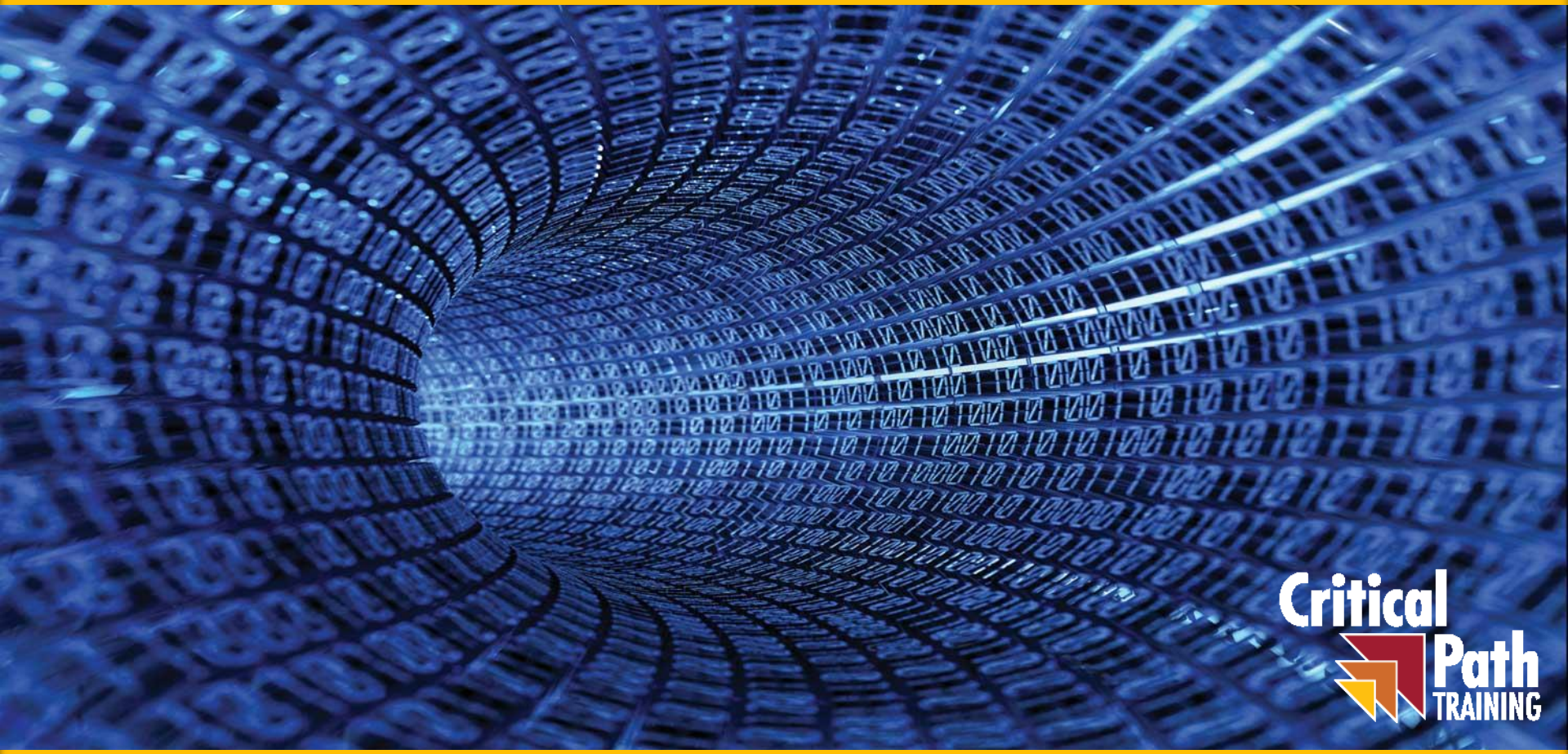
Developing Custom Solutions for SharePoint Online



Agenda

- SharePoint Online Development Strategies
- Understanding Modern Team Site and Modern Pages
- Programming the Client-side Object Model (CSOM)
- Creating Site Columns, Content Types and Lists
- JavaScript Injection and the SharePoint REST API

