Microsoft® Official Course

Module 1

SharePoint Development Basics

Microsoft®

Module Overview

- Introducing the SharePoint Developer Landscape
- Choosing Approaches to SharePoint Development
- Understanding SharePoint 2013 Deployment and Execution Models

SharePoint Server 2013 Workloads

- Portals and Collaboration
- Search
- Enterprise Content Management
- Web Content Management
- Social and Communities
- Business Connectivity Services
- Business Intelligence

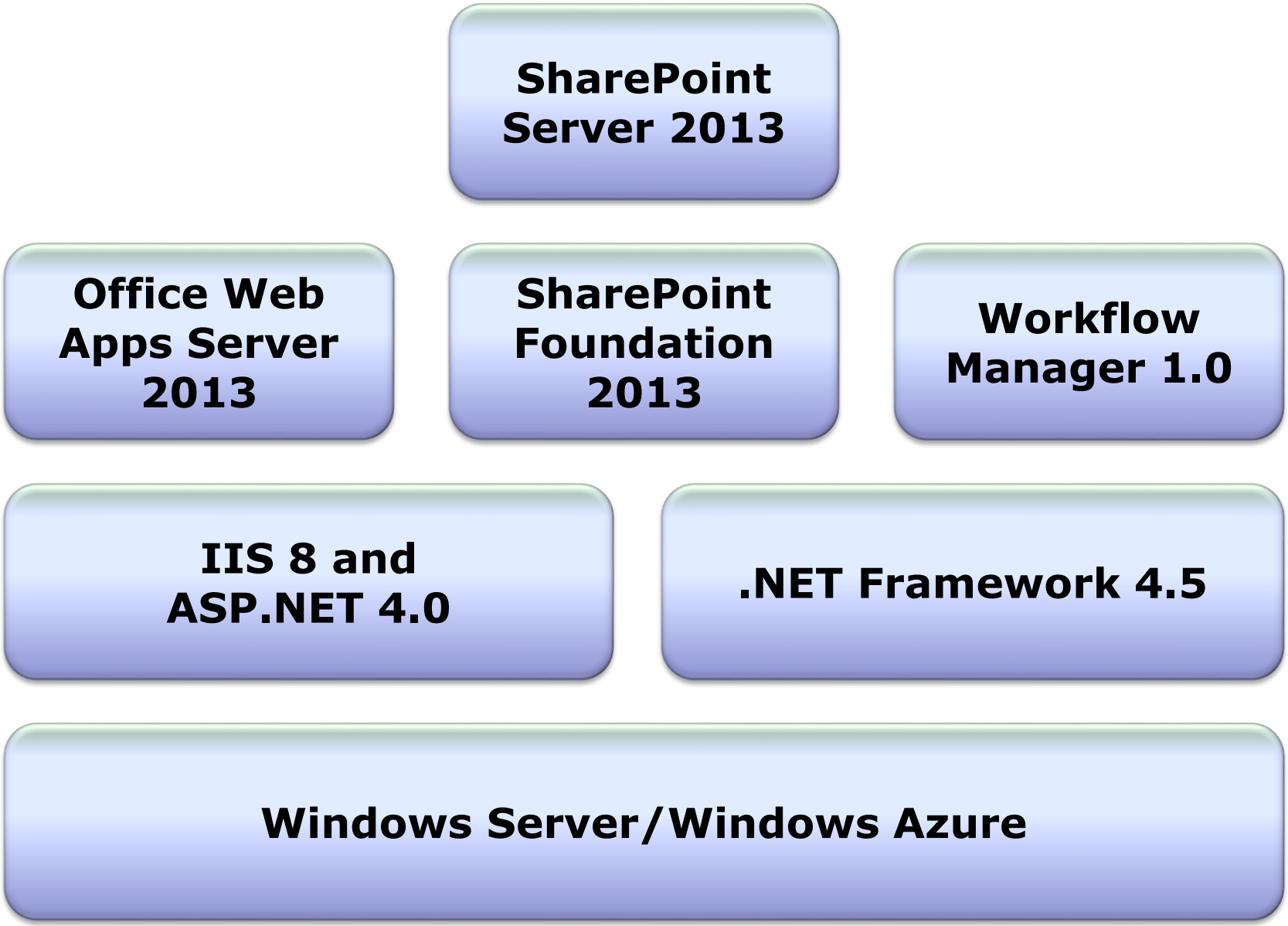
Developer Tools for SharePoint Server 2013

