Exercise 3: Display Votes for Each Suggestion

### ► Task 1: Add the voteCount Function

1. In Solution Explorer, in the **Scripts** folder, double-click **App.js**.
2. Place the cursor within the **Contoso.SuggestionsApp** function, but before the **return** statement and outside any of the child functions.
3. Type the following code:

```
var voteCount = function() {
};
```

4. Place the cursor within the new function.

5. Type the following code:

```
var netVotes = 0;
```

6. Press Enter.

7. Type the following code:

```
var votesListURL = _spPageContextInfo.webServerRelativeUrl;
```

8. Press Enter.

9. Type the following code:

```
votesListURL += "/_api/web/lists/getByTitle('Votes')/items"
```

10. Press Enter.



11. Type the following code:

```
votesListURL += "?$filter=SuggestionLookup eq " + currentSuggestion.get_item('ID');
```

12. Press Enter.

13. Type the following code:

```
$.ajax({
});
```

14. Place the cursor within the `ajax()` function you just created.

15. Type the following code:

```
url: votesListURL,
type: "GET",
```

16. Press Enter.

17. Type the following code:

```
headers: { "accept": "application/json;odata=verbose" },
```

18. Press Enter.

19. Type the following code:

```
success: function (data) {
},
```

20. Place the cursor within the function you just created.

21. Type the following code:

```
$.each(data.d.results, function (i, result) {
});
```

22. Place the cursor within the anonymous function you just created.

23. Type the following code:

```
if (result.Positive) {
 netVotes ++;
} else {
 netVotes --;
}
```

24. Place the cursor after the `$.each()` loop but within the `success` function.

25. Type the following code:

```
$('#votes-count').html(netVotes);
```

26. Place the cursor after the end of the `success` function but within the `ajax()` function.

27. Type the following code:

```
error: function (err) {
}
```

28. Place the cursor within the function you just created.

29. Type the following code:

```
alert("Could not count the votes for this suggestion.");
```

30. On the **File** menu, click **Save All**.

► **Task 2: Call the voteCount Function**

1. In the App.js code file, locate the **onDisplaySuggestionSuccess** function.
2. Within the **onDisplaySuggestionSuccess** function, locate the following line of code:

```
$('#item-display').fadeIn('fast');
```

3. Immediately before the located code, insert the following line of code:

```
voteCount();
```

4. On the **File** menu, click **Save All**.

► **Task 3: Test the Suggestions App**

1. On the **DEBUG** menu, click **Start Debugging**.
2. If the **Windows Security** dialog box appears, in the **Username** box, type **Administrator**.
3. In the Password box, type **Pa\$\$w0rd**, and then click **OK**.
4. In the **Subject** box, type **A good idea**.
5. In the **Feedback** box, type **This is to test positive votes**, and then click **Submit Query**.
6. In the **Subject** box, type **An average idea**.
7. In the **Feedback** box, type **This is to test net votes**, and then click **Submit Query**.
8. In the **Subject** box, type **A bad idea**.
9. In the **Feedback** box, type **This is to test negative votes**, and then click **Submit Query**.
10. In the list of current suggestions, click **A good idea**.
11. Click **Like**, and then click **OK**.
12. Click **Like** a second time, and then click **OK**.
13. In the list of current suggestions, click **An average idea**.
14. Click **Like**, and then click **OK**.
15. Click **Dislike**, and then click **OK**.
16. In the list of current suggestions, click **A bad idea**.
17. Click **Dislike**, and then click **OK**.
18. Click **Dislike** a second time, and then click **OK**.
19. In the list of current suggestions, click **A good idea**.



**Note:** The app shows that the net vote for the suggestion is 2.

20. In the list of current suggestions, click **An average idea**.



**Note:** The app shows that the net vote for the suggestion is 0.

21. In the list of current suggestions, click **A bad idea**.



**Note:** The app shows that the net vote for the suggestion is -2.

22. Close Internet Explorer.

23. Close Visual Studio.

**Results:** A SharePoint-hosted app that uses a filtered REST API call to obtain and process the votes for each suggestion.



# Module 9: Developing Remote-hosted Apps for SharePoint

## Lab A: Configuring a Provider-Hosted SharePoint App

### Exercise 1: Configuring An S2S Trust Relationship

#### ► Task 1: Create and Register an X.509 Certificate

1. On the Start screen, type **PowerShell**, and then click **Windows PowerShell ISE**.
2. On the **File** menu, click **Open**.
3. In the **Open** dialog box, browse to the **E:\Labfiles\LabA\PowerShell\Starter** folder, click **CertificateCreator.ps1**, and then click **Open**.
4. Locate the following line of code:

```
Create a new folder here
```

5. Below the comment add the following code:

```
New-Item $certFolder -ItemType Directory -Force -Confirm:$false | Out-Null
```

6. Locate the following line of code:

```
Create the new certificate here
```

7. In the next line of code, replace the **<CertificateStoreLocationPlaceholder>** placeholder with **localMachine** so that the line reads as follows:

```
& $makecert -r -pe -n "CN=$remoteDomain" -b 01/01/2013 -e 01/01/2023 -eku 1.3.6.1.5.5.7.3.1 -ss my -sr localMachine -sky exchange -sy 12 -sp "Microsoft RSA SChannel Cryptographic Provider" $publicCertPath
```

8. Locate the following line of code:

```
Register the certificate here
```

9. In the next line of code, replace the **<SystemStoreLocationPlaceholder>** placeholder with **localMachine** so that the line reads as follows :

```
& $certmgr /add $publicCertPath /s /r localMachine root
```

10. Locate the following line of code:

```
Export the private key to a variable here
```

11. In the next line of code, replace the **<PasswordPlaceholder>** placeholder with **"Password"** so that the line reads as follows:

```
$privateCertBytes = $_.Export($type, "Password")
```

12. On the **File** menu, click **Save**.
13. On the **Debug** menu, click **Run/Continue** and wait for the script to finish.

14. Switch to File Explorer.
15. Browse to **C:\Certificates** and verify that it contains two files.



**Note:** The Windows PowerShell code you wrote has created the Certificates folder, the certificate file and the encrypted private key file in this folder.

### ► Task 2: Register a Trusted Security Token Issuer

1. Switch back to Windows PowerShell ISE.
2. On the **File** menu, click **Open**.
3. In the **Open** dialog box, click **TrustedTokenIssuerCreator.ps1**, and then click **Open**.
4. Locate the following line of code:

```
Create the new trusted token issuer here
```

5. In the next line of code, replace the **<CertificatePlaceholder>** placeholder with **\$publicCertificate** so that the line reads as follows:

```
New-SPTrustedSecurityTokenIssuer -Name $issuerID -RegisteredIssuerName $issuerName -
Certificate $publicCertificate -IsTrustBroker
```

6. On the **File** menu, click **Save**.
7. On the **Debug** menu, click **Run/Continue** and wait for the script to finish.

### ► Task 3: Register an App Principal

1. On the **File** menu, click **Open**.
2. In the **Open** dialog box, click **AppPrincipalCreator.ps1**, and then click **Open**.
3. Locate the following line of code:

```
Create the new registration here
```

4. In the next line of code, replace the **<RootWebPlaceholder>** placeholder with **\$targetWeb.RootWeb** so that the line reads as follows:

```
Register-SPAppPrincipal -NameIdentifier $AppIdentifier -Site $targetWeb.RootWeb -DisplayName
$appDisplayName
```

5. On the **File** menu, click **Save**.
6. On the **Debug** menu, click **Run/Continue** and wait for the script to finish.
7. Close all open windows.

**Results:** A working trust relationship that an app can use to authenticate with SharePoint and access resources.

## Exercise 2: Creating a Provider-hosted App

### ► Task 1: Configure the App Manifest File

1. On the Start screen, click **Visual Studio 2012**.
2. On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open** dialog box, navigate to the **E:\Labfiles\LabA\App\Starter\ContosoLedgersApp** folder, click **ContosoLedgersApp.sln**, and then click **Open**.
4. In Solution Explorer, right-click **AppManifest.xml**, and then click **View Code**.
5. Locate the following line of code:

```
<RemoteWebApplication ClientId="" />
```

6. Replace the located code with the following line of code:

```
<RemoteWebApplication ClientId="66142967-91BB-4C50-9B70-06375D5240BC" />
```

7. Locate the following line of code:

```
</App>
```

8. Immediately before the located code, insert the following lines of code:

```
<AppPermissionRequests>
</AppPermissionRequests>
```

9. Place the cursor within the **AppPermissionRequests** element you just created.
10. Type the following code:

```
<AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web" Right="Read" />
```

11. On the **FILE** menu, click **Save All**.

### ► Task 2: Configure Web.config

1. In Solution Explorer, double-click **Web.config**.
2. Locate the following line of code:

```
<add key="ClientId" value="" />
```

3. Replace the located code with the following line of code:

```
<add key="ClientId" value="66142967-91BB-4C50-9B70-06375D5240BC" />
```

4. On the **FILE** menu, click **Save All**.

### ► Task 3: Test the App

1. On the **DEBUG** menu, click **Start Debugging**.
2. In the **Security Alert** dialog box, click **Yes**.
3. In the **Security Warning** dialog box, click **Yes**.
4. On the **Do you trust Contoso Ledgers?** webpage, click **Trust It**.
5. Note the numbers of sales and purchases that the page reports.

6. In the Internet Explorer address bar, type **http://dev.contoso.com**, and then press Enter.
7. In the Quick Menu on the left, click **Site Contents**.
8. Click the **Sales Ledger** list, and then note the number of items in the list.
9. On the Quick Menu on the left, click **Site Contents**.
10. Click the **Purchase Ledger** list, and then note the number of items in the list
11. Close Internet Explorer.
12. Close Visual Studio.

**Results:** A simple provider-hosted app that uses an S2S trust to access SharePoint and displays item totals from lists in the host web.



# Lab B: Developing a Provider-Hosted SharePoint App

## Exercise 1: Working with SharePoint Data

### ► Task 1: Build the User Interface

1. On the Windows Start page, click **Computer**.
2. In File Explorer, browse to the following **E:\Labfiles\LabB\Starter\ContosoLedgersApp** folder.
3. Double-click **ContosoLedgersApp.sln**.
4. In the **How do you want to open this type of file (.sln)?** dialog box, click **Visual Studio 2012**.
5. In Solution Explorer, expand **Pages**, and then double-click **Default.aspx**.
6. Switch to File Explorer.
7. Browse to **E:\LabFiles\LabB\Markup**, and then double-click **SalesAndPurchaseTables.txt**.
8. In the **How do you want to open this type of file (.txt)?** dialog box, click **Notepad**.
9. In Notepad, on the **Edit** menu, click **Select All**.
10. On the **Edit** menu, click **Copy**.
11. Switch to Visual Studio.
12. Locate and select the following line of code:

```
<!-- Display Sales and Purchase Summary Tables Here -->
```

13. On the **EDIT** menu, click **Paste**.
14. On the **DEBUG** menu, click **Start Debugging**.
15. In the **Do you trust Contoso Ledgers** page, click **Trust It**.
16. Close Internet Explorer.
17. Close File Explorer.

### ► Task 2: Obtain Regions from SharePoint

1. In the Solution Explorer pane, right-click **Default.aspx**, and then click **View Code**.
2. Locate the following line of code:

```
//Get the three relevant lists
```

3. Immediately after the located code, insert the following lines of code:

```
List regionsList = clientContext.Web.Lists.GetByTitle("Regions");
```

4. `clientContext.Load(regionsList);`
5. Locate the following line of code:

```
//Get the items collections
```

6. Immediately after the located code, insert the following lines of code:

```
Microsoft.SharePoint.Client.ListItemCollection regionItems = regionsList.GetItems(new
CamlQuery());
clientContext.Load(regionItems);
```

7. On the **FILE** menu, click **Save All**.

### ► Task 3: Display Data in the User Interface

1. Locate the following line of code:

```
lblTotalNumberOfPurchases.Text = purchaseLedger.ItemCount.ToString();
```

2. Immediately after the located code, insert the following lines of code:

```
foreach (Microsoft.SharePoint.Client.ListItem regionItem in regionItems)
{
}
```

3. Place the cursor within the **foreach** loop you just created.

4. Type the following code:

```
TableRow saleRow = createRow(regionItem, saleItems);
```

5. Press Enter.

6. Type the following code:

```
SalesTable.Rows.Add(saleRow);
```

7. Press Enter.

8. Type the following code:

```
TableRow purchaseRow = createRow(regionItem, purchaseItems);
```

9. Press Enter.

10. Type the following code:

```
PurchasesTable.Rows.Add(purchaseRow);
```

11. On the **DEBUG** menu, click **Start Debugging**.

12. Close Internet Explorer.

**Results:** A provider-hosted app that displays a summary of global sales and purchases.

## Exercise 2: Using the Chrome Control

### ► Task 1: Set Up Chrome Control Scripts

1. In Solution Explorer, right-click the **Scripts** folder, point to **Add**, and then click **Existing Item**.
2. In the **Add Existing Item** dialog box, browse to **C:\Program Files\Common Files\microsoft shared\web server extensions\15\TEMPLATE\LAYOUTS**.
3. Click **sp.ui.controls.js**, and then click **Add**.
4. In Solution Explorer, double-click **Default.aspx**.
5. Locate the following line of code:

```
<script type="text/javascript" src="../Scripts/App.js"></script>
```

6. Immediately before the located code, insert the following line of code:

```
<script type="text/javascript" src="../Scripts/sp.ui.controls.js"></script>
```

7. Locate the following line of code:

```
<body>
```

8. Immediately after the located code, insert the following line of code:

```
<div id="chrome-control-placeholder"></div>
```

9. On the **FILE** menu, click **Save All**.

### ► Task 2: Render the Chrome Control

1. In Solution Explorer, expand **Scripts**, and then double-click **App.js**.
2. Locate the following line of code:

```
alert("Chrome Control is not yet implemented");
```

3. Replace the located code with the following code:

```
var options = {
 "appIconUrl": "../Images/AppIcon.png",
 "appTitle": "Contoso Ledgers App"
};
```

4. Press Enter.
5. Type the following code:

```
var navControl = new SP.UI.Controls.Navigation("chrome-control-placeholder", options);
```

6. Press Enter.
7. Type the following code:

```
navControl.setVisible(true);
```

8. Locate the following line of code:

```
//Call the chrome control render method here
```

9. Replace the located code with the following line of code:

```
Contoso.ChromeControl1.render();
```

10. On the **FILE** menu, click **Save All**.

► **Task 3: Test the App**

1. On the **DEBUG** menu, click **Start Debugging**.
2. If the **Do you trust Contoso Ledgers** page appears, click **Trust It**.
3. Click the **Tools** icon, and then click **F12 developer tools**.
4. Click **Cache**, and then click **Clear browser cache**.
5. In the **Clear Browser Cache** dialog box, click **Yes**.
6. On the **File** menu, click **Exit**.
7. On the Internet Explorer address bar, click **Refresh**.
8. In the **Windows Internet Explorer** dialog box, click **Retry**.
9. Click **Show All Content**, and then click **Retry**.
10. Close Internet Explorer.
11. Close Visual Studio.