Evolution of the SharePoint Platform

- Farm Solutions
- ~~Sandboxed Solutions~~
- SharePoint Add-ins
- JavaScript Injection
- Remote Provisioning
- SharePoint Framework (SPFx)

APIs used by SharePoint Add-ins

- Client-side Object Model (CSOM)
  - Commonly used with .NET/C# code
  - Good fit when creating desktop clients (e.g. Console app)
  - Good fit when developing provider-hosted add-ins
  - Used to perform remote provisioning in SPO sites
- SharePoint REST API
  - Commonly used with client-side JavaScript code
  - Good fit when developing with JavaScript injection
  - Good fit when developing SharePoint-hosted add-ins
  - Accessible to any type of client on any platform

Agenda

- ✓ SharePoint Online Development Strategies
- ➢ Understanding Modern Team Site and Modern Pages
- Programming the Client-side Object Model (CSOM)
- Creating Site Columns, Content Types and Lists
- JavaScript Injection and the SharePoint REST API


DEMO
Looking at Modern Pages

Agenda

- ✓ SharePoint Online Development Strategies
- ✓ Understanding Modern Team Site and Modern Pages
- ➢ Programming the Client-side Object Model (CSOM)
- Creating Site Columns, Content Types and Lists
- JavaScript Injection and the SharePoint REST API

Why Client Object Model (CSOM)?

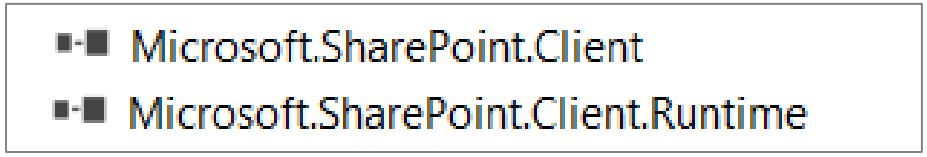
- Advantages of CSOM over the REST API
  - Strongly-typed programming
  - Format Digest managed automatically
  - Higher productivity when writing C# or VB
  - Provides ability to batch requests to web server
  - CSOM provides functionality beyond REST APIs
- CSOM more preferable on server-side C#
  - CSOM isn't best fit for JavaScript apps

Supported CSOM Functionality

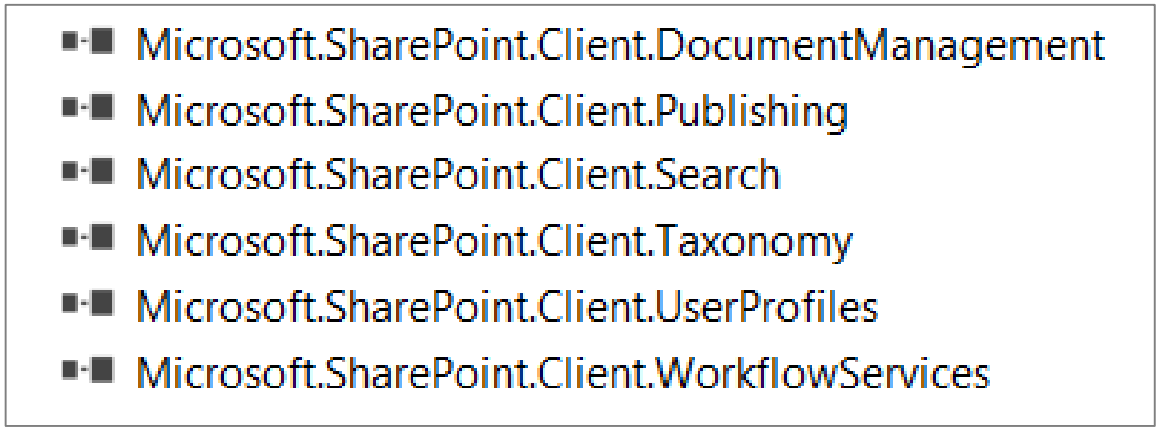
- What can you do with CSOM?
  - Work within a specific site collection
  - Read and modify site properties
  - Create site columns and content types
  - Create lists, items, views and list types
  - Register remote event handlers
  - Create folder and upload and download files
  - Add web part and web part pages
  - Create new site collections