- Microsoft Visual Studio 2012
- Microsoft Office Tools for Visual Studio 2012
- Microsoft SharePoint Designer 2013
- Web design tools

What's New for Developers in SharePoint 2013

- The SharePoint App Model
- New Client-Side Programming Models
  - JavaScript
  - .NET Framework client
  - Silverlight/Mobile
  - REST/Odata endpoints
- New Design Model
- New Workflow Model
- Other Key Enhancements

The SharePoint 2013 Technology Stack



The SharePoint Page Rendering Process

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

Discussion: Page Rendering in SharePoint

- Is a web part page a content page or an application page?
- When might you want to deploy a custom application page to a SharePoint environment?

Entry Points for Developers in SharePoint 2013

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

Demonstration: Developer Tools for SharePoint 2013

In this demonstration, you will see a brief overview of the developer tools for SharePoint 2013.

Lesson 2: Choosing Approaches to SharePoint Development

- Declarative Components
- Client-Side Code
- Web Parts
- Application Pages
- Timer Jobs
- Event Receivers
- Workflow
- Discussion: Choosing a Suitable Development Approach

Declarative Components

- 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

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>
```

Client-Side Code

- 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?
  - 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

Web Parts

- 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)

WebPart

```
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>");
   }
}
```

Application Pages

- 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

Timer Jobs

- 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

Event Receivers

- Use event receivers to:
  - Run background tasks on a scheduled basis
  - Process queues of work items on a scheduled basis
- When should you create an event receiver?
  - When you do not require user interaction
  - When you want to remove logic from the page load process
- Where can you use event receivers?
  - On-premises deployments
  - SharePoint Online

Event Receiver Types

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.

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

Workflow

- 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

Lesson 3: Understanding SharePoint 2013 Deployment and Execution Models

- SharePoint Features
- Farm Solutions
- Sandboxed Solutions
- Apps for SharePoint

SharePoint Features

- 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

Feature Manifest

```
<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>
```

Farm Solutions

- 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

Farm Solution