**Results:** A complete Contoso Ledgers app that summarizes global sales and purchasing data for Contoso executives.

## Module 10: Publishing and Distributing Apps

# Lab A: Publishing an App to a Corporate Catalog

### Exercise 1: Creating an App Catalog

#### ► Task 1: Create an App Catalog

1. On the Start screen, type **SharePoint**, and then click **SharePoint 2013 Central Administration**.
2. On the Quick Launch menu, click **Apps**.
3. On the **Apps** page, under **App Management**, click **Manage App Catalog**.
4. On the **Manage App Catalog** page, make sure that **Create a new app catalog site** is selected, and then click **OK**.
5. On the **Create App Catalog** page, in the **Title** box, type **Contoso Intranet App Catalog**.
6. In the **Description** box, type **Publish apps here for use on the Contoso Intranet**.
7. In the **URL** list, click **/sites/**, in the text box that appears, type **appcatalog**.
8. In the **User name** box, type **Administrator**, and then click the **Check Names** icon.
9. In the **Users/Groups** box, type **All Users**, and then click the **Check Names** icon.
10. Scroll to the bottom of the page, and then click **OK**.
11. On the **Manage App Catalog** page, under **Site URL**, click **http://london/sites/appcatalog**.
12. On the Quick Launch menu, click **Apps for SharePoint**.



**Note:** There are no apps in the list because the app catalog is new.

13. Close Internet Explorer.

**Results:** A SharePoint farm with an app catalog set up in the default web application.

### Exercise 2: Creating an App Package

#### ► Task 1: Prepare the App for Publishing

1. In the Start screen, click **Computer**.
2. In File Explorer, browse to **E:\Labfiles\LabA\Starter**, and then double-click **SiteSuggestionsApp.sln**.
3. In the **How do you want to open this type of file (.sln)?** dialog box, click **Visual Studio 2012**.
4. In Visual Studio, in Solution Explorer, double-click **AppManifest.xml**.
5. In the App Manifest designer, on the **General** tab, in the **Version** text boxes, type **1.0.0.1**.
6. On the **FILE** menu, click **Save All**.

7. In Solution Explorer, click **Solution 'SiteSuggestionsApp'**.
8. In the **Properties** pane, in the **Active config** property, click **Release|Any CPU**.
9. On the **BUILD** menu, click **Build Solution**.

► **Task 2: Package the App**

1. In File Explorer, browse to **E:\Labfiles\LabA\Starter\SiteSuggestionsApp\bin\Release**.
2. Note the contents of the folder.
3. Switch to Visual Studio.
4. In Solution Explorer, right-click **SiteSuggestionsApp**, and then click **Publish**.
5. In the **Publish apps for Office and SharePoint** dialog box, read the summary information.
6. Clear the **Open output folder after successful packaging** check box, and then click **Finish**.
7. Switch to File Explorer, and note the contents of the Release folder.
8. Double-click **app.publish**, and then double-click **1.0.0.1**.
9. Note the contents of the folder.

**Results:** An app package that can be used to publish the Site Suggestions app in a SharePoint app catalog.

### Exercise 3: Publishing an App Package

► **Task 1: Publish the App**

1. On the Start screen, click **Internet Explorer**.
2. In the address bar, type **http://london/sites/appcatalog**, and then press Enter.
3. On the Quick Launch menu, click **Apps for SharePoint**.
4. On the **Apps for SharePoint** page, click **new app**.
5. In the **Add a document** dialog box, click **Browse**.
6. In the **Choose File to Upload** dialog box, browse to **E:\Labfiles\LabA\Starter\SiteSuggestionsApp\bin\Release\app.publish\1.0.0.1**.
7. Click **SiteSuggestionsApp.app**, and then click **Open**.
8. In the **Add a document** dialog box, click **OK**.
9. In the **Apps for SharePoint - SiteSuggestionsApp.app** dialog box, in the **Short Description** box, type **This app collects feedback on SharePoint sites from users**.
10. In the **Description** box, type **Users can submit new ideas for each site and vote on other people's ideas**.
11. Under **Category**, click **Specify your own value**, and then type **Site Administration**.
12. In the **Publisher Name** box, type **Contoso**.
13. Select the **Featured** check box, and then click **Save**.

► **Task 2: Test the App Catalog**

1. In Internet Explorer, in the address bar, type **http://dev.contoso.com**, and then press Enter.
2. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, and in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
3. On the Quick Launch menu, click **Site Contents**.
4. On the **Site Contents** page, in the list of apps for the site, click **add an app**.
5. On the Quick Launch menu, click **From Your Organization**.



**Note:** The SiteSuggestionsApp is the only app displayed because it is the only app published in the new app catalog.

6. Close Internet Explorer.
7. Close Visual Studio.
8. Close File Explorer.

**Results:** A published SharePoint hosted app in a corporate app catalog.

# Lab B: Installing, Updating, and Uninstalling Apps

## Exercise 1: Installing an App

### ► Task 1: Install the Site Suggestions App from the App Catalog

1. On the Start screen, click **Internet Explorer**.
2. In the address bar, type **http://dev.contoso.com**, and then press Enter.
3. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, and in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
4. On the Quick Launch menu, click **Site Contents**.
5. On the **Site Contents** page, in the list of apps, click **add an app**.
6. On the Quick Launch menu, click **From Your Organization**.
7. Under **SiteSuggestionsApp**, click **App Details**.
8. On the **SiteSuggestionsApp** page, click **ADD IT**.
9. In the **Do you trust SiteSuggestionsApp?** dialog box, review the permission requests for the Site Suggestions app, and then click **Trust It**.



**Note:** SharePoint adds and configures the new app.

### ► Task 2: Test the App

1. On the **Site Contents** page, in the list of apps, click **SiteSuggestionsApp**.
2. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, and in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
3. On the **Site Suggestions** page, in the **Subject** box, type **Test Suggestion**.
4. In the **Feedback** box, type **This is just to test this deployment**, and then click **Submit Query**.
5. Under **Here are the current suggestions**, click **Test Suggestion**. SharePoint displays the details.
6. Close Internet Explorer.

**Results:** A deployed and functional SharePoint-hosted app.



## Exercise 2: Upgrading an App

### ► Task 1: Repackage Version 2

1. On the Start screen, click **Computer**.
2. In File Explorer, browse to **E:\Labfiles\LabB\Starter**, and then double-click **SiteSuggestionsApp.sln**.
3. In Solution Explorer, double-click **AppManifest.xml**.
4. In the **Version** text boxes, type **2.0.0.0**.
5. On the **FILE** menu, click **Save All**.
6. In Solution Explorer, click **Solution 'SiteSuggestionsApp'**.
7. In the **Properties** pane, in the **Active config** property, click **Release|Any CPU**.
8. In Solution Explorer, right-click **SiteSuggestionsApp**, and then click **Publish**.
9. In the **Publish apps for Office and SharePoint** dialog box, clear the **Open output folder after successful packaging** check box, and then click **Finish**.

### ► Task 2: Update the App Catalog

1. On the Start screen, click **Internet Explorer**.
2. In the address bar, type **http://london/sites/appcatalog**, and then press Enter.
3. On the Quick Launch menu, click **Apps for SharePoint**.
4. On the **Apps for SharePoint** page, click **new app**.
5. In the **Add a document** dialog box, click **Browse**.
6. In the **Choose File to Upload** dialog box, browse to **E:\Labfiles\LabB\Starter\SiteSuggestionsApp\bin\Release\app.publish\2.0.0.0**.
7. Click **SiteSuggestionsApp.app**, and then click **Open**.
8. In the **Version Comments** box, type **New voting functionality**, and then click **OK**.
9. In the **Apps for SharePoint - SiteSuggestionsApp.app** dialog box, on the ribbon, click **Save**.



**Note:** Notice that the version number for the app has increased to 2.0.0.0

### ► Task 3: Update a Deployed App

1. In Internet Explorer, in the address bar, type **http://dev.contoso.com**, and then press Enter.
2. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, and in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
3. On the Quick Launch menu, click **Site Contents**.
4. On the **Site Contents** page, point to **SiteSuggestionsApp**, click the ellipsis, and then click **ABOUT**.
5. Notice that there is a new version of the app, and then click **GET IT**.
6. In the **Do you trust SiteSuggestionsApp?** dialog box, click **Trust It**.
7. On the **Site Contents** page, when the app deployment is complete, click **SiteSuggestionsApp**.
8. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, and in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.

9. On the **Site Suggestions** page, in the **Subject** box, type **Testing version 2**.
10. In the **Feedback** box, type **Version 2 includes voting functionality**, and then click **Submit Query**.
11. Under **Here are the current suggestions**, click **Testing version 2**, and then click **Like**.
12. In the **Message from webpage** dialog box, click **OK**.
13. Under **Here are the current suggestions**, click **Testing version 2**.



**Note:** The app displays the number of votes that the suggestion has received.

**Results:** An upgraded app deployed through an app catalog.

### Exercise 3: Removing an App

#### ► Task 1: Remove the App from the App Catalog

1. In Internet Explorer, in the address bar, type **http://london/sites/appcatalog**, and then press Enter.
2. On the Quick Launch menu, click **Apps for SharePoint**.
3. On the **Apps for SharePoint** page, in the **SiteSuggestionsApp** row, click the **Edit** icon.
4. On the ribbon, click **Delete Item**.
5. In the **Message from webpage** dialog box, click **OK**.
6. In the address bar, type **http://dev.contoso.com**, and then press Enter.
7. On the Quick Launch menu, click **Site Contents**.
8. On the **Site Contents** page, click **add an app**.
9. On the Quick Launch menu, click **From Your Organization**.



**Note:** No apps are displayed because the app catalog is now empty.

#### ► Task 2: Remove the Deployed Instance of the Site Suggestions App

1. In the address bar, type **http://dev.contoso.com**, and then press Enter.
2. On the Quick Launch menu, click **SiteSuggestionsApp**.



**Note:** Notice that the instance of the app remains deployed even though the app has been removed from the app catalog.

3. At the upper left of the page, click **Contoso Development Site**.
4. On the Quick Launch menu, click **Site Contents**.
5. On the **Site Contents** page, in the list of apps, point to **SiteSuggestionsApp**, click the ellipsis, and then click **REMOVE**.
6. In the **Message from webpage** dialog box, click **OK**.

7. Close Internet Explorer.
8. Close Visual Studio.
9. Close File Explorer.

**Results:** A SharePoint site with no custom apps installed or available to install.



## Module 11: Automating Business Processes

# Lab A: Building Workflows in Visio 2013 and SharePoint Designer 2013

### Exercise 1: Creating Workflows by Using Visio

#### ► Task 1: Create a New Workflow in Visio 2013

1. Start the 20488B-LON-SP-11 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visio 2013**, and then press Enter.
4. In the Visio window, click **Flowchart**, and then click **Microsoft SharePoint 2013 Workflow**.
5. In the **Microsoft SharePoint 2013 Workflow** window, make sure that **US Units** is selected, and then click **Create**.

#### ► Task 2: Implement Workflow Logic

1. In the editor, double-click **Stage 1**, type **Write Leaflet**, and then press Esc.
2. In the **Shapes** task pane, click **Actions - SharePoint 2013 Workflow**, and then click and drag **Assign a task** to the center of the stage.
3. In the newly created task, double-click **Assign a task**, type **Pharmacologist to write leaflet**, and then press Esc.
4. In the **Shapes** task pane, click **Components - SharePoint 2013 Workflow**, and then click and drag **Stage** to the right of the **Write Leaflet** stage.
5. In the editor, double-click **Stage 2**, type **Copy Edit**, and then press Esc.
6. On the ribbon, on the **HOME** tab, in the **Tools** group, click **Connector**.
7. Click the red square on the **Write Leaflet** stage, drag to the green triangle on the **Copy Edit** stage to link the two objects together, and then press Esc twice.
8. In the **Shapes** task pane, click **Actions - SharePoint 2013 Workflow**, and then click and drag **Assign a task** to the center of the **Copy Edit** stage.
9. In the newly created task, double-click **Assign a task**, type **Editor to review and log any changes required**, and then press Esc.
10. In the **Shapes** task pane, click **Conditions - SharePoint 2013 Workflow**, and then click and drag **If any value equals value** to the right of the existing stage.
11. Double-click **If any value equals value**, type **Ready to publish?** and then press Esc.
12. On the ribbon, on the **HOME** tab, in the **Tools** group, click **Connector**.
13. Click the red square on the **Copy Edit** stage, drag to the equals sign on the newly created condition to link the two objects together, and then press Esc twice.
14. In the **Shapes** task pane, click **Components - SharePoint 2013 Workflow**, and then click and drag **Stage** to the right of the **Ready to publish?** condition.
15. In the editor, double-click **Stage 3**, type **Publish**, and then press Esc.
16. On the ribbon, on the **HOME** tab, in the **Tools** group, click **Connector**.

17. Click the **Ready to publish?** condition, drag to the side of the green triangle on the **Publish** stage to link the two objects together, and then press Esc twice.
  18. Right-click the new connector, click **Yes**, and then press Esc.
  19. On the ribbon, on the **HOME** tab, in the **Tools** group, click **Connector**.
  20. Click the **Ready to publish?** condition, drag to the side of the green triangle on the **Write Leaflet** stage to link the two objects together, and then press Esc twice.
  21. Right-click the new connector, click **No**, and then press Esc
  22. In the **Shapes** task pane, click **Actions - SharePoint 2013 Workflow**, and then click and drag **Assign a task** to the center of the **Publish** stage.
  23. In the newly created task, double-click **Assign a task**, type **Manager to publish leaflet**, and then press Esc.
  24. In the **Shapes** task pane, click **Components - SharePoint 2013 Workflow**, and then click and drag **Stage** to the right of the **Publish** stage.
  25. In the editor, double-click **Stage 4**, type **Annual Review**, and then press Esc.
  26. On the ribbon, on the **HOME** tab, in the **Tools** group, click **Connector**.
  27. Click the red square on the **Publish** stage, drag to the side of the green triangle on the **Annual Review** stage to link the two objects together, and then press Esc twice.
  28. In the **Shapes** task pane, click **Actions - SharePoint 2013 Workflow**, and then click and drag **Add time to date** to the center of the **Annual Review** stage.
  29. In the **Shapes** task pane, click **Actions - SharePoint 2013 Workflow**, and then click and drag **Assign a task** to the center of the **Annual Review** stage.
  30. In the newly created task, double-click **Assign a task**, type **Manager to check that the leaflet is still current**, and then press Esc.
- **Task 3: Validate and Save the Workflow**
1. On the ribbon, on the **PROCESS** tab, in the **Diagram Validation** group, click **Check Diagram**.
  2. In the **Microsoft Visio** dialog box, click **OK**.
  3. On the ribbon, on the **FILE** menu, click **Save As**, and then click **Browse**.
  4. In the **Save As** dialog box, browse to the **E:\Labfiles\Starter** folder, in the **File name** box, type **Leaflet**, and then click **Save**.
  5. On the ribbon, on the **FILE** menu, click **Close**, and then close Visio.