CSOM in SharePoint Online

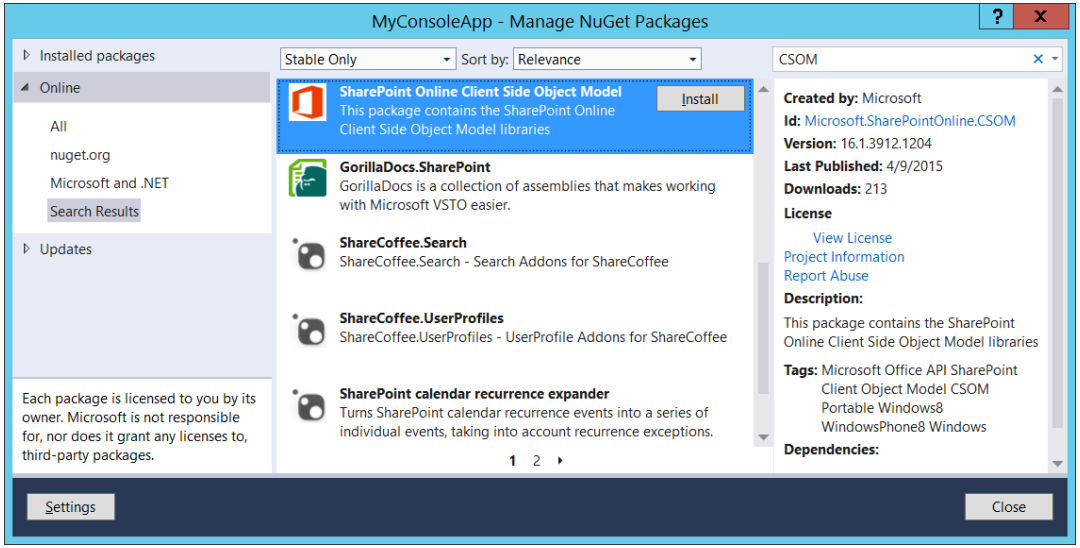
- CSOM Assemblies for SharePoint Foundation
  - Version 15 intended for SharePoint 2013 On-premises
  - Version 16.0 intended for SharePoint 2016 On-premises
  - Version 16.1 (or greater) intended for SharePoint Online