```
<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>
```

Sandboxed Solutions

- 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

Apps for SharePoint

- 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

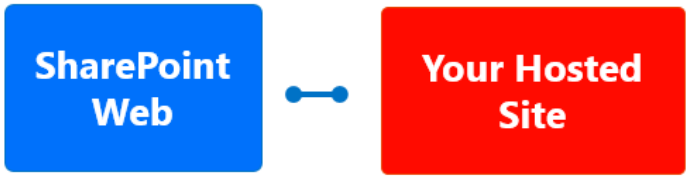
App Models

Cloud-based Apps

Get remote events from SharePoint
Use CSOM/REST + OAuth to work with SP

Provider-Hosted App

"Bring your own server hosting infrastructure"



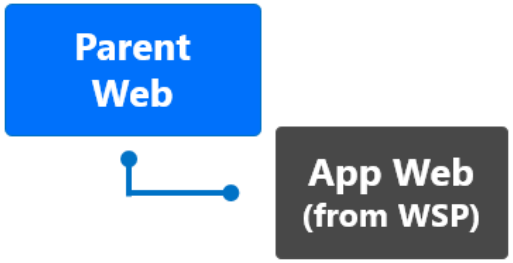
Autohosted App

Windows Azure + SQL Azure provisioned invisibly as apps are installed

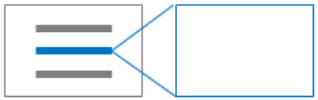
SharePoint-Hosted App

Provision an isolated sub web on a parent web

- Reuse web elements (lists, files, out-of-box web parts)
- No server code allowed; use client JavaScript for logic, UX



App Shapes

	Shape	Description	Example
	Immersive Full Page App	App that implements a new scenario for customers	Resource Tracking, Budgeting
	App Part	Provides new parts you can add to your sites	Weather, Team Mascot, News
	Extension App	Add new actions for documents and items	Display Document Visualization, Print to Print Service Vendor

SharePoint Framework

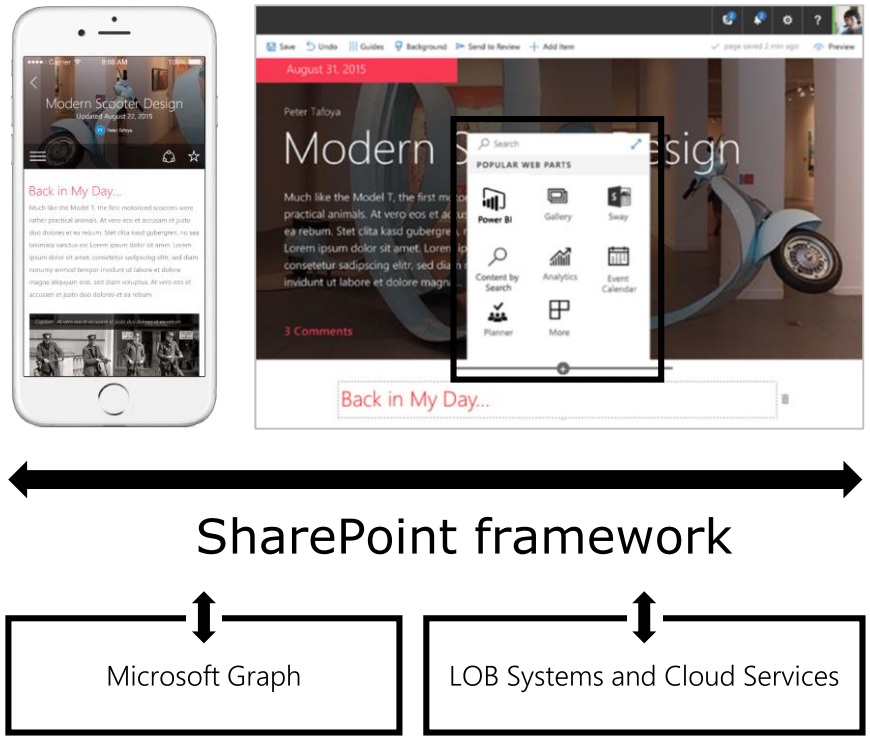