**Results:** After completing this exercise, you should have implemented workflow logic by using Visio.

## Exercise 2: Editing Workflows by Using SharePoint Designer

### ► Task 1: Import a Workflow into SharePoint Designer

1. On the Start screen, type **SharePoint Designer**, and then press Enter.
2. In SharePoint Designer screen, click **Open Site**.
3. In the **Open Site** dialog box, in the **Site name** box, type **http://team.contoso.com**, and then click **Open**.
4. In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
5. In SharePoint Designer, in the **Navigation** pane, click **Workflows**.
6. On the ribbon, on the **WORKFLOWS** tab, in the **Manage** group, click **Import from Visio**, and then click **Import Visio 2013 Diagram**.
7. In the **Import Workflow from Visio Drawing** dialog box, browse to the **E:\Labfiles\Starter** folder, click **Leaflet.vsd**, and then click **Open**.
8. In the **Create Workflow** dialog box, in the **Name** box, type **Leaflet Workflow**, in the **Description** box, type **Workflow to guide product leaflets through the edit and publish processes**, in the **Workflow Type** list, click **Reusable Workflow**, and then click **OK**.
9. In the **Leaflet Workflow** pane, review the workflow and note that it appears as it did in Visio.

### ► Task 2: Update the Workflow by Using the Text-Based Designer

1. On the ribbon, on the **WORKFLOW** tab, in the **Manage** group, click **Views**.



**Note:** This switches the view to the text-based designer, where you can customize the workflow.

2. Review the existing stages and note how they each include one or more *Go to stage* steps to move to the next stage in the process.
3. In the **Write Leaflet** stage, click **this user**.
4. In the **Assign a Task** dialog box, in the **Participant** box, type **Paul**, in the **Task Title** box, type **Write product leaflet**, and then click **OK**.
5. In the **Copy Edit** stage, click **this user**.
6. In the **Assign a Task** dialog box, in the **Participant** box, type **Danny**, in the **Task Title** box, type **Full copy edit of product leaflet**, and then click **OK**.
7. In the **Copy Edit** stage, in the **Transition to stage** section, click the first **value**, and then click the function button.
8. In the **Define Workflow Lookup** dialog box, in the **Data source** list, click **Workflow Variables and Parameters**, in the **Field from source** list, click **Variable: Outcome1**, and then click **OK**.
9. In the **Copy Edit** stage, in the **Transition to stage** section, click **value**, and then click **Approved**.
10. In the **Publish** stage, click **this user**.
11. In the **Assign a Task** dialog box, in the **Participant** box, type **Dominik**, in the **Task Title** box, type **Leaflet ready for publishing**, and then click **OK**.
12. In the **Annual Review** stage, click the first **0**.

13. Type **12**, and then press Enter.
14. Click **date**, and then click the ellipsis button.
15. In the **Date Value** dialog box, click **Current date**, and then click **OK**.
16. Click under **Add 12 months**, type **Pause**, and then press Enter.
17. Click **Pause until Date**, click **this time**, and then click the function button.
18. In the **Lookup for Date/Time** dialog box, in the **Data source** list, click **Workflow Variables and Parameters**, in the **Field from source** list, click **Variable: date**, and then click **OK**.
19. Click **this user**.
20. In the **Assign a Task** dialog box, in the **Participant** box, type **Dominik**, in the **Task Title** box, type **Annual review due**, and then click **OK**.

► **Task 3: Publish and Test the Workflow**

1. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Check for Errors**.
2. In the **Microsoft SharePoint Designer** dialog box, click **OK**.
3. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Save**.
4. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Publish**.
5. On the ribbon, on the **WORKFLOW** tab, in the **Manage** group, click **Workflow Settings**.
6. On the ribbon, on the **WORKFLOW SETTINGS** tab, in the **Manage** group, click **Associate to List**, and then click **Product Leaflets**.
7. In the **Windows Security** dialog box, in the **User name** box, type **Paul**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
8. In Internet Explorer, on the **Settings - Add a Workflow** page, in the **Name** box, type **Leaflet Workflow**, select the **Creating a new item will start this workflow** check box, and then click **OK**.
9. On the Quick Launch menu, click **Product Leaflets**.
10. On the Product Leaflets page, click **new document**.
11. In the **Add a document** dialog box, click **Browse**.
12. In the **Choose File to Upload** dialog box, browse to the **E:\Labfiles\Starter** folder, click **ProductA.docx**, and then click **Open**.
13. In the **Add a document** dialog box, click **OK**.
14. On the Quick Launch menu, click **Site Contents**.
15. On the Site Contents page, click **Workflow Tasks**.
16. Click **Write product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
17. Click **Approved** to approve the item and move the workflow to the next stage.
18. In the Title bar, click **Paul West**, and then click **Sign Out**.
19. In the **Windows Internet Explorer** dialog box, click **Yes**.
20. On the Start screen, click **Internet Explorer**.
21. In the **Windows Security** dialog box, in the **User name** box, type **Danny**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.



22. In the Quick Launch menu, click Site Contents.
23. On the Site Contents page, click **Workflow Tasks**.
24. Click **Full copy edit of product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
25. Click **Approved** to approve the item and move the workflow to the next stage.
26. In the Title bar, click **Danny Levin**, and then click **Sign Out**.
27. In the **Windows Internet Explorer** dialog box, click **Yes**.
28. On the Start screen, click **Internet Explorer**.
29. In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
30. On the Quick Launch menu, click **Site Contents**.
31. On the Site Contents page, click **Workflow Tasks**.
32. Click **Leaflet ready for publishing**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
33. Click **Approved** to approve the item and the workflow will then pause for 12 months until the annual review is due.
34. In the Title bar, click **Dominik Dubicki** and then click **Sign Out**.
35. In the **Windows Internet Explorer** dialog box, click **Yes**.
36. On the Start screen, click **Internet Explorer**.
37. In the **Windows Security** dialog box, in the **User name** box, type **Paul**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
38. On the Quick Launch menu, click **Site Contents**.
39. On the Quick Launch menu, click **Product Leaflets**.
40. On the Product Leaflets page, click **new document**.
41. In the **Add a document** dialog box, click **Browse**.
42. In the **Choose File to Upload** dialog box, browse to the **E:\Labfiles\Starter** folder, click **ProductB.docx**, and then click **Open**.
43. In the **Add a document** dialog box, click **OK**.
44. On the Quick Launch menu, click **Site Contents**.
45. On the Site Contents page, click **Workflow Tasks**.
46. Click **Write product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
47. Click **Approved** to approve the item and move the workflow to the next stage.
48. In the Title bar, click **Paul West**, and then click **Sign Out**.
49. In the **Windows Internet Explorer** dialog box, click **Yes**.
50. On the Start screen, click **Internet Explorer**.
51. In the **Windows Security** dialog box, in the **User name** box, type **Danny**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.

52. On the Quick Launch menu, click **Site Contents**.
53. On the Site Contents page, click **Workflow Tasks**.
54. Click **Full copy edit of product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
55. Click **Rejected** to reject the item and send the leaflet back to Paul for rewriting.
56. In the Title bar, click **Danny Levin**, and then click **Sign Out**.
57. In the **Windows Internet Explorer** dialog box, click **Yes**.
58. On the Start screen, click **Internet Explorer**.
59. In the **Windows Security** dialog box, in the **User name** box, type **Paul**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
60. On the Quick Launch menu, click **Site Contents**.
61. On the Site Contents page, click **Workflow Tasks**.
62. Verify that because Danny rejected the leaflet, the workflow has returned to the **Write leaflet** task assigned to Paul. In the Title bar, click **Paul West**, and then click **Sign Out**.
63. In the **Windows Internet Explorer** dialog box, click **Yes**.
64. In SharePoint Designer, on the **FILE** menu, click **Close**.
65. Close SharePoint Designer.

**Results:** After completing this exercise, you should have created and published the Leaflet workflow by using SharePoint Designer.

# Lab B: Creating Workflow Actions in Visual Studio 2012

## Exercise 1: Creating Custom Workflow Actions

### ► Task 1: Create a Custom Workflow Action Project

1. Start the 20488B-LON-SP-11 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
4. On the **FILE** menu, point to **New**, and then click **Project**.
5. In the **New Project** dialog box, in the **Templates** list, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
6. In the center pane, click **SharePoint 2013 - Empty Project**, in the **Name** box, type **ProductDistributionProject**, in the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
7. In the **SharePoint Customization Wizard**, in the **What site do you want to use for debugging?** box, type **http://team.contoso.com**, ensure that **Deploy as a sandboxed solution** is selected, and then click **Finish**.
8. In Solution Explorer, right-click **ProductDistributionProject**, point to **Add**, and then click **New Item**.
9. In the **Add New Item - ProductDistributionProject** dialog box, click **Workflow Custom Activity**, in the **Name** box, type **Distribution**, and then click **Add**.

### ► Task 2: Implement Workflow Logic

1. In the designer pane, click **Sequence**, and then at the bottom of the designer pane, click **Variables**.
2. Click **Create Variable**, type **WebUri**, and then press Enter.
3. Click **Create Variable**, type **WebResponse**, and then press Enter. In the **Variable type** column, click **Browse for Types**.
4. In the **Browse and Select a .Net Type** dialog box, expand **Microsoft.Activities [1.0.0.0]**, expand **Microsoft.Activities**, click **DynamicValue**, and then click **OK**.
5. At the bottom of the designer pane, click **Arguments**.
6. Click **Create Argument**, type **ProdName**, and then press Enter.
7. Click **Create Argument**, type **ProdDist**, in the **Direction** list, click **Out**, and then press Enter.
8. In the Toolbox, in the **Primitives** group, click **Assign**, and then drag it to the **Drop activity here** text in the designer pane.
9. Click **To**, type **WebUri**, and then press Enter.
10. In the **Properties** pane, click the ellipsis next to **Value**.
11. In the **Expression Editor** dialog box, in the **Value (InArgument)** box, type **"http://localhost:32999/ProductDistribution/Products.svc/ProductDists(" + ProdName + ")/Distribution"**, and then click **OK**.
12. In the Toolbox, in the **Messaging** group, click **HttpSend**, and then drag it to below the **Assign** activity in the **Sequence** in the designer pane.
13. Click **Enter a C# expression**, type **WebUri**, and then press Enter.

14. In the **Properties** pane, in the **Method** list, click **GET**.
15. Click the ellipsis next to **RequestHeaders**.
16. In the **RequestHeaders** dialog box, click **Create Header**, type **Accept**, click under **Header Value**, type "**application/json; odata=verbose**", and then press Enter.
17. Click **Create Header**, type **Content-Type**, click in the **Header Value** column, type "**application/json; odata=verbose**", and then click **OK**.
18. Click the ellipsis next to **ResponseContent**.
19. In the **Expression Editor** dialog box, in the **ResponseContent (OutArgument)** box, type **WebResponse**, and then click **OK**.
20. In the Toolbox, in the **DynamicValue** group, click **GetDynamicValueProperty<T>**, and then drag it to under the **HttpSend** activity in the **Sequence** in the designer pane.
21. In the **Select Types** dialog box, click **String**, and then click **OK**.
22. In the **Properties** pane, in the **PropertyName** box, type "**d/Distribution**", and then press Enter.
23. In the **Result** box, type **ProdDist**, and then press Enter.
24. In the **Source** box, type **WebResponse**, and then press Enter.
25. On the **FILE** menu, click **Save All**.

### ► Task 3: Create the actions4 File

1. In Solution Explorer, expand **Distribution**, and then double-click **Distribution.actions4**.
2. In the **RuleDesigner** opening tag, change the contents of the **Sentence** attribute as follows:

```
<RuleDesigner Sentence="Get distribution area for %1 (output to %2)">
```

3. Add the following markup to the body of the **RuleDesigner** element.

```
<FieldBind Field="ProdName" Text="Product name" Id="1" DesignerType="TextArea" />
<FieldBind Field="ProdDist" Text="Product distribution" Id="2" DesignerType="ParameterNames" />
```

4. Add the following markup to the body of the **Action** element.

```
<Parameters>
<Parameter Name="ProdName" Type="System.String, mscorlib" Direction="In"
DesignerType="TextArea" Description="The name of the product" />
<Parameter Name="ProdDist" Type="System.String, mscorlib" Direction="Out"
DesignerType="ParameterNames" Description="The distribution for the product" />
</Parameters>
```

5. On the **FILE** menu, click **Save All**.

### ► Task 4: Deploy the Workflow

1. On the **Build** menu, click **Build Solution**.
2. On the **Build** menu, click **Deploy Solution**.
3. On the Start screen, type **Internet Explorer**, and then press Enter.
4. In the **Windows Security** dialog box, in the **User name** box, type **Administrator**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
5. Click **Settings**, click **Site settings**, and then in the Site Actions group, click **Manage site features**.

6. Locate the **ProductDistributionProject Feature1** item in the list and verify that it is **Active**.
7. Close Internet Explorer and then close Visual Studio.

**Results:** After completing this exercise, you should have created and deployed a workflow custom activity.

## Exercise 2: Using a Custom Workflow in SharePoint Designer

### ► Task 1: Add the Custom Workflow to the SharePoint Designer Workflow



**Note:** Before opening SharePoint Designer, you need to clear the cache folders so that it can download the new workflow content from the server.

1. On the Start screen, type **File Explorer**, and then press Enter.
2. Click in the address bar, type **%APPDATA%\Microsoft\Web Server Extensions\Cache**, and then press Enter.
3. On the ribbon, in the **Select** group, click **Select all**, and then in the **Organize** group, click **Delete**.
4. Click in the address bar, type **%USERPROFILE%\AppData\Local\Microsoft\WebsiteCache**, and then press Enter.
5. On the ribbon, in the **Select** group, click **Select all**, and then in the **Organize** group, click **Delete**.
6. Close File Explorer.
7. On the Start screen, type **SharePoint Designer**, and then press Enter.
8. In SharePoint Designer, click **Open Site**.
9. In the **Open Site** dialog box, in the **Site name** box, type **http://team.contoso.com**, and then click **Open**.
10. In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
11. In the **Navigation** pane, click **Workflows**, in the **Workflows** pane, click **Leaflet Workflow**, and then on the ribbon, in the **Edit** group, click **Edit Workflow**.
12. Click above the **Publish** stage, and then on the ribbon, click **Stage**.
13. Click **Stage 5**, type **Get Product Distribution**, and then press Enter.
14. On the ribbon, click **Action**, and then click **Distribution**.
15. Click **ProductName** and then click the function button.
16. In the **Lookup for String** dialog box, in the **Field from source** list, click **Title**, and then click **OK**.
17. Click **(Insert go-to actions with conditions for transitioning to the stage)**, type **Go to**, and then press Enter.
18. Click **a stage**, and then click **Publish**.
19. In the **Publish** stage, click **Dominik Dubicki**.
20. In the **Assign a Task** dialog box, by the **Task Title**, click the ellipsis.