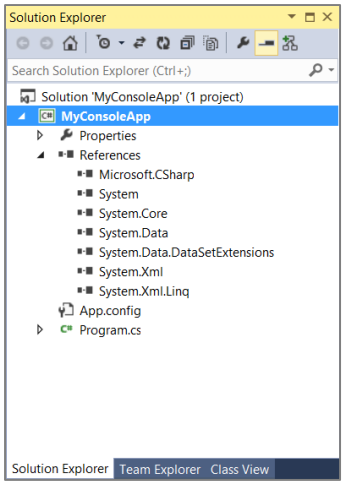


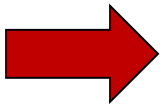
- CSOM Assemblies for SharePoint Server

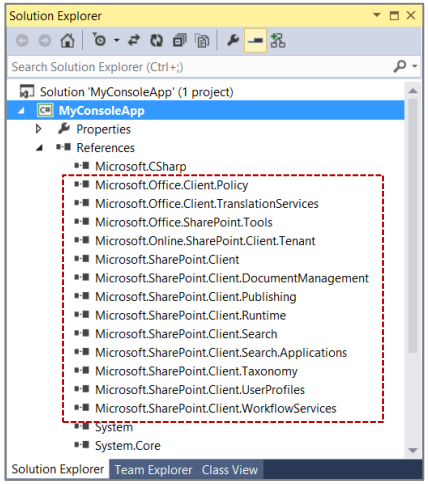


SPO CSOM NuGet Package



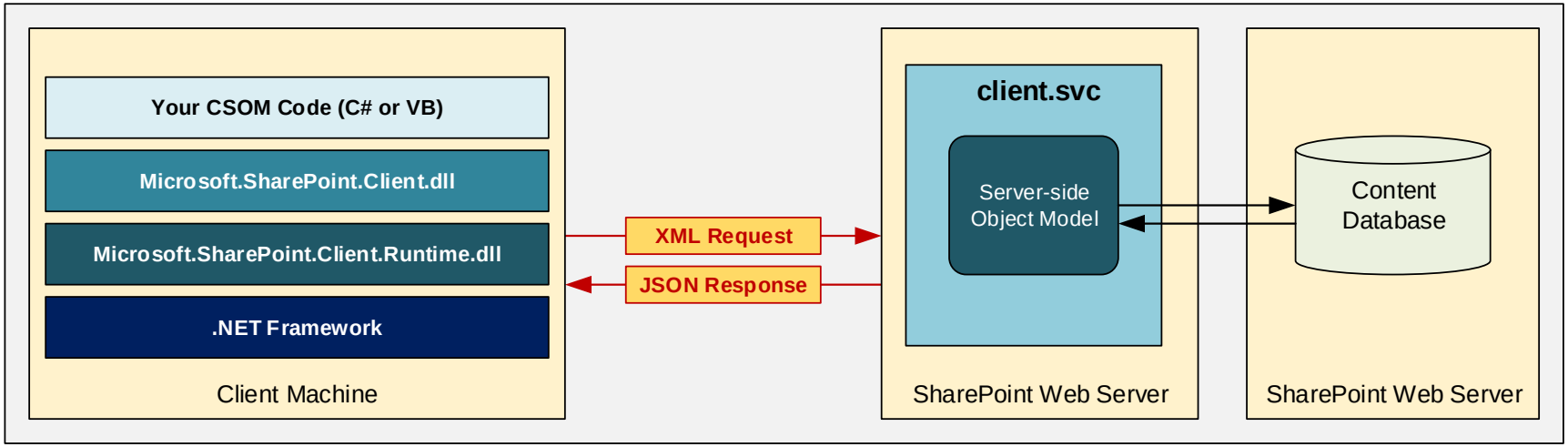






CSOM Architecture

- CSOM Objects act as client-side proxies
  - CSOM uses Windows Communication Foundation (WCF)
  - CSOM Runtime layer handles WCF calls behind scenes
  - Request body contains XML document of instructions
  - Response returned in JavaScript Object Nation (JSON)



ClientContext

- CSOM coding starts with ClientContext
  - Provides connection to SharePoint site
  - Provides access to site and site collection
  - Provides authentication behavior
  - Provides ExecuteQuery method to call server

```
string siteUrl = "http://intranet.wingtip.com";

ClientContext clientContext = new ClientContext(siteUrl);
```