Modern client-side development

Lightweight web and mobile

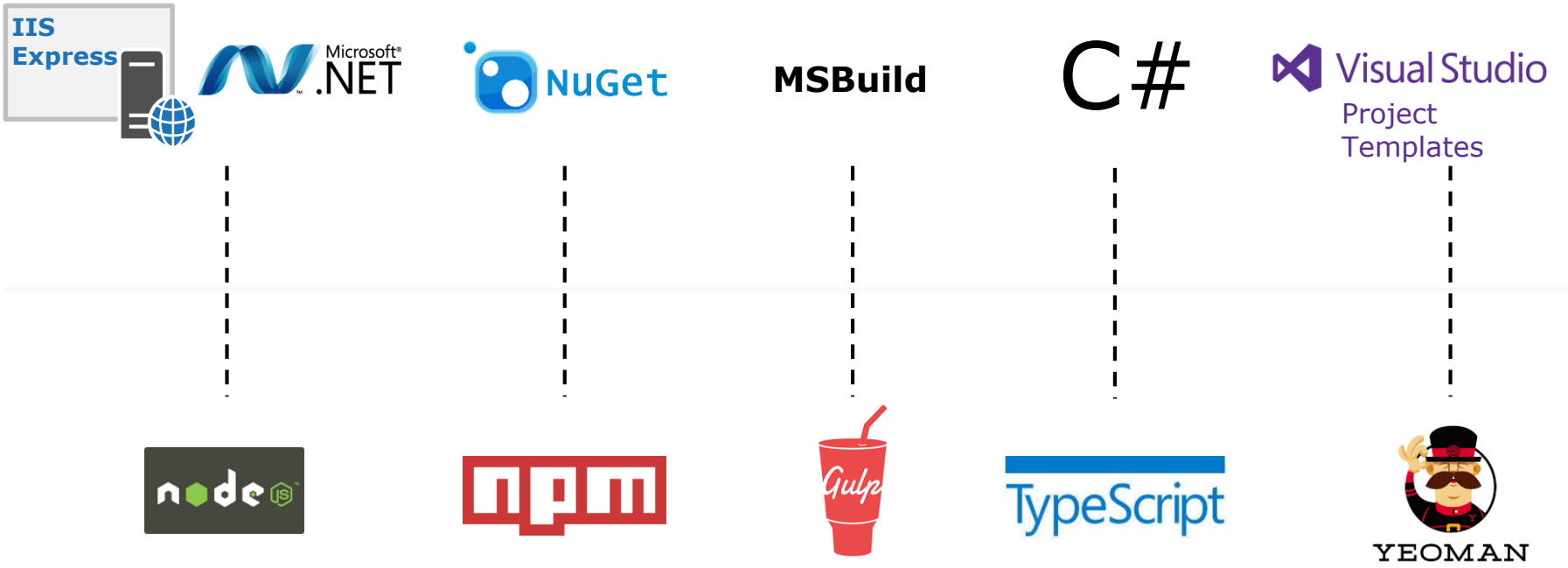
Available O365 & On-Premise

Backward compatible

Supports open source tools
and JavaScript web frameworks



Tool Comparison



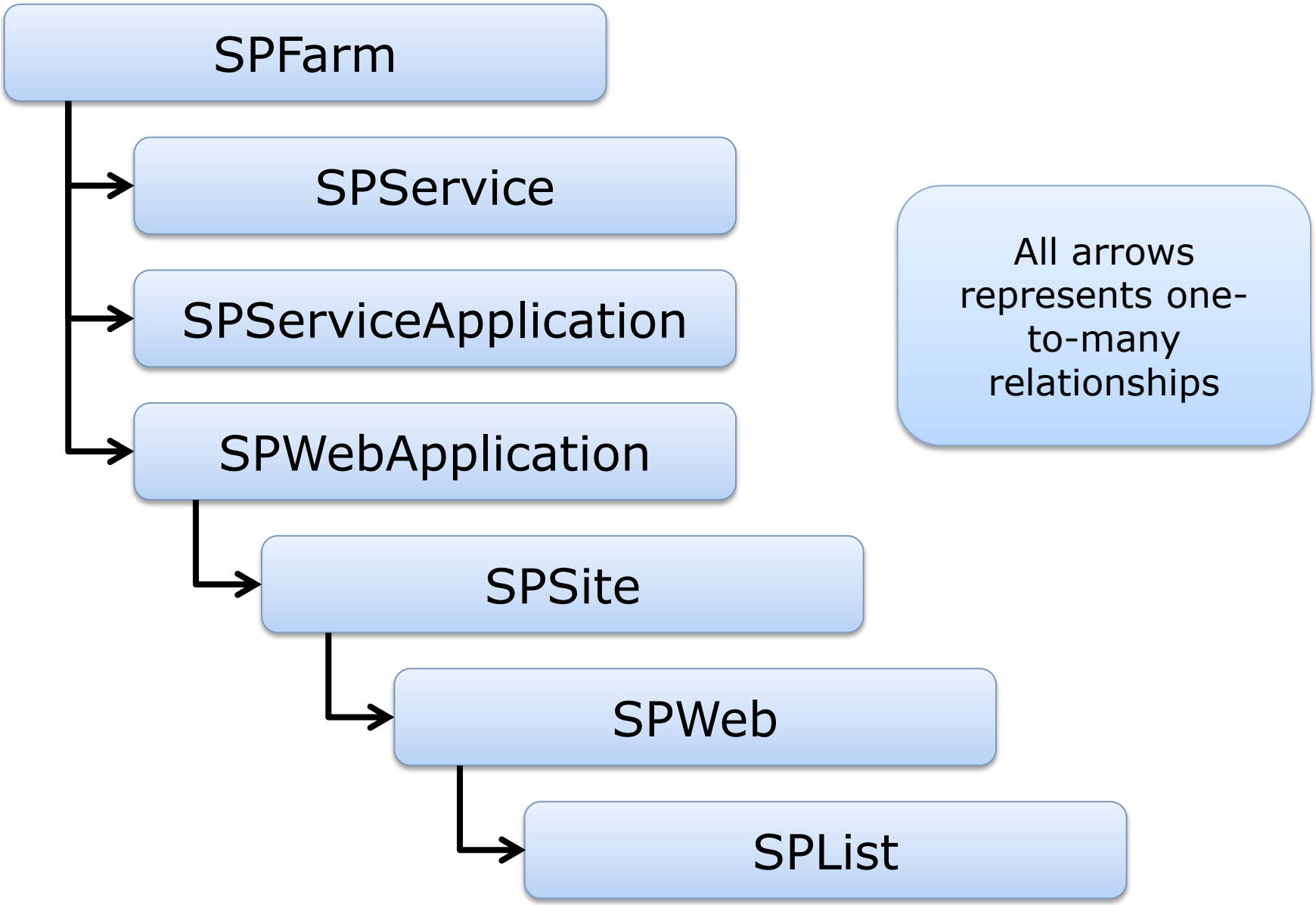
SharePoint Object Model Overview

- Understanding the SharePoint Object Hierarchy
- Working with Sites and Webs
- Working with Execution Contexts

Lesson 1: Understanding the SharePoint Object Hierarchy

- The SharePoint Object Hierarchy
- The SPFarm Class
- Working with Services
- Working with Service Applications
- Discussion: Understanding the Service Application Architecture
- Working with Web Applications
- Site Collections and the SPSite Class
- Individual Sites and the SPWeb Class

The SharePoint Object Hierarchy



The SPFarm Class

- Highest-level object in the hierarchy
- Represents farm-wide configuration
- Instantiate through the static **Local** property

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

- Use properties and methods to retrieve configuration settings

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

Working with Web Applications

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