21. In the **String Builder** dialog box, click at the end of the existing text, press the Spacebar, type - **publish to**, and then click **Add or Change Lookup**.
22. In the **Lookup for String** dialog box, in the **Data source** list, click **Workflow Variables and Parameters**, in the **Field from source** list, click **Variable: Product distribution**, and then click **OK**.
23. In the **String Builder** dialog box, click **OK**, and then in the **Assign a Task** dialog box, click **OK**.
24. In the **Copy Edit** stage, click **Publish**, and then click **Get Product Distribution**.
25. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Check for Errors**.
26. In the **Microsoft SharePoint Designer** dialog box, click **OK**.
27. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Save**.
28. On the ribbon, on the **WORKFLOW** tab, in the **Save** group, click **Publish**.
29. Close SharePoint Designer.

### ► Task 2: Test the Workflow

1. On the Start screen, click **Internet Explorer**.
2. In the **Windows Security** dialog box, in the **User name** box, type **Paul**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
3. On the Quick Launch menu, click **Product Leaflets**, and then click **new document**.
4. In the **Add a document dialog** box, click **Browse**, browse to the **E:\Labfiles\Starter** folder, click **ProductC.docx**, click **Open**, and then click **OK**.
5. On the Quick Launch menu, click **Site Contents**.
6. On the Site Contents page, click **Workflow Tasks**.
7. In the **ProductC.docx** row, click **Write product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
8. Click **Approved** to approve the item and move the workflow to the next stage.
9. In the Title bar, click **Paul West** and then click **Sign Out**.
10. In the **Windows Internet Explorer** dialog box, click **Yes**.
11. On the Start screen, click **Internet Explorer**.
12. In the **Windows Security** dialog box, in the **User name** box, type **Danny**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.
13. On the Quick Launch menu, click **Site Contents**.
14. On the Site Contents page, click **Workflow Tasks**.
15. Click **Full copy edit of product leaflet**, and then on the ribbon, on the **VIEW** tab, in the **Manage** group, click **Edit Item**.
16. Click **Approved** to approve the item and move the workflow to the next stage.
17. In the Title bar, click **Danny Levin**, and then click **Sign Out**.
18. In the **Windows Internet Explorer** dialog box, click **Yes**.
19. On the **Start** screen, click **Internet Explorer**.
20. In the **Windows Security** dialog box, in the **User name** box, type **Dominik**, in the **Password** box, type **Pa\$\$w0rd**, and then click **OK**.

21. On the Quick Launch menu, click **Site Contents**.
22. On the Site Contents page, click **Workflow Tasks**.
23. Verify that the new task is titled **Leaflet ready for publishing - publish to International**.
24. In the Title bar, click **Dominik Dubicki** and then click **Sign Out**.
25. In the Windows Internet Explorer dialog box, click Yes.

**Results:** After completing this exercise, you should have used and tested a custom workflow action.





## Module 12: Managing Taxonomy

# Lab A: Working with Content Types

### Exercise 1: Create a System to Capture Vacation Requests

#### ► Task 1: Create an Empty SharePoint 2013 Project

1. Start the 20488B-LON-SP-12 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start Page, click **New Project**.
5. In the **New Project** dialog box, in the navigation pane, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
6. In the center pane, click **SharePoint 2013 - Empty Project**.
7. In the **Name** box, type **VacationRequests**.
8. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
9. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://hr.contoso.com**, and then click **Validate**.
10. In the **Microsoft Visual Studio** dialog box, verify that the dialog reports a successful connection, and then click **OK**.
11. Click **Deploy as a farm solution**, and then click **Finish**.

#### ► Task 2: Create New Site Columns

1. In Solution Explorer, right-click the **VacationRequests** project node, point to **Add**, and then click **New Item**.
2. In the center pane, click **Site Column**.
3. In the **Name** box, type **ContosoDepartment**, and then click **Add**.
4. In the **Elements.xml** file, amend the **Field** element as shown in bold:

```
<Field
 ID="{...}"
 Name="ContosoDepartment"
 DisplayName="Contoso Department"
 Type="Choice"
 Required="TRUE"
 Group="Contoso Columns">
 <CHOICES>
 <CHOICE>Finance</CHOICE>
 <CHOICE>Human Resources</CHOICE>
 <CHOICE>IT</CHOICE>
 <CHOICE>Legal</CHOICE>
 <CHOICE>Manufacturing</CHOICE>
 <CHOICE>Research</CHOICE>
 </CHOICES>
</Field>
```

5. Save and close the **Elements.xml** file.
6. In Solution Explorer, right-click the **VacationRequests** project node, point to **Add**, and then click **New Item**.

7. In the center pane, click **Site Column**.
8. In the **Name** box, type **LineManager**, and then click **Add**.
9. In the **Elements.xml** file, amend the **Field** element as shown in bold:

```
<Field
 ID="{...}"
 Name="LineManager"
 DisplayName="Line Manager"
 Type="User"
 Required="TRUE"
 Group="Contoso Columns">
</Field>
```

10. Save and close the **Elements.xml** file.
11. In Solution Explorer, right-click the **VacationRequests** project node, point to **Add**, and then click **New Item**.
12. In the center pane, click **Site Column**.
13. In the **Name** box, type **VacationRequestStatus**, and then click **Add**.
14. In the **Elements.xml** file, amend the **Field** element as shown in bold:

```
<Field
 ID="{...}"
 Name="VacationRequestStatus"
 DisplayName="Vacation Request Status"
 Type="Choice"
 Required="TRUE"
 Group="Contoso Columns">
 <CHOICES>
 <CHOICE>Pending</CHOICE>
 <CHOICE>Approved</CHOICE>
 <CHOICE>Rejected</CHOICE>
 <CHOICE>Booked</CHOICE>
 </CHOICES>
 <Default>Pending</Default>
</Field>
```

15. Save and close the **Elements.xml** file.

### ► Task 3: Create the Vacation Request Content Type

1. In Solution Explorer, right-click the **VacationRequests** project node, point to **Add**, and then click **New Item**.
2. In the center pane, click **Content Type**.
3. In the **Name** box, type **VacationRequest**, and then click **Add**.
4. In the **SharePoint Customization Wizard** dialog box, in the **Which base content type should this content type inherit from** list, make sure **Item** is selected, and then click **Finish**.
5. On the **VacationRequest** page, on the **Columns** tab, in the first row, type **Employee**, and then press Tab three times.
6. Type **Contoso Department**, and then press Tab three times.
7. Type **Line Manager**, and then press Tab three times.
8. Type **Start Date**, and then press Tab twice.
9. Press Space to select the **Required** check box, and then press Tab.

10. Type **End Date**, and then press Tab twice.
11. Press Space to select the **Required** check box, and then press Tab.
12. Type **Vacation Request Status**, and then press Tab.
13. On the **Content Type** tab, in the **Content Type Name** box, type **Vacation Request**.
14. In the **Description** box, type **Request a new vacation booking**.
15. In the **Group Name** box, type **Contoso Content Types**.
16. On the **FILE** menu, click **Save All**, and then close the **Vacation Request** page.

#### ► Task 4: Create the Vacation Requests List Instance

1. In Solution Explorer, right-click the **VacationRequests** project node, point to **Add**, and then click **New Item**.
2. In the center pane, click **List**.
3. In the **Name** box, type **VacationRequests**, and then click **Add**.
4. In the **SharePoint Customization Wizard** dialog box, in the **What name do you want to display for your list** box, type **Vacation Requests**, and then click **Finish**.
5. On the **Vacation Requests** page, on the **Columns** tab, click **Content Types**.
6. In the **Content Type Settings** dialog box, select the **Item** row and the **Folder** row, and then press Delete.
7. Type **Vacation Request**, press Tab, and then click **OK**.
8. On the **Vacation Requests** page, ensure that **Required** is selected for every column in the list.
9. Click **Content Types**.
10. In the **Content Type Settings** dialog box, select the **ListFieldsContentType** row, press Delete, and then click **OK**.
11. On the **List** tab, in the **Description** box, type **Use this list to create and manage requests for vacations**.
12. On the **FILE** menu, click **Save All**, and then close the **Vacation Requests** page.

#### ► Task 5: Deploy and Test the Vacation Requests List

1. In Solution Explorer, expand **Features**, right-click **Feature1**, and then click **Rename**.
2. Type **VacationRequests**, and then press Enter.
3. Double-click the **VacationRequests** node you just renamed.
4. On the **Design** tab, in the **Title** box, type **Vacation Requests**.
5. In the **Description** box, type **Deploys site columns, a content type, and a list for managing vacation requests**.
6. On the **FILE** menu, click **Save All**.
7. On the **DEBUG** menu, click **Start Without Debugging**.
8. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$wOrd**.
9. In Internet Explorer, after the page finishes loading, on the Quick Launch navigation menu, click **Vacation Requests**.

10. On the ribbon, on the **ITEMS** tab, on the **New Item** menu, click **Vacation Request**.
11. In the **Title** box, type **Two weeks trekking in the Rockies!**.
12. In the **Employee** box, type **Joel**, and then click **Joel Frauenheim**.
13. In the **Contoso Department** list, click **IT**.
14. In the **Line Manager** box, type **Tihomir**, and then click **Tihomir Sasic**.
15. In the **Start Date** box, select a date within the next twelve months.
16. In the **End Date** box, select a date two weeks from the start date, and then click **Save**.



**Note:** In a real-world deployment, you would use a workflow to manage the **Vacation Request Status** field, rather than allowing users to edit it from the default forms.

17. Close Internet Explorer.
18. Close Visual Studio 2012.

**Results:** After completing this exercise, you should have deployed all the components required to capture and manage vacation requests.

# Lab B: Working with Advanced Features of Content Types

## Exercise 1: Creating an Event Receiver Assembly

### ► Task 1: Create an Empty SharePoint 2013 Project

1. If you are not already logged on, log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
2. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
3. On the Start Page, click **New Project**.
4. In the **New Project** dialog box, in the navigation pane, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
5. In the center pane, click **SharePoint 2013 - Empty Project**.
6. In the **Name** box, type **VacationRequestsEventReceiver**.
7. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
8. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://hr.contoso.com**, and then click **Validate**.
9. In the **Microsoft Visual Studio** dialog box, verify that the dialog reports a successful connection, and then click **OK**.
10. Select **Deploy as a farm solution**, and then click **Finish**.

### ► Task 2: Create the Event Receiver Class

1. In Solution Explorer, right-click the **VacationRequestsEventReceiver** project node, point to **Add**, and then click **Class**.
2. In the **Name** box, type **VacationRequestEventReceiver.cs**, and then click **Add**.



**Note:** You are adding a class manually, rather than adding an **Event Receiver** project item, because you do not want Visual Studio to generate event receiver registration components. The built-in project items enable you to associate event receivers with lists, but they do not enable you to associate event receivers with content types.

3. At the top of the page, after the existing **using** statements, add the following code on a new line:

```
using Microsoft.SharePoint;
```

4. Amend the class declaration as shown by the following code:

```
public class VacationRequestEventReceiver : SPItemEventReceiver
```

5. Within the **VacationRequestEventReceiver** class, type **override ItemAdding**, and then press Enter. Visual Studio should generate the following code:

```
public override void ItemAdding(SPItemEventProperties properties)
{
 base.ItemAdding(properties);
}
```

6. Within the **VacationRequestEventReceiver** class, on a new line after the closing brace of the `ItemAdding` method, type **override ItemUpdated**, and then press Enter. Visual Studio should generate the following code:

```
public override void ItemUpdated(SPItemEventProperties properties)
{
 base.ItemUpdated(properties);
}
```

7. On the **FILE** menu, click **Save All**.

### ► Task 3: Create the ItemAdding Method

1. In the **VacationRequestEventReceiver** class, within the **ItemAdding** method, delete the following code:

```
base.ItemAdding(properties);
```

2. Add the following code:

```
using (var web = properties.OpenWeb())
{
}
```



**Note:** The code for steps 2-6 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabBEx1.txt file.

3. On a new line between the braces, add the following code to get the duration of the vacation request in days:

```
// Get the duration of the vacation in days.
string strStartDate = properties.AfterProperties["StartDate"].ToString();
string strEndDate = properties.AfterProperties["_EndDate"].ToString();
var startDate = DateTime.Parse(strStartDate).Date;
var endDate = DateTime.Parse(strEndDate).Date;
TimeSpan span = endDate - startDate;
int vacationLength = span.Days;
```

4. On a new line, immediately below the code you just added, add the following code to get the **SPUser** instance for the employee:

```
// Get the SPUser instance for the employee.
var employeeFieldVal = new SPFieldUserValue(web,
properties.AfterProperties["Employee"].ToString());
SPUser employee = web.EnsureUser(employeeFieldVal.LookupValue);
```

- On a new line, immediately below the code you just added, add the following code to get the list item for the employee from the Vacation Tracker list:

```
// Get the list item for the employee from the Vacation Tracker list.
var listVT = web.GetList("Lists/VacationTracker");
var query = new SPQuery();
query.Query = String.Format(@"
<Where>
 <Eq>
 <FieldRef Name=""Employee"" LookupId=""TRUE""></FieldRef>
 <Value Type=""Integer"">{0}</Value>
 </Eq>
</Where>
", employee.ID);
var trackerItems = listVT.GetItems(query);
```

- On a new line, immediately below the code you just added, add the following code to block the request if the employee does not have sufficient vacation entitlement remaining:

```
// If you cannot find the employee in the Vacation Tracker list, fail gracefully.
if (trackerItems.Count == 0)
{
 properties.ErrorMessage = string.Format("User {0} not found in Vacation Tracker list.",
employee.Name);
 properties.Status = SPEventReceiverStatus.CancelWithError;
}
else
{
 // Calculate whether the user has sufficient vacation entitlement to cover the request.
 var trackerItem = trackerItems[0];
 int daysRemaining;
 int.TryParse(trackerItem["Days Remaining"].ToString(), out daysRemaining);
 // If the user does not have sufficient vacation entitlement, block the request.
 if (vacationLength > daysRemaining)
 {
 properties.ErrorMessage = string.Format("User {0} only has {1} vacation days
remaining.", employee.Name, daysRemaining);
 properties.Status = SPEventReceiverStatus.CancelWithError;
 }
}
}
```

- On the **FILE** menu, click **Save All**.