```
// 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();
```

Site Collections and the SPSite Class

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

```
// 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();
```

Individual Sites and the SPWeb Class

- 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;
```

Lesson 2: Working with Sites and Webs

- Managing Object Lifecycles
- Retrieving and Updating Properties
- Demonstration: Updating Properties
- Creating and Deleting Sites and Webs

Managing Object Lifecycles

- **SPSite** and **SPWeb** objects are memory-intensive
- Developers must manage the object lifecycle
- 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

Disposing

```
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();
}
```

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

Retrieving and Updating Properties

- 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;
```

Creating Sites and Webs

- 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** method on the object
  - You must still dispose of the object properly

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

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();
}
```

Lesson 3: Working with Execution Contexts

- Understanding the SharePoint Context
- Working with Users and Permissions
- Discussion: Adapting Content for Different User Permissions
- Manipulating the Execution Context

Understanding the SharePoint Context

- **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

Working with Users and Permissions

- 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>
```

Manipulating the Execution Context

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

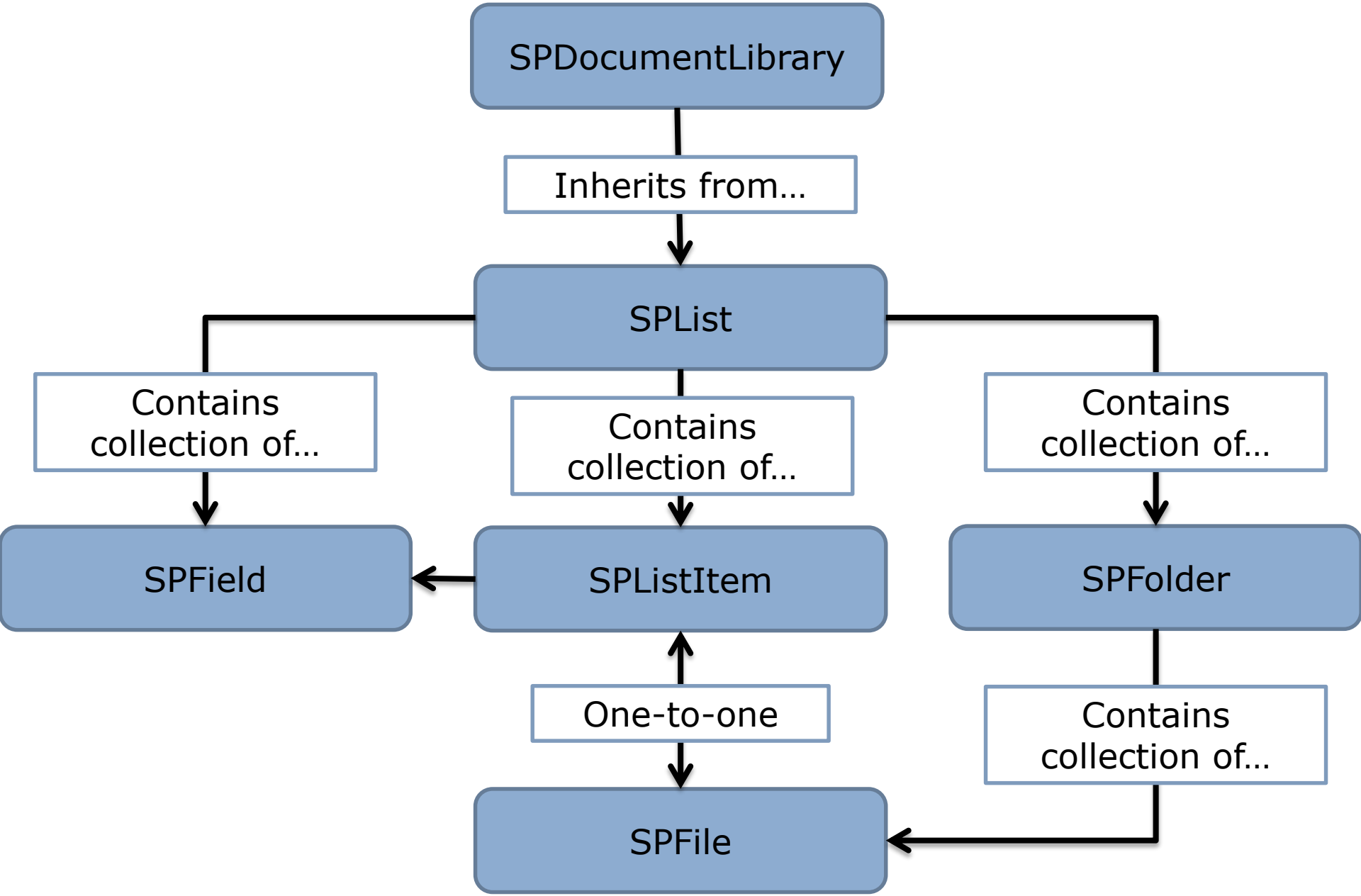
```
var delegateDPO = new
   SPSecurity.CodeToRunElevated(DoPrivilegedOperation);
SPSecurity.RunWithElevatedPrivileges(delegateDPO);

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

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

Using List and Library Objects

- Understanding the List and Library Class Hierarchy
- Retrieving List and Library Objects
- Creating and Deleting List and Library Objects
- Working with List Items
- Demonstration: Creating List Items
- Working with Fields
- Working with Files

Understanding the List and Library Class Hierarchy



Retrieving List and Library Objects

- 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;
}
```

Creating and Deleting List and Library Objects

- 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();
```

Working with List Items

- 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*)

Adding / Modifiying a List Item

```
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();
```

```
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();
```

Demonstration: Creating List Items

In this demonstration, you will see an example of how to create list items programmatically.

Working with Fields

- 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;
```

Working with Files

- 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

Lesson 2: Querying and Retrieving List Data

- Approaches to Querying List Data
- Discussion: Retrieving List Data in Code
- Building CAML Queries
- Using the SPQuery Class
- Using the SPSiteDataQuery Class
- Using LINQ to SharePoint
- Using SPMetal to Generate Entity Classes
- Demonstration: Generating Entity Classes in Visual Studio 2012

Approaches to Querying List Data

- 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

Discussion: Retrieving 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?

CAML Query 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.

Building 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>
```

Using the SPQuery Class

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);
```

Using the SPSiteDataQuery Class

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);
```

Using LINQ to SharePoint

- 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;
```

Using SPMetal to Generate Entity Classes

- 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
```

Working with Features

- Introduction to Features
- Developing Features
- Using Feature Receivers
- Defining Feature Dependencies
- Demonstration: Developing a Feature
- Manually Deploying Features
- Versioning and Upgrading Features
- Demonstration: Upgrading a Feature

Introduction to Features

- Deploy declarative components by using CAML – based XML files
- SharePoint root\LAYOUTS\FEATURES folder
- Farm solution, sandboxed solution, or an app
- Feature manifest (feature.xml) file
- Scope

Developing Features

- Feature manifest (feature.xml)

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
    Id="BC51F5D5-7DCB-4B7A-9AD5-7BD19A5DDEA9"
    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>
```

- Element manifest (elements.xml)

Developing Features

- Feature manifest (feature.xml)

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
    Id="BC51F5D5-7DCB-4B7A-9AD5-7BD19A5DDEA9"
    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>
```

- Element manifest (elements.xml)