#### ► Task 4: Create the ItemUpdated Method

- In the **VacationRequestEventReceiver** class, within the **ItemUpdated** method, delete the following code:

```
base.ItemUpdated(properties);
```

- Add the following code to determine whether the vacation request status has been set to "Approved":

```
var status = properties.AfterProperties["VacationRequestStatus"].ToString();
// No action required unless the status has been set to "Approved".
if (!status.Equals("Approved"))
 return;
```



**Note:** The code for steps 2-8 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabBEx1.txt file

3. On a new line, immediately below the code you just added, add the following code:

```
using (var web = properties.OpenWeb())
{
}
```

4. On a new line between the braces, add the following code to get the duration of the vacation request in days:

```
// Get the duration of the vacation in days.
var item = properties.ListItem;
string strStartDate = item["StartDate"].ToString();
string strEndDate = item["_EndDate"].ToString();
var startDate = DateTime.Parse(strStartDate).Date;
var endDate = DateTime.Parse(strEndDate).Date;
TimeSpan span = endDate - startDate;
int vacationLength = span.Days;
```

5. On a new line, immediately below the code you just added, add the following code to get the **SPUser** instance for the employee:

```
// Get the SPUser instance for the employee.
var employeeFieldVal = new SPFieldUserValue(web,
properties.AfterProperties["Employee"].ToString());
var employee = employeeFieldVal.User;
```

6. On a new line, immediately below the code you just added, add the following code to get the list item for the employee from the Vacation Tracker list:

```
// Get the list item for the employee from the Vacation Tracker list.
var listVT = web.GetList("Lists/VacationTracker");
var query = new SPQuery();
query.Query = String.Format(@"
 <Where>
 <Eq>
 <FieldRef Name=""Employee"" LookupId=""TRUE""></FieldRef>
 <Value Type=""Integer"">{0}</Value>
 </Eq>
 </Where>
", employee.ID);
var trackerItems = listVT.GetItems(query);
SPListItem trackerItem;
if(trackerItems.Count > 0)
{
 trackerItem = trackerItems[0];
}
else return;
```

7. On a new line, immediately below the code you just added, add the following code to update the remaining holiday entitlement in the Vacation Tracker list:

```
// Update the remaining holiday entitlement in the Vacation Tracker list.
int daysRemaining;
int.TryParse(trackerItem["Days Remaining"].ToString(), out daysRemaining);
int balance = daysRemaining - vacationLength;
trackerItem["Days Remaining"] = balance;
trackerItem.Update();
```



- On a new line, immediately below the code you just added, add the following code to change the status of the vacation request to "Booked":

```
// Update the vacation request status to "Booked".
item["Vacation Request Status"] = "Booked";
item.Update();
```

- On the **FILE** menu, click **Save All**.

**Results:** After completing this exercise, you should have created an event receiver assembly.

## Exercise 2: Registering an Event Receiver with a Site Content Type

### ► Task 1: Add a Feature and a Feature Receiver Class

- In Solution Explorer, right-click **Features**, and then click **Add Feature**.
- In Solution Explorer, right-click **Feature1**, and then click **Rename**.
- Type **VacationEntitlementChecker**, and then press Enter.
- On the **VacationEntitlementChecker.feature** page, in the **Title** box, type **Vacation Entitlement Checker**.
- In the **Description** box, type **Checks vacation requests against remaining vacation entitlement, and updates remaining vacation entitlement when a vacation is booked**.
- In the **Scope** dropdown list, click **Site**.
- In Solution Explorer, right-click **VacationEntitlementChecker**, and then click **Add Event Receiver**.
- Within the **VacationEntitlementCheckerEventReceiver** class, select the **FeatureActivated** method (including the braces), and then click **Uncomment the selected lines**.
- Select the **FeatureDeactivating** method (including the braces), and then click **Uncomment the selected lines**.
- On the **FILE** menu, click **Save All**.

### ► Task 2: Add Code to Register the ItemAdded Event Receiver When the Feature Is Activated

- In the **VacationEntitlementCheckerEventReceiver** class, on a new line immediately after the opening brace, add the following code:

```
const string itemAddingName = "Vacation Request ItemAdding";
```

- On a new line, immediately after the code you just added, add the following code:

```
private static void AddItemAddingReceiver(SPFeatureReceiverProperties properties)
{
 using (var site = properties.Feature.Parent as SPSite)
 {
 using (var web = site.RootWeb)
 {
 }
 }
}
```



**Note:** The code for steps 2-3 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabEx2.txt file.

- On a new line between the innermost set of braces, add the following code to add the **ItemAdding** event receiver to the Vacation Request content type:

```
// Get the Vacation Request content type.
var vacationRequestCT = web.ContentTypes["Vacation Request"];
if (vacationRequestCT != null)
{
 SPEventReceiverDefinition erd = vacationRequestCT.EventReceivers.Add();
 erd.Assembly = "[AssemblyNamePlaceholder]";
 erd.Class = "VacationRequestsEventReceiver.VacationRequestEventReceiver";
 erd.Type = SPEventReceiverType.ItemAdding;
 erd.Name = itemAddingName;
 erd.Synchronization = SPEventReceiverSynchronization.Synchronous;
 erd.SequenceNumber = 10050;
 erd.Update();
 vacationRequestCT.Update(true);
}
```

- On the **FILE** menu, click **Save All**, and then on the **BUILD** menu click **Build Solution**. Without closing Visual Studio, switch to the Start screen.
- On the Start screen, type **ISE**, and then click **Windows PowerShell ISE**.
- At the command prompt, type the following command, and then press Enter:

```
$assembly =
"E:\Labfiles\Starter\VacationRequestsEventReceiver\VacationRequestsEventReceiver\bin\Debug\VacationRequestsEventReceiver.dll"
```

- At the command prompt, type the following command, and then press Enter:

```
[System.Reflection.AssemblyName]::GetAssemblyName($assembly).FullName
```

- Select and copy the assembly strong name returned by the command.
- Switch back to Visual Studio 2012.
- In the **AddItemAddingReceiver** method, replace the text **[AssemblyNamePlaceholder]** with the text you copied from the Windows PowerShell editor. The line of code should now resemble the following:

```
erd.Assembly = "VacationRequestsEventReceiver, Version=1.0.0.0, Culture=neutral, PublicKeyToken=xxxxxxxxxxxxxxxx";
```

- In the **FeatureActivated** method, add the following code:

```
AddItemAddingReceiver(properties);
```

- On the **FILE** menu, click **Save All**.

### ► Task 3: Add Code to Register the ItemUpdated Event Receiver When the Feature Is Activated

1. In the **VacationEntitlementCheckerEventReceiver** class, locate the following line of code:

```
const string itemAddingName = "Vacation Request ItemAdding";
```

2. Immediately after this code, on a new line, add the following code:

```
const string itemUpdatedName = "Vacation Request ItemUpdated";
```

3. Within the **VacationEntitlementCheckerEventReceiver** class, but outside any methods, add the following code:

```
private static void AddItemUpdatedReceiver(SPFeatureReceiverProperties properties)
{
 using (var site = properties.Feature.Parent as SPSite)
 {
 using (var web = site.RootWeb)
 {
 }
 }
}
```



**Note:** The code for steps 3-4 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabBEx2.txt file.

4. On a new line between the innermost set of braces, add the following code to add the **ItemUpdated** event receiver to the Vacation Request content type:

```
// Get the Vacation Request content type.
var vacationRequestCT = web.ContentTypes["Vacation Request"];
if (vacationRequestCT != null)
{
 SPEventReceiverDefinition erd = vacationRequestCT.EventReceivers.Add();
 erd.Assembly = "[AssemblyNamePlaceholder]";
 erd.Class = "VacationRequestsEventReceiver.VacationRequestEventReceiver";
 erd.Type = SPEventReceiverType.ItemUpdated;
 erd.Name = itemUpdatedName;
 erd.Synchronization = SPEventReceiverSynchronization.Synchronous;
 erd.SequenceNumber = 10050;
 erd.Update();
 vacationRequestCT.Update(true);
}
```

5. Replace the text **[AssemblyNamePlaceholder]** with the assembly strong name you used in the previous task.
6. In the **FeatureActivated** method, on a new line below the existing code, add the following code:

```
AddItemUpdatedReceiver(properties);
```

7. On the **FILE** menu, click **Save All**.

### ► Task 4: Add Code to Remove the ItemAdding Event Receiver When the Feature Is Deactivated

1. Within the **VacationEntitlementCheckerEventReceiver** class, but outside any methods, add the following code:

```
private static void RemoveItemAddingReceiver(SPFeatureReceiverProperties properties)
{
 using (var site = properties.Feature.Parent as SPSite)
 {
 using (var web = site.RootWeb)
 {
 }
 }
 }
}
```



**Note:** The code for steps 1-2 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabBEx2.txt file.

2. On a new line between the innermost set of braces, add the following code to remove the **ItemAdding** event receiver from the Vacation Request content type:

```
var vacationRequestCT = web.ContentTypes["Vacation Request"];
if (vacationRequestCT != null)
{
 Guid existingEventReceiverId = Guid.Empty;
 foreach (SPEventReceiverDefinition eventReceiver in vacationRequestCT.EventReceivers)
 {
 if (String.Equals(eventReceiver.Name, itemAddingName))
 {
 existingEventReceiverId = eventReceiver.Id;
 }
 }
 if (!existingEventReceiverId.Equals(Guid.Empty))
 {
 vacationRequestCT.EventReceivers[existingEventReceiverId].Delete();
 vacationRequestCT.Update(true);
 }
}
```

3. In the **FeatureDeactivating** method, add the following code:

```
RemoveItemAddingReceiver(properties);
```

4. On the **FILE** menu, click **Save All**.

## ► Task 5: Add Code to Remove the ItemUpdated Event Receiver When the Feature Is Deactivated

1. Within the **VacationEntitlementCheckerEventReceiver** class, but outside any methods, add the following code:

```
private static void RemoveItemUpdatedReceiver(SPFeatureReceiverProperties properties)
{
 using (var site = properties.Feature.Parent as SPSite)
 {
 using (var web = site.RootWeb)
 {
 }
 }
 }
}
```



**Note:** The code for steps 1-2 is available in the E:\Labfiles\Starter\LabCodeSnippets-LabBEx2.txt file.

2. On a new line between the innermost set of braces, add the following code to remove the **ItemAdding** event receiver from the Vacation Request content type:

```
var vacationRequestCT = web.ContentTypes["Vacation Request"];
if (vacationRequestCT != null)
{
 Guid existingEventReceiverId = Guid.Empty;
 foreach (SPEventReceiverDefinition eventReceiver in vacationRequestCT.EventReceivers)
 {
 if (String.Equals(eventReceiver.Name, itemUpdatedName))
 {
 existingEventReceiverId = eventReceiver.Id;
 }
 }
 if (!existingEventReceiverId.Equals(Guid.Empty))
 {
 vacationRequestCT.EventReceivers[existingEventReceiverId].Delete();
 vacationRequestCT.Update(true);
 }
}
```

3. In the **FeatureDeactivating** method, add the following code:

```
RemoveItemUpdatedReceiver(properties);
```

4. On the **FILE** menu, click **Save All**.
5. On the **BUILD** menu, click **Build Solution**.

## ► Task 6: Test the ItemAdding Event Receiver

1. On the **DEBUG** menu, click **Start Without Debugging**.
2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. In Internet Explorer, after the page finishes loading, on the Quick Launch navigation menu, click **Vacation Tracker**.
4. Notice that the user **Ankur Chavda** has eight days of vacation remaining.
5. On the Quick Launch navigation menu, click **Vacation Requests**.

6. On the **Vacation Requests** page, click **new item**.
7. In the **Title** box, type **Two weeks in the Caribbean!**
8. In the **Employee** box, type **Ankur Chavda**, and then press Enter.
9. In the **Department** drop-down list, click **Legal**.
10. In the **Line Manager** box, type **Dominik Dubicki**, and then press Enter.
11. In the **Start Date** box, select a date within the next 12 months.
12. In the **End Date** box, select a date two weeks after the start date.
13. Click **Save**.
14. Verify that you are prevented from submitting the vacation request with the message **User Ankur Chavda only has 8 vacation days remaining**.
15. In the **Title** box, type **One week in the Caribbean!**
16. In the **End Date** box, select a date one week after the start date, and then click **Save**.
17. Verify that the vacation request was created successfully.

► **Task 7: Test the ItemUpdated Event Receiver**

1. In the **Vacation Requests** list, select the **One week in the Caribbean!** row.
2. On the ribbon, on the **ITEMS** tab, click **Edit Item**.
3. In the **Vacation Request Status** drop-down list, click **Approved**, and then click **Save**.
4. Verify that the vacation request status of the list item is changed to **Booked**.
5. On the Quick Launch navigation menu, click **Vacation Tracker**.
6. Verify that the list now shows **Ankur Chavda** as having one day of vacation remaining.
7. Close Internet Explorer.
8. Close Visual Studio.
9. Close Windows Powershell ISE.

**Results:** After completing this exercise, you should have used a feature receiver to add an event receiver to a site content type.

## Module 13: Managing Custom Components and Site Life Cycles

# Lab: Managing Custom Components and Site Life Cycles

### Exercise 1: Creating a Site Definition

#### ► Task 1: Create a New Solution

1. On the Start screen, type **Visual Studio**, and then click **Visual Studio 2012**.
2. In Visual Studio, on the **FILE** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, expand **Visual C#**, expand **Office/SharePoint**, click **SharePoint Solutions**, and then click **SharePoint 2013 - Empty Project**.
4. In the **Name** box, type **SalesSite**.
5. In the **Location** box, type **E:\Labfiles\Starter\**, and then click **OK**.
6. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://dev.contoso.com**, click **Deploy as farm solution**, and then click **Finish**.

#### ► Task 2: Add a Site Definition to the Solution

1. In Solution Explorer, right-click **SalesSite**, point to **Add**, and then click **New Item**.
2. In the **Add New Item - SalesSite** dialog box, scroll down and click **Site Definition (Farm Solution Only)**.
3. In the **Name** box, type **SalesSiteDefinition**, and then click **Add**.
4. In Solution Explorer, double-click **webtemp\_SalesSiteDefinition.xml**.
5. In the **webTemp\_SalesSiteDefinition.xml** file, replace the value of the **ID** attribute of the **Template** element with **12000**.
6. Replace the value of the **Title** attribute of the **Configuration** element with **Sales Site**.
7. Replace the value of the **Description** attribute of the **Configuration** element with **A site for Contoso sales teams**.
8. Replace the value of the **DisplayCategory** attribute of the **Configuration** element with **Contoso Sites**.
9. Click **Save All**.