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/" >
    <Module Name="ContosoWebPart" List="113" Url="_catalogs/wp">
        <File Path="Contoso\WebPart.webpart" Url="ContosoWebPart.webpart"
Type="GhostableInLibrary">
            <Property Name="Group" Value="ContosoWebParts" />
        </File>
    </Module>
</Elements>
```

Using Feature Receivers

- Handle Feature lifecycle events
  - Feature activated
  - Feature deactivated
  - Feature installed
  - Feature uninstalled
  - Feature upgrading

- Inherit from the SPFeatureReceiver class
- Feature manifest attributes that must be defined:
  - ReceiverAssembly
  - ReceiverClass

Feature Receiver

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
    Title="Contoso Feature"
    Scope="Site"
    Id="68FE723C-2056-48B8-AAED-FE4791134290"
    ReceiverAssembly="ContosoFeatureReceivers"
    ReceiverClass="ContosoFeatureReceivers.ContosoFeatureReceiver">
</Feature>
```

Defining Feature Dependencies

Define in the Feature manifest

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
Title="ContosoDependentFeature"
    Id="BD7BF2E4-FA17-4BFB-ADF0-561EEC6D9CBB" Scope="Web">
    <ActivationDependencies>
<ActivationDependency FeatureId="4F533AE4-7A53-4BB8-95E6-68A2615407E2"
            FeatureTitle="ContosoFeature" />
<ActivationDependency FeatureId="54AE4C0C-6E7C-4859-B625-42DC7B0CA06A"
            FeatureTitle="ContosoFeature2" />
    </ACticationDependencies>
</Feature>
```

Demonstration: Developing a Feature

- In this demonstration you will see how to:
  - Edit a Feature by using Visual Studio
  - Define Feature dependencies by using Visual Studio
  - Add an event receiver to a Feature
  - Edit Feature.xml files manually in Visual Studio

Manually Deploying Features

- App, farm solution, sandboxed solution, or manual deployment
- 15\TEMPLATES\FEATURES*CustomFeature* folder
- Install-SPFeature
- Enable-SPFeature
- Disable-SPFeature
- Uninstall-SPFeature

Introduction to Farm Solutions

- Server-side component deployment
- Deploy once; propagation to SharePoint servers
- Reduce the risk of human error
- Retract solutions
- Alternatives
  - App
  - Sandboxed solution

Packaging a Solution

- Cabinet file with wsp extension
- Visual Studio or makecab.exe
- Manifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
    SolutionId=" C4085A8F-7610-4DBC-8508-950092AE33D0" SharePointProductVersion="15.0">
    <Assemblies>
        <Assembly Location="CustomCode.dll" DeploymentTarget="GlobalAssemblyCache">
            <SafeControls>
<SafeControl Assembly="CustomCode, Version=1.0.0.0, Culture=neutral,
                    PublicKeyToken=0d89fe196be0ab46" Namespace="CustomCode.WebPartCode"
                    TypeName="*" />
                <!-- Add additional SafeControl elements here. -->
            </SafeControls>
        </Assembly>
        <!-- Add additional Assembly elements here. -->
    </Assemblies>
    <RootFiles>
        <RootFile Location="CONFIG\CustomConfig.config" />
        <!-- Add additional RootFile elements here. -->
    </RootFiles>
    <TemplateFiles>
        <TemplateFile Location="Layouts\ContosoApplicationPages\ApplicationPage.aspx" />
        <!-- Add additional TemplateFile elements here -->
    </TemplateFiles>
    <FeatureManifests>
        <FeatureManifest Location="ContosoFeature\Feature.xml" /?
        <!-- Add additional FeatureManifest elements here -->
    </FeatureManifests>
</Solution>
```

Demonstration: Creating a Solution

- In this demonstration you will see how to:
  - Create a solution by using Visual Studio.
  - Map SharePoint folders.
  - Add assemblies and define safe control entries.
  - Add Features to a solution.
  - Edit the solution manifest manually in Visual Studio.

  - Teacher Book 4-14

Deploying Solutions

- Add-SPSolution
  - LiteralPath
- Install-SPSolution
  - Identity
  - GACDeployment
  - CompatibilityLevel
- Uninstall-SPSolution
  - Identity
  - CompatibilityLevel
- Remove-SPSolution
  - Identity