#### ► Task 3: Customize the Site Definition

1. In Solution Explorer, double-click **onet.xml**.
2. Add the following code shown in bold to the **NavBars** element.

```
<NavBars>
 <NavBar ID="1002" Name="$Resources:core,category_Top;">
 <NavBarLink Name="Global Sales Site" Url="http://sales.contoso.com" />
 <NavBarLink Name="Contoso Home Page" Url="http://www.contoso.com" />
 </NavBar>
</NavBars>
```

3. Locate the **<Lists />** element.
4. Replace the **<Lists />** element with the following code:

```
<Lists>
 <List Type="101" Url="Lists/SalesDocuments" Title="Sales Documents" Description="Use this
 library to store documents related to specific sales opportunities." />
 <List Type="101" Url="Lists/SalesResources" Title="Sales Resources" Description="Use this
 library to store generic documents that all members of the sales team can use to support any
 sales opportunity." />
</Lists>
```

5. Click **Save All**.

#### ► Task 4: Modify the Site Home Page

1. In Solution Explorer, double-click **default.aspx**.
2. Replace the **Welcome to the custom site SalesSiteDefinition** text with **Welcome to the Sales Portal**.
3. On a new line after the closing **h1** element, add the following code shown in bold.

```
</h1>
<p>Use the Sales Documents library to store
documents related to a specific sales opportunity.</p>
<p>Use the Sales Resources library to store generic
documents to support any sales opportunity.</p>
<WebPartPages:WebPartZone runat="server" ID="Center" Title="Center" />
</asp:Content>
```

4. Click **Save All**.

#### ► Task 5: Test the Site Definition

1. In Solution Explorer, click **SalesSite**.
2. In the **Properties** pane, in the **Include Assembly In Package** row, on the drop-down list, click **False**.
3. Click **Save All**.
4. On the **DEBUG** menu, click **Start without debugging**.



**Note:** The first time that you run the app, your browser may display an error due to a timeout issue. You can ignore this error and continue with step 5.

5. In Internet Explorer, browse to **http://pain-relief.contoso.com**.
6. On the **Pain Relief Team Site** page, click **Site Contents**.
7. On the **Site Contents** page, click **new subsite**.
8. On the **New SharePoint Site** page, in the **Title** box, type **Sales Site**.
9. In the **URL name** box, type **sales**.
10. Under **Template Selection**, on the **Contoso Sites** tab, click **Sales Site**, and then click **Create**.
11. On the **Sales** site, verify that the home page contains links to the **Sales Documents** and **Sales Resources** document libraries.
12. Click **Site Contents**.



13. On the **Site Contents** page, verify that the site contains two document libraries, named **Sales Documents**, and **Sales Resources**.
14. Close Internet Explorer.

**Results:** After completing this exercise, you should have created a site definition.

## Exercise 2: Creating a List Definition

### ► Task 1: Add a List Definition and Instance

1. In Visual Studio, in Solution Explorer, right-click **SalesSite**, point to **Add**, and then click **New Item**.
2. In the **Add New Item - SalesSite** dialog box, click **List**, in the **Name** box, type **SalesLeads**, and then click **Add**.
3. In the **SharePoint Customization Wizard** dialog box, in the **What name do you want to display for your list** box, type **Sales Leads**, and then click **Finish**.
4. On the **Columns** tab, in a new row, in the **Column Display Name** column, type **Customer**.
5. In a new row, in the **Column Display Name** column, type **Estimated Value**, and then in that row, in the **Type** column, in the **Type** list, click **Currency**.
6. In a new row, in the **Column Display Name** column, type **Expected Completion**, in the **Type** column, in the **Type** list, click **Date and Time**.
7. In a new row, in the **Column Display Name** column, type **Sales Person**, in the **Type** column, in the **Type** list, click **Person or Group**.
8. In a new row, in the **Column Display Name** column, type **Quote Number**.
9. In a new row, in the **Column Display Name** column, type **Lead Notes**, in the **Type** column, in the **Type** list, click **Multiple Lines of Text**.
10. On the **List** tab, in the **Title** box, type **Sales Leads**.
11. In the **List URL** box, type **Lists/SalesLeads**.
12. In the **Description** box, type **Use this list to record and track sales opportunities**.
13. Click **Save All**.

### ► Task 2: Modify the List Deployment Feature

1. In Solution Explorer, double-click **Feature1**.
2. In the Feature designer, in the **Title** box, type **Sales Leads List**.
3. In Solution Explorer, right-click **Feature1**, and then click **Rename**.
4. Type **SalesLeadsList**, and then press Enter.
5. Click **Save All**.

### ► Task 3: Add the SalesLeadsList Feature to the Site Definition

1. On the **Manifest** tab, in the **Preview of Packaged Manifest** box, make a note of the value of the **ID** attribute of the **Feature** element.
2. In Solution Explorer, double-click **onet.xml**.

3. Add the following code shown in bold. Replace *IdPlaceholder* with the Feature ID that you noted in step one of this task.

```
<WebFeatures>
 <Feature Name="Sales Leads List" ID="IdPlaceholder" />
</WebFeatures>
```

4. Click **Save All**.

► **Task 4: Test the Sales Leads List and Sales Site Definition**

1. On the **DEBUG** menu, click **Start without debugging**.
2. In Internet Explorer, browse to **http://anti-virals.contoso.com**.
3. On the **Anti Virals Team Site** page, click **Site Contents**.
4. On the **Site Contents** page, click **new subsite**.
5. On the **New SharePoint Site** page, in the **Title** box, type **Sales Site**.
6. In the **URL name** box, type **sales**.
7. Under **Template Selection**, on the **Contoso Sites** tab, click **Sales Site**, and then click **Create**.
8. On the **Sales** site, verify that the home page contains links to the **Sales Documents** and **Sales Resources** document libraries.
9. Click **Site Contents**.
10. On the **Site Contents** page, verify that the site contains two document libraries, named **Sales Documents**, and **Sales Resources**, and a list named **Sales Leads**.
11. Click **Sales Leads**.
12. On the **Sales Leads** page, click **new item**.
13. On the **Sales Leads - New Item** page, in the **Title** box, type **Antiviral Trial**.
14. In the **Customer** box, type **Fabrikam**.
15. In the **Estimated Value** box, type **5000**.
16. In the **Estimated Completion** box, select a date two weeks from today.
17. In the **Sales Person** box, type **Dan Park**.
18. In the **Quote Number** box, type **Q1001**.
19. In the **Lead Notes** box, type **The customer showed an interest in trialing our antiviral products**, and then click **Save**.
20. Verify that the new item appears in the list, and that all seven fields are displayed.
21. Close Internet Explorer.
22. In Visual Studio, on the **FILE** menu, click **Close Solution**.

**Results:** After completing this exercise, you should have created a list definition, created a list instance, and modified a site definition to include a Feature.

## Exercise 3: Developing an Event Receiver

### ► Task 1: Create a New Solution

1. In Visual Studio, on the **FILE** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, click **SharePoint 2013 - Empty Project**.
3. In the **Name** box, type **SiteArchiveFeature**.
4. In the **Location** box, type **E:\Labfiles\Starter\**, and then click **OK**.
5. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://dev.contoso.com**, click **Deploy as farm solution**, and then click **Finish**.

### ► Task 2: Add an Event Receiver to the Solution to Back Up a Site Collection

1. In Solution Explorer, right-click **SiteArchiveFeature**, point to **Add**, and then click **New Item**.
2. In the **Add New Item - SiteArchiveFeature** dialog box, click **Event Receiver**, in the **Name** box type **SalesWebDeletingReceiver**, and then click **Add**.
3. In the **SharePoint Customization Wizard** dialog box, in the **What type of event receiver do you want** list, click **Web Events**.
4. In the **Handle the following events** list, select **A site is being deleted**, and then click **Finish**.
5. In the **SalesWebDeletingReceiver.cs** code file, remove the following line of code from the **WebDeleting** method.

```
base.WebDeleting(Properties);
```

6. Add the code shown in bold to the **WebDeleting** method.

```
public override void WebDeleting(SPWebEventProperties properties)
{
 using (var site = new SPSite(properties.SiteId))
 {
 string backupLocation = @"E:\Labfiles\Starter\Backup.backup";
 site.WebApplication.Sites.Backup(site.Url, backupLocation, true);
 }
}
```

7. On the **BUILD** menu, click **Build Solution**, and verify that the solution builds successfully.

### ► Task 3: Modify the Event Receiver Deployment Feature

1. In Solution Explorer, double-click **Feature1**.
2. In the Feature designer, in the **Title** box, type **Web Deleting Event Receiver**.
3. In Solution Explorer, right-click **Feature1**, and then click **Rename**.
4. Type **WebDeletingEventReceiver**, and then press Enter.
5. Click **Save All**.

### ► Task 4: Add a Feature to Staple the Event Receiver Feature to the Sales Site Definition

1. In Solution Explorer, right-click **Features**, and then click **Add Feature**.
2. In the Feature designer, in the **Title** box, type **Sales Site Event Receiver Feature Stapler**.
3. In the **Scope** list, click **Farm**.

4. In Solution Explorer, right-click **Feature1**, and then click **Rename**.
5. Type **SalesSiteEventReceiverFeatureStapler**, and then press Enter.
6. In Solution Explorer, double-click **WebDeletingEventReceiver**.
7. In the Feature designer, on the **Manifest** tab, in the **Preview of Packaged Manifest** box, make a note of the value of the **ID** attribute of the **Feature** element.
8. Right-click **SiteArchiveFeature**, point to **Add**, and then click **New Item**.
9. In the **Add New Item - SiteArchiveFeature** dialog box, click **Empty Element**, in the **Name** box, type **FeatureStapler**, and then click **Add**.
10. Add the code shown in bold to the Elements.xml code file. Replace *IdPlaceholder* with the Feature ID that you noted in step seven of this task.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <FeatureSiteTemplateAssociation Id="IdPlaceholder"
 TemplateName="SalesSiteDefinition#0" />
</Elements>
```

11. Click **Save All**.
12. In Solution Explorer, double-click **WebDeletingEventReceiver**, then click **Design Tab**.
13. In the **Items in the Feature** box, click **FeatureStapler**, and then click the left chevron button (<).
14. In Solution Explorer, double-click **SalesSiteEventReceiverFeatureStapler**.
15. In the **Items in the Solution** box, click **FeatureStapler**, and then click the right chevron button (>).
16. Click **Save All**.

#### ► Task 5: Test the Feature Stapler and Event Receiver

1. On the **DEBUG** menu, click **Start without debugging**.
2. In Internet Explorer, browse to **http://cosmetics.contoso.com**.
3. On the **Cosmetics Team** site, click **Site Contents**.
4. On the **Site Contents** page, click **new subsite**.
5. On the **New SharePoint Site** page, in the **Title** box, type **Sales Site**.
6. In the **URL name** box, type **sales**.
7. Under **Template Selection**, on the **Contoso Sites** tab, click **Sales Site**, and then click **Create**.
8. On the **Settings** menu, click **Site settings**.
9. On the **Site Settings** page, under **Site Actions**, click **Manage site features**.
10. On the **Site Features** page, verify that the **Web Deleting Event Receiver** has a status of **Active**.
11. Click **Site Settings**.
12. On the **Site Settings** page, under **Site Actions**, click **Delete this site**.
13. On the **Delete This Site** page, click **Delete**.
14. In the **Message from webpage** dialog box, click **OK**.
15. Close Internet Explorer.
16. On the taskbar, click **File Explorer**.

17. In File Explorer, browse to **E:\Labfiles\Starter**. Verify that a backup file named **Backup.backup** exists in the folder.
18. Close File Explorer.
19. Close Visual Studio.

**Results:** After completing this exercise, you should have developed an event receiver that responds to a site deleting event. You should also have used Feature stapling to add a Feature to an existing site definition.



## Module 14: Customizing User Interface Elements

# Lab A: Using the Edit Control Block to Launch an App

### Exercise 1: Configuring an App to Display Customer Orders

#### ► Task 1: Set Permissions for the App to Read the List

1. Start the 20488B-LON-SP-14 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, click **Visual Studio 2012**.
4. On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
5. In the **Open Project** dialog box, browse to the **E:\Labfiles\Starter\CustomerOrder** folder, click **CustomerOrder.sln**, and then click **Open**.
6. In Solution Explorer, expand **CustomerOrder**, and then double-click **AppManifest.xml**.
7. In the main window, click **Permissions**.
8. In the **Scope** column, click the blank row, and then in the drop-down list, click **List**.
9. In the **Permission** column, click the empty cell, and then in the drop-down list, click **FullControl**.
10. On the **FILE** menu, click **Save AppManifest.xml**.
11. On the Start screen, click **Internet Explorer**.
12. In the address bar, type **http://team.contoso.com**, and then press Enter.
13. On the Quick Launch toolbar, click **Customers** and review the customer information that is displayed.
14. On the Quick Launch toolbar, click **Orders** and review the order information that is displayed. Note that the data in the Customer column links through to display information from the Customers table.
15. Close Internet Explorer.

#### ► Task 2: Create a CAML Query to Find the Customer ID

1. In Visual Studio, in Solution Explorer, expand **Scripts**, and then double-click **App.js**.
2. In the code editor, click at the end of the **TODO: Task 1. Create CAML query to retrieve the orders list for a given customer ID** comment, press Enter, and then type the following code:

```
var query = new SP.CamlQuery();
query.set_viewXml("<View><Query><Where><Contains><FieldRef Name='Customer_x003a_ID' /><Value Type='Text'>" + customerID + "</Value></Contains></Where></Query></View>");
list = targetList.getItems(query);
```



**Note:** Because the colon in the field name Customer:ID is a special character, in your code you must use the internal name which replaces the colon with `_x003a_`.

3. Click at the end of the **TODO: Task 2. Submit query results** comment, press Enter, and then type the following code:

```
currentContext.load(list);
currentContext.executeQueryAsync(Function.createDelegate(this, onSuccess),
Function.createDelegate(this, onFail));
```

### ► Task 3: Test the App

1. On the **DEBUG** menu, click **Start Debugging**.
2. In Internet Explorer, on the **Do you trust CustomerOrder?** page, in the **Let it have full control of the list** list, click **Orders**, and then click **Trust It**.
3. Click **Contoso Team Site**.
4. On the Quick Launch toolbar, click **Site Contents**.
5. On the **Site Contents** page, point to **CustomerOrder**, click the ellipsis, and then click **PERMISSIONS**.
6. On the **"CustomerOrder" uses the following permissions** page, in the **Let it have full control of the list** list, click **Customers**, and then in the **If there's wrong something with the app's permissions, click here to trust it again** text, click **here**.
7. On the Quick Launch toolbar, click **CustomerOrder**.
8. In the **Customer ID** box, type **4**, and then click **Search**.
9. Verify that four orders are displayed for Suroor Fatima.
10. In the **Customer ID** box, type **2**, and then click **Search**.
11. Verify that three orders are displayed for Eli Bowen.
12. In the **Customer ID** box, type **1**, and then click **Search**.
13. Verify that no orders are displayed.
14. Close Internet Explorer.
15. In Visual Studio, on the **FILE** menu, click **Close Solution**.

**Results:** After completing this exercise, you should have implemented and tested the CustomerOrder app.

## Exercise 2: Use a Custom Action to Launch an App

### ► Task 1: Get the SPListItemID Token from the Query String

1. In Visual Studio, on the **FILE** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, browse to **E:\Labfiles\Starter\EditControlBlock**, click **EditControlBlock.sln**, and then click **Open**.
3. In Solution Explorer, expand **EditControlBlock**, expand **Scripts**, and then double-click **App.js**.



- In the code editor, click at the end of the **TODO: Task 1. Get the SPListItemID token from the query string** comment, press Enter, and then type the following code:

```
var customerID = getQueryStringParams("SPListItemId");
getList(customerID);
```

- Locate the **TODO: Task 2. Modify the getList function and the caml query** comment.
- Modify the **getList()** function signature to look like the following:

```
function getList(customerID)
```

- In the **getList(customerID)** function, remove the following line of code:

```
var customerID = $("#CustomerID").val();
```

### ► Task 2: Add an ECB to the Customers List

- In Solution Explorer, right click **EditControlBlock**, point to **Add**, and then click **New Item**.
- In the **Add New Item – EditControlBlock** dialog box, in the templates list, click **Menu Item Custom Action**, in the **Name** box, type **CustomerOrderECB**, and then click **Add**.
- In the **Create Custom Action for Menu Item** dialog box, under **Where do you want to expose the custom action?**, click **Host Web**.
- In the **Which particular item is the custom action scoped to?** drop-down list, click **Contacts**, and then click **Next**.
- In the **What is the text on the menu item?** box, type **Customer Orders**, and then click **Finish**.

### ► Task 3: Launch the App in a Dialog Box

- In Solution Explorer, expand **CustomerOrderECB**, and then double-click **Elements.xml**.
- In the XML window, at the end of the **CustomAction** element, click between the " and > characters, press Enter, and then type following XML:

```
HostWebDialog="true"
HostWebDialogHeight="650"
HostWebDialogWidth="650"
```

- In Solution Explorer, expand **Pages**, and then double-click **Default.aspx**.
- In the code editor, click at the end of the **<asp:Content ID="Content3" ContentPlaceHolderID="PlaceHolderMain" runat="server">** markup, press Enter, and then type the following:

```
<WebPartPages:AllowFraming ID="AllowFraming" runat="server" />
```

- In the code editor, remove the following markup.

```
Customer ID
<input id="CustomerID"/>
<button type="button" onclick="getList()">Search</button>
```

### ► Task 4: Deploy the App

- On the **Build** menu, click **Deploy Solution**.
- In Internet Explorer, on the **Do you trust EditControlBlock?** page, in the **Let it have full control of the list** drop-down list, click **Orders**, and then click **Trust It**.

3. Click **Contoso Team Site**.
4. On the Quick Launch toolbar, click **Site Contents**.
5. On the **Site Contents** page, point to **EditControlBlock**, click the ellipsis, and then click **PERMISSIONS**.
6. On the "**EditControlBlock**" **uses the following permissions** page, in the **Let it have full control of the list** drop-down list, click **Customers**, and then in the **If there's something wrong with the app's permissions, click here to trust it again** text, click **here**.
7. On the Quick Launch toolbar, click **Customers**.
8. In the **Customer** list, click the ellipsis for **Fatima**, and then click **Customer Orders**.
9. In the **Customer Orders** window, verify that four orders are displayed.
10. Close the Customer Orders window.
11. In the customer list, click the ellipsis for **Bowen**, and then click **Customer Orders**.
12. In the **Customer Orders** window, verify that three orders are displayed.
13. Close the Customer Orders window.
14. Close Internet Explorer, and then close Visual Studio.

**Results:** After completing this exercise, you should have added a custom action to the project, written code to use the custom action, and deployed the app.

# Lab B: Using jQuery to Customize the SharePoint List User Interface

## Exercise 1: Creating a Custom List View

### ► Task 1: Create the Custom List View by Using jQuery UI

1. Start the 20488B-LON-SP-14 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, click **Visual Studio 2012**.
4. On the **FILE** menu, point to **Open**, and then click **Project/Solution**.
5. In the **Open Project** dialog box, navigate to the **E:\Labfiles\Starter\CustomerInfo** folder, click **CustomerInfo.sln**, and then click **Open**.
6. On the **DEBUG** menu, click **Start Debugging**.
7. In Internet Explorer, review the customer information, and then close Internet Explorer.
8. In Solution Explorer, right-click **Scripts**, point to **Add**, and then click **Existing Item**.
9. In the **Add Existing Item – CustomerInfo** dialog box, double-click **Scripts**, click **jquery-1.7.1.js**, hold the CTRL key, click **jquery-ui.js**, and then click **Add**.
10. Right-click **CustomerInfo**, point to **Add**, and then click **New Item**.
11. In the **Add New Item – CustomerInfo** dialog box, in the **Template** list, click **Module**, in the **Name** box, type **SiteAssets**, and then click **Add**.
12. Right-click **SiteAssets**, point to **Add**, and then click **Existing Item**.
13. In the **Add Existing Item – CustomerInfo** dialog box, navigate to **E:\Labfiles\Starter\SiteAssets**, click **jquery-ui.css**, and then click **Add**.
14. Right-click **SiteAssets**, point to **Add**, and then click **New Folder**.
15. Type **Images**, and then press Enter.
16. Right-click **Images**, point to **Add**, and then click **Existing Item**.
17. In the **Add Existing Item – CustomerInfo** dialog box, double-click **images**, click **animated-overlay.gif**, hold the Shift key, click **ui-icons\_ffffff\_256x240.png**, and then click **Add**.
18. Right-click **CustomerInfo**, point to **Add**, and then click **New Item**.
19. In the **Add New Item – Customer Info** dialog box, in the **Template** list, click **Module**, in the **Name** box, type **AddjQuery**, and then click **Add**.
20. In **Elements.xml**, in the **Elements** element, add the following XML:

```
<CustomAction
 ScriptSrc=~Site/Scripts/jquery-1.7.1.js"
 Location="ScriptLink"
 Sequence="10">
</CustomAction>
<CustomAction
 ScriptSrc=~Site/Scripts/jquery-ui.js"
 Location="ScriptLink"
 Sequence="20">
</CustomAction>
```

21. On the **FILE** menu, click **Save AdjQuery\Elements.xml**.
22. Right-click **Scripts**, point to **Add**, and then click **New Item**.
23. In the **Add New Item – CustomerInfo** dialog box, in the left-hand pane, click **Web**, in the Templates list, click **JavaScript File**, in the **Name** box, type **CustomizeView.js**, and then click **Add**.
24. In **CustomizeView.js**, add the following code to process the accordion item:

```

window.Cust = window.Cust || {};
window.Cust.accordionItem = {
 customItemHtml: function (ctx) {
 var accordionItemHtml = "<h3>" + ctx.CurrentItem.CustomerName + "</h3>";
 accordionItemHtml += "<div>" + ctx.CurrentItem.Details + "</div>";
 return accordionItemHtml;
 }
};

```

25. In **CustomizeView.js**, add the following code to create the new view:

```

(function () {
 // Initialize the variable that store the objects.
 var overrideCtx = {};
 overrideCtx.Templates = {};
 overrideCtx.Templates.Header = "<div id=\"\"accordion\"\">";
 overrideCtx.Templates.Item = window.Cust.accordionItem.customItemHtml;
 overrideCtx.Templates.Footer = "</div>";
 overrideCtx.BaseViewID = 1;
 overrideCtx.ListTemplateType = 10000;
 // Register the template overrides.
 SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
})();

```

26. In **CustomizeView.js**, add the following code to create the accordion:

```

$(document).ready(function () {
 $("#accordion").find("#scriptBodyWPQ1").remove();
 $("#accordion").accordion({ heightStyle: "content" });
});

```

27. In Solution Explorer, expand **CustomerInformation**, and then double-click **Schema.xml**.
28. Locate the **View** element with a **DefaultView** attribute of **TRUE**.
29. Change the contents of the **JSLink** child element to **~site/Scripts/CustomizeView.js**.
30. On the **DEBUG** menu, click **Start Debugging**.
31. When Internet Explorer loads, verify that the list now displays using the accordion theme.
32. Click **Sara Davis** to expand her information.
33. Close Internet Explorer.

### ► Task 2: Apply a Custom Style Sheet from jQuery UI

1. In Visual Studio, in Solution Explorer, expand **Scripts**, and then double-click **CustomizeView.js**.
2. Scroll to the end of the file and click on a blank line under the existing code.
3. On the **EDIT** menu, click **Insert File As Text**.
4. In the **Insert File** dialog box, navigate to the **E:\Labfiles\Starter\CustomerInfo** folder, in the **File Type** drop-down box, click **Text Files (\*.txt)**, click **CodeSnippets.txt**, and then click **Open**.

5. At the beginning of the anonymous function, add the following code:

```
var appWebUrl = decodeURIComponent(getQueryStringParameter("SPAppWebUrl"));
var scriptFolder = appWebUrl + "/SiteAssets/";
loadcssfile(scriptFolder + "jquery-ui.css", "css");
```

6. On the **BUILD** menu, click **Retract** and wait for the retraction to succeed.
7. On the **DEBUG** menu, click **Start Debugging**.
8. When Internet Explorer loads, verify that the list now displays using the accordion style sheet.
9. Click **Sara Davis** to expand her information.
10. Close Internet Explorer.
11. In Visual Studio, on the **BUILD** menu, click **Deploy CustomerInfo**.
12. After the solution successfully deploys, look in the Output window and make a note of the URL where the app was installed.
13. On the Start screen, click **Internet Explorer**.
14. In Internet Explorer, in the Address bar, type the URL where your app was installed, and then press Enter.
15. Verify that the Customer Information list displays using the accordion theme and style sheet.
16. Close Internet Explorer.
17. In Visual Studio, on the **FILE** menu, click **Exit**.

**Results:** After completing this exercise, you should have implemented and tested a custom list view.



## Module 15: Working with Branding and Navigation

# Lab A: Branding and Designing Publishing Sites

### Exercise 1: Creating SharePoint Master Pages

#### ► Task 1: Map the Master Page Gallery as a Network Drive

1. Start the 20488B-LON-SP-15 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **File**, and then click **File Explorer**.
4. In File Explorer, right-click **Computer**, and then click **Map network drive**.
5. In the **Map Network Drive** dialog box, in the **Folder** box, type **http://publishing.contoso.com/\_catalogs/masterpage**, and then click **Finish**.
6. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
7. Verify that the File Explorer window displays the contents of the master page gallery, and then close the File Explorer window.

#### ► Task 2: Create a New Minimal Master Page

1. On the Start screen, click Internet Explorer, and browse to **http://publishing.contoso.com**.
2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. When the page loads, on the **Settings** menu, click **Design Manager**.
4. On the **Design Manager: Welcome** page, on the list of options on the left side of the page, click **Edit Master Pages**.
5. On the **Design Manager: Edit Master Pages** page, click **Create a minimal master page**.
6. In the **Create a Master Page** dialog box, in the **Name** box, type **ContosoPrototype1**, and then click **OK**.
7. On the **Design Manager: Edit Master Pages** page, click **ContosoPrototype1**.
8. On the **Change Preview Page** drop-down menu, click **Select Existing**.
9. In the **Select Existing Page for Previewing** dialog box, click **URL**.
10. In the **URL** box, type **http://publishing.contoso.com/pages/default.aspx**, and then click **OK**.



**Note:** The preview now shows how the Default.aspx page would look if it was rendered in your new master page. As you edit the master page, you can return to this preview page to preview your changes.

#### ► Task 3: Add HTML to the Master Page

1. Without closing Internet Explorer, on the taskbar, click **File Explorer**.
2. In File Explorer, under **Computer**, click **masterpage**.

3. Right-click **ContosoPrototype1.html**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.

4. In Visual Studio, in the **ContosoPrototype1.html** file, locate the following line of code:

```
<div id="s4-workspace">
```

5. On a new line immediately before the line of code you just located, add the following code:

```
<header>
 <div class="contentContainer">
 </div>
</header>
```



**Note:** When you type a **class** attribute, Visual Studio may display dialog boxes warning that it is unable to edit certain built-in CSS files. You are not trying to edit the CSS files, so you can safely close these dialog boxes and continue.

6. Locate the following line of code:

```
<div id="s4-bodyContainer">
```

7. On a new line immediately after the line of code you just located, add the following code:

```
<section id="content">
```

8. Locate the following code:

```
<!--CE: End PlaceholderMain Snippet-->
</div>
```

9. On a new line immediately after the **</div>** element in the code you just located, add the following code:

```
</section>
```

10. On a new line immediately before the closing **</body>** tag, add the following code:

```
<footer>
 <div class="contentContainer">
 </div>
</footer>
```

11. On the **FILE** menu, click **Save All**.

#### ► Task 4: Add CSS to the Master Page

1. On the taskbar, click **File Explorer**.
2. Browse to **E:\Labfiles\Starter**, right-click **ContosoLayout.css**, and then click **Copy**.
3. Under **Computer**, click **masterpage**.
4. Right-click any white space within the list of files, and then click **Paste**.
5. Switch back to Visual Studio.



6. Within the **head** element, locate the following line of code:

```
<!--ME:</SharePoint:CssLink>-->
```

7. On a new line immediately after the line of code you just located, add the following code:

```
<link href="ContosoLayout.css" rel="stylesheet" type="text/css" />
```



**Note:** SharePoint uses the **CssLink** control to load default CSS style sheets. By adding your CSS links after the **CssLink** control, you ensure that your styles can override the default styles. Similarly, by adding your CSS links before the **PlaceHolderAdditionalPageHead** control, you ensure that page layouts can insert CSS links after your master page links and thereby override your styles.

8. On the **FILE** menu, click **Save All**.

### ► Task 5: Preview Your Master Page Changes

1. Switch back to Internet Explorer.
2. On the preview page, reload the page.
3. Verify that the preview page reflects the changes you have made to the **ContosoPrototype1** master page.
4. Leave Internet Explorer open. You will continue to work with Design Manager in the next exercise.

**Results:** After completing this exercise, you should have created a new SharePoint master page, added design elements, and previewed your design in Design Manager.

## Exercise 2: Building Master Page Functionality

### ► Task 1: Add the Site Logo to the Master Page

1. In Internet Explorer, at the top of the preview page, click **Snippets**.
2. On the ribbon, in the **Administration** group, click **Site Logo**.
3. Under the **HTML Snippet** box, click **Copy to Clipboard**.
4. If the **Internet Explorer** dialog box appears, click **Allow access**.
5. Switch back to Visual Studio.
6. In the **header** element, locate the following line of code:

```
<div class="contentContainer">
```

7. On a new line immediately below the line of code you just located, on the **EDIT** menu, click **Paste**.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

8. Locate the following line of code:

```
<div data-name="SiteLogo">
```

9. Amend the **div** tag to include an **id** attribute value of **logo**, as follows:

```
<div id="logo" data-name="SiteLogo">
```



**Note:** The custom CSS file, ContosoLayout.css, uses the **logo** ID to position the site logo.

10. On the **FILE** menu, click **Save All**.
11. Switch back to Internet Explorer.
12. On the preview page, reload the page.
13. Verify that the preview page now includes the site logo, positioned on the left of the site header.
14. Open a new Internet Explorer tab, and browse to **http://publishing.contoso.com**.
15. On the **Settings** menu, click **Site settings**.
16. On the **Site Settings** page, under **Look and Feel**, click **Title, description, and logo**.
17. Under **Insert Logo**, click **FROM COMPUTER**.
18. In the **Add a document** dialog box, click **Browse**.
19. In the **Choose File to Upload** dialog box, browse to **E:\Labfiles\Starter**, click **ContosoLogoHorizontal.png**, and then click **Open**.
20. In the **Add a document** dialog box, click **OK**.
21. On the **Title, Description, and Logo** page, click **OK**.
22. Switch back to the tab that shows a preview of your master page.
23. On the preview page, reload the page.
24. Verify that the preview page now displays the updated site logo.

### ► Task 2: Add the Site Title to the Master Page

1. In Internet Explorer, switch to the **Snippet Gallery** tab.
2. On the ribbon, in the **Administration** group, click **Site Title**.
3. Under the **HTML Snippet** box, click **Copy to Clipboard**.
4. If the **Internet Explorer** dialog box appears, click **Allow access**.
5. Switch back to Visual Studio.
6. In the **header** element, locate the following line of code:

```
<div class="contentContainer">
```

7. On a new line immediately below the line of code you just located, on the **EDIT** menu, click **Paste**.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

8. Locate the following line of code:

```
<div data-name="SiteTitle">
```

9. Amend the **div** tag to include an **id** attribute value of **mainTitle**, as follows:

```
<div id="mainTitle" data-name="SiteTitle">
```



**Note:** The custom CSS file, ContosoLayout.css, uses the **mainTitle** ID to style and position the site title.

10. On the **FILE** menu, click **Save All**.
11. Switch back to Internet Explorer.
12. On the preview page, reload the page.
13. Verify that the preview page now includes the site title, displayed centrally in the header.

### ► Task 3: Add Global Navigation to the Master Page

1. In Internet Explorer, switch to the **Snippet Gallery** tab.
2. On the ribbon, in the **Navigation** group, click **Top Navigation**.
3. Under the **HTML Snippet** box, click **Copy to Clipboard**.
4. If the **Internet Explorer** dialog box appears, click **Allow access**.
5. Switch back to Visual Studio.
6. In the **header** element, locate the following line of code:

```
<div class="contentContainer">
```

7. On a new line immediately below the line of code you just located, on the **EDIT** menu, click **Paste**.



**Note:** To improve the formatting of the pasted code, press Ctrl+K+D.

8. Notice that the code you just added includes a **div** element with a **class** value that includes **ms-core-listMenu-horizontalBox**. The ContosoLayouts.css file overrides this class to style and position the navigation controls.
9. On the **FILE** menu, click **Save All**.
10. Switch back to Internet Explorer.
11. On the preview page, reload the page.
12. Verify that the preview page now includes global navigation links, displayed in white at the lower-right of the header.

**Results:** After completing this exercise, you should have added a site logo, a site title, and global navigation links to a custom SharePoint master page.

## Exercise 3: Publishing and Applying Design Assets

### ► Task 1: Publish the Draft Assets

1. In the Internet Explorer address bar, type **http://publishing.contoso.com**, and then press Enter.
2. On the **Settings** menu, click **Design Manager**.
3. On the **Design Manager: Welcome** page, on the list of options on the left side of the page, click **Edit Master Pages**.
4. On the **Design Manager: Edit Master Pages** page, in the **ContosoPrototype1** row, click the ellipsis.
5. In the **ContosoPrototype1.html** callout box, click the ellipsis, and then click **Publish a Major Version**.
6. On the **Publish Major Version** dialog box, type a comment, and then click **OK**.



**Note:** This creates a published version of both the .html file and the associated .master file.

7. On the **Settings** menu, click **Site settings**.
8. On the **Site Settings** page, under **Web Designer Galleries**, click **Master pages and page layouts**.
9. On the **ContosoLayout.css** drop-down menu, click **Publish a Major Version**.
10. In the **Publish Major Version** dialog box, type a comment, and then click **OK**.

### ► Task 2: Apply the Master Page to the Site

1. On the **Settings** menu, click **Site settings**.
2. On the **Site Settings** page, under **Look and Feel**, click **Master page**.
3. On the **Site Master Page Settings** page, under **Specify a master page to be used by this site and all sites that inherit from it**, select **ContosoPrototype1**.
4. Click **OK**.
5. Click **Contoso Publishing** to return to the site home page.
6. Verify that the new master page renders correctly.
7. Close all open windows (Visual Studio, File Explorer and Internet Explorer).

**Results:** After completing this exercise, you should have published and applied a new site master page and associated resources.

# Lab B: Configuring Farm-Wide Navigation

## Exercise 1: Creating a Custom Site Map Provider

### ► Task 1: Create an Empty SharePoint 2013 Project

1. Start the 20488B-LON-SP-15 virtual machine.
2. Log on to the LONDON machine as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. On the Start screen, type **Visual Studio 2012**, and then click **Visual Studio 2012**.
4. On the Start Page, click **New Project**.
5. In the **New Project** dialog box, in the navigation pane, expand **Visual C#**, expand **Office/SharePoint**, and then click **SharePoint Solutions**.
6. In the center pane, click **SharePoint 2013 – Empty Project**.
7. In the **Name** box, type **CrossSiteNavigation**.
8. In the **Location** box, type **E:\Labfiles\Starter**, and then click **OK**.
9. In the **SharePoint Customization Wizard** dialog box, in the **What site do you want to use for debugging** box, type **http://publishing.contoso.com**, and then click **Validate**.
10. In the **Microsoft Visual Studio** dialog box, verify that the dialog reports a successful connection, and then click **OK**.
11. Select **Deploy as a farm solution**, and then click **Finish**.

### ► Task 2: Create an XML Site Map

1. In Solution Explorer, right-click the **CrossSiteNavigation** project node, point to **Add**, and then click **SharePoint "Layouts" Mapped Folder**.
2. In Solution Explorer, under the **Layouts** folder, right-click **CrossSiteNavigation**, point to **Add**, and then click **New Item**.
3. In the **Add New Item – CrossSiteNavigation** dialog box, in the navigation pane, under **Visual C# Items**, click **Data**.
4. In the center pane, click **XML File**.
5. In the **Name** box, type **Contoso.sitemap**, and then click **Add**.
6. On a new line immediately below the XML declaration, add the following code:

```
<siteMap>
 <siteMapNode title="Root" url="http://publishing.contoso.com">
 <siteMapNode title="North" url="http://north.contoso.com" />
 <siteMapNode title="South" url="http://south.contoso.com" />
 <siteMapNode title="East" url="http://east.contoso.com" />
 <siteMapNode title="West" url="http://west.contoso.com" />
 </siteMapNode>
</siteMap>
```

7. On the **FILE** menu, click **Save All**, and then close the **Contoso.sitemap** tab.

### ► Task 3: Add a Site Map Provider to the Web.config File

1. In Solution Explorer, right-click **Features**, and then click **Add Feature**.
2. In Solution Explorer, right-click **Feature1**, and then click **Rename**.

3. Type **CrossSiteNavigation**, and then press Enter.
4. On the **CrossSiteNavigation.feature** tab, in the **Title** box, type **Cross-Site Navigation**.
5. In the **Description** box, type **Configures a site map provider for cross-site navigation**.
6. In the **Scope** drop-down list, click **WebApplication**.
7. In Solution Explorer, under **Features**, right-click **CrossSiteNavigation**, and then click **Add Event Receiver**.
8. In the **CrossSiteNavigation.EventReceiver.cs** file, on a new line immediately below the existing **using** statements, add the following code:

```
using Microsoft.SharePoint.Administration;
```

9. In the **CrossSiteNavigationEventReceiver** class, immediately after the opening brace, add the following code:

```
const string modificationName = "add[@Name='ContosoCrossSiteProvider']";
```



**Note:** This is an XPath statement that you will use to uniquely identify your Web.config modification.

10. Uncomment the **FeatureActivated** method.
11. Within the **FeatureActivated** method, add the following code:

```
var webapp = properties.Feature.Parent as SPWebApplication;
SPWebConfigModification modification = new SPWebConfigModification();
modification.Path = "configuration/system.web/siteMap/providers";
modification.Name = modificationName;
modification.Sequence = 0;
modification.Owner = "administrator@contoso.com";
modification.Type = SPWebConfigModification.SPWebConfigModificationType.EnsureChildNode;
modification.Value = "<add name='ContosoCrossSiteProvider'
 siteMapFile='_layouts/15/CrossSiteNavigation/Contoso.sitemap'
 type='Microsoft.SharePoint.Navigation.SPXmlContentMapProvider, Microsoft.SharePoint,
 Version=15.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c' />";
webapp.WebConfigModifications.Add(modification);
webapp.Update();
webapp.WebService.ApplyWebConfigModifications();
```

12. Uncomment the **FeatureDeactivating** method.
13. Within the **FeatureDeactivating** method, add the following code:

```
SPWebConfigModification modificationToRemove = null;
var webapp = properties.Feature.Parent as SPWebApplication;
var modifications = webapp.WebConfigModifications;
foreach (var modification in modifications)
{
 if (modification.Name == modificationName)
 {
 modificationToRemove = modification;
 }
}
modifications.Remove(modificationToRemove);
webapp.Update();
webapp.WebService.ApplyWebConfigModifications();
```

14. On the **BUILD** menu, click **Build Solution**.

15. Verify that the build completes without errors.

#### ► Task 4: Deploy and Test the Solution

1. On the **DEBUG** menu, click **Start Without Debugging**.
2. If you are prompted for credentials, log in as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. Open a File Explorer window and browse to the **C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS** folder.
4. Verify that the **LAYOUTS** folder includes a subfolder named **CrossSiteNavigation**.
5. Verify that the **CrossSiteNavigation** folder includes a file named **Contoso.sitemap**.
6. Close the File Explorer window.
7. In Visual Studio, on the **FILE** menu, point to **Open**, and then click **File**.
8. In the **Open File** dialog box, browse to the following directory:  

```
C:\inetpub\wwwroot\wss\VirtualDirectories\80
```
9. Click **web.config**, and then click **Open**.
10. In the Web.config file, in the configuration element, in the **system.web** element, in the **siteMap** element, in the **providers** element, verify that there is an **add** element with a name attribute value of **ContosoCrossSiteProvider**.
11. Close Visual Studio.

**Results:** After completing this lab, you should have deployed a custom site map file and registered a custom site map provider.

## Exercise 2: Adding Custom Navigation Controls to a Master Page

#### ► Task 1: Add a Data Source to the Master Page

1. Open Internet Explorer and browse to **http://publishing.contoso.com**.
2. If you are prompted for credentials, log on as **CONTOSO\Administrator** with the password **Pa\$\$w0rd**.
3. When the page loads, on the **Settings** menu, click **Design Manager**.
4. On the **Design Manager: Welcome** page, on the list on the left side of the page, click **Edit Master Pages**.
5. On the **Design Manager: Edit Master Pages** page, click **ContosoPrototype1**.
6. On the preview page, click **Snippets**.
7. On the ribbon, click **Custom ASP.NET Markup**.

- Under **Create Snippets From Custom ASP.NET Markup**, in the text box, type the following code:

```
<asp:SiteMapDataSource
 ID="bottomSiteMap"
 ShowStartingNode="false"
 SiteMapProvider="ContosoCrossSiteProvider"
 runat="server" />
```



**Note:** This ASP.NET element creates a **SiteMapDataSource** instance that retrieves navigation data from the site map provider you registered in the previous exercise.

- Click **Update**.
- Under **HTML Snippet**, at the bottom of the page, click **Copy to Clipboard**.
- If required, in the **Internet Explorer** dialog box, click **Allow access**.
- Without closing Internet Explorer, open a File Explorer window.
- In the File Explorer window, under **Computer**, click **masterpage**.



**Note:** The **masterpage** drive is the network drive that you mapped to a master page gallery in the first lab in this module.

- Right-click **ContosoPrototype1.html**, point to **Open with**, and then click **Microsoft Visual Studio 2012**.
- In the **ContosoPrototype1.html** file, in the **footer** element, on a new line within the **div** element, on the **EDIT** menu, click **Paste**.
- On the **FILE** menu, click **Save All**.
- Switch back to Internet Explorer.
- On the preview page tab, in the address bar, click **Refresh**.
- Verify that the master page preview displays without errors.



**Note:** There are no new visible page components at this stage. However, the page preview will display an error if your ASP.NET markup contains errors.



► **Task 2: Add a Custom Navigation Control to the Master Page**

1. In Internet Explorer, switch to the **Snippet Gallery** tab.
2. On the ribbon, click **Custom ASP.NET Markup**.
3. Under **Create Snippets From Custom ASP.NET Markup**, in the text box, type the following code:

```
<SharePoint:AspMenu
 ID="FooterNavigationMenu"
 runat="server"
 DataSourceID="bottomSiteMap"
 UseSimpleRendering="true"
 UseSeparateCss="false"
 Orientation="Horizontal"
 StaticDisplayLevels="1"
 MaximumDynamicDisplayLevels="0"
 SkipLinkText="" />
```



**Note:** This ASP.NET element creates an **AspMenu** instance that displays navigation data provided by the data source that you added in the previous task.

4. Click **Update**.
5. Under **HTML Snippet**, at the bottom of the page, click **Copy to Clipboard**.
6. If required, in the **Internet Explorer** dialog box, click **Allow access**.
7. Switch back to Visual Studio 2012.
8. In the **ContosoPrototype1.html** file, in the **footer** element, on a new line within the **div** element, on the **EDIT** menu, click **Paste**.
9. On the **FILE** menu, click **Save All**.
10. Switch back to Internet Explorer.
11. On the preview page tab, in the address bar, click **Refresh**.
12. Verify that the master page preview now displays four navigation links labeled **North**, **South**, **East**, and **West** in the page footer.

**Results:** After completing this exercise, you should have added custom navigation controls to a SharePoint master page.