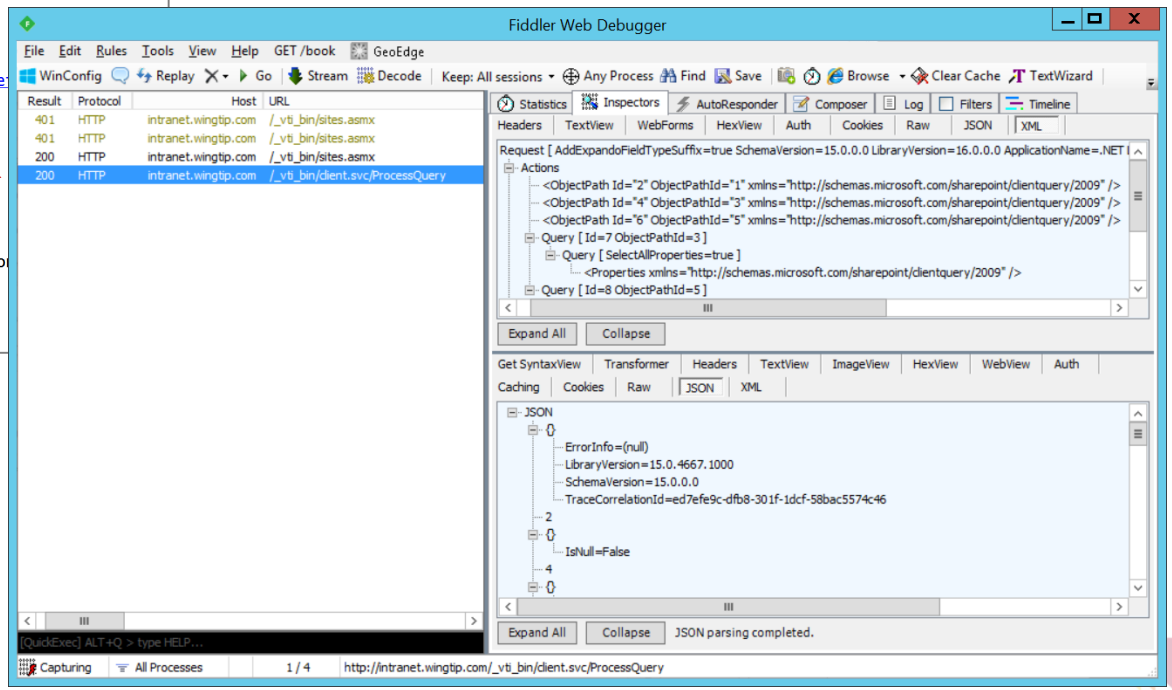
Hello CSOM

```
using System;
using Microsoft.SharePoint.Client;

namespace HelloCSOM {
  class Program {
    static void Main() {

      ClientContext clientContext = new ClientContext("http://intranet.wingtip.com");

      Site siteCollection = clientContext.Site;
      Web site = clientContext.Web;

      clientContext.Load(siteCollection);
      clientContext.Load(site);

      clientContext.ExecuteQuery();

      Console.WriteLine("The site collection URL is " + siteCollection.Url);
      Console.WriteLine("The site title is " + site.Url);
    }
  }
}
```

Inspecting CSOM Calls with Fiddler

- `ExecuteQuery` triggers call to SharePoint web server
  - CSOM calls made behind the scenes using WCF
  - CSOM calls target `/_vti_bin/client.svc/ProcessQuery`
  - Can be helpful to inspect CSOM calls using Fiddler Web Debugger

```
using System;
using Microsoft.SharePoint.Client;

namespace HelloCSOM {
  class Program {
    static void Main() {

      ClientContext clientContext = new ClientContext("http://intrane

      Site siteCollection = clientContext.Site;
      Web site = clientContext.Web;

      clientContext.Load(siteCollection);
      clientContext.Load(site);

      clientContext.ExecuteQuery();

      Console.WriteLine("The site collection URL is " + siteCollectio
      Console.WriteLine("The site title is " + site.Url);
    }
  }
}
```



User Authentication (On-premises)

```
string siteUrl = "http://intranet.wingtip.com";
ClientContext clientContext = new ClientContext(siteUrl);

// set up authentication credentials
string userName = @"WINGTIP\Administrator";
string userPassword = "Password1";
clientContext.Credentials = new NetworkCredential(userName, userPassword);

// get title of the target site
Web site = clientContext.Web;
clientContext.Load(site);

// call across network
clientContext.ExecuteQuery();

// display title
Console.WriteLine(site.Title);
```

User Authentication (SPO)

```
string siteUrl = "https://SharepointConfessions.sharepoint.com";
ClientContext clientContext = new ClientContext(siteUrl);

string userName = "tedp@sharepointconfessions.onmicrosoft.com";
string userPassword = "PinkieDoo@42";
// convert password to SecureString format
SecureString secureUserPassword = new SecureString();
foreach (char c in userPassword.ToCharArray()) {
  secureUserPassword.AppendChar(c);
}

// create SharePointOnlineCredentials object to authenticate
clientContext.Credentials =
    new SharePointOnlineCredentials(userName, secureUserPassword);

// get title of the target site
Web site = clientContext.Web;
clientContext.Load(site);

// call across network
clientContext.ExecuteQuery();

// display title
Console.WriteLine(site.Title);
```

Agenda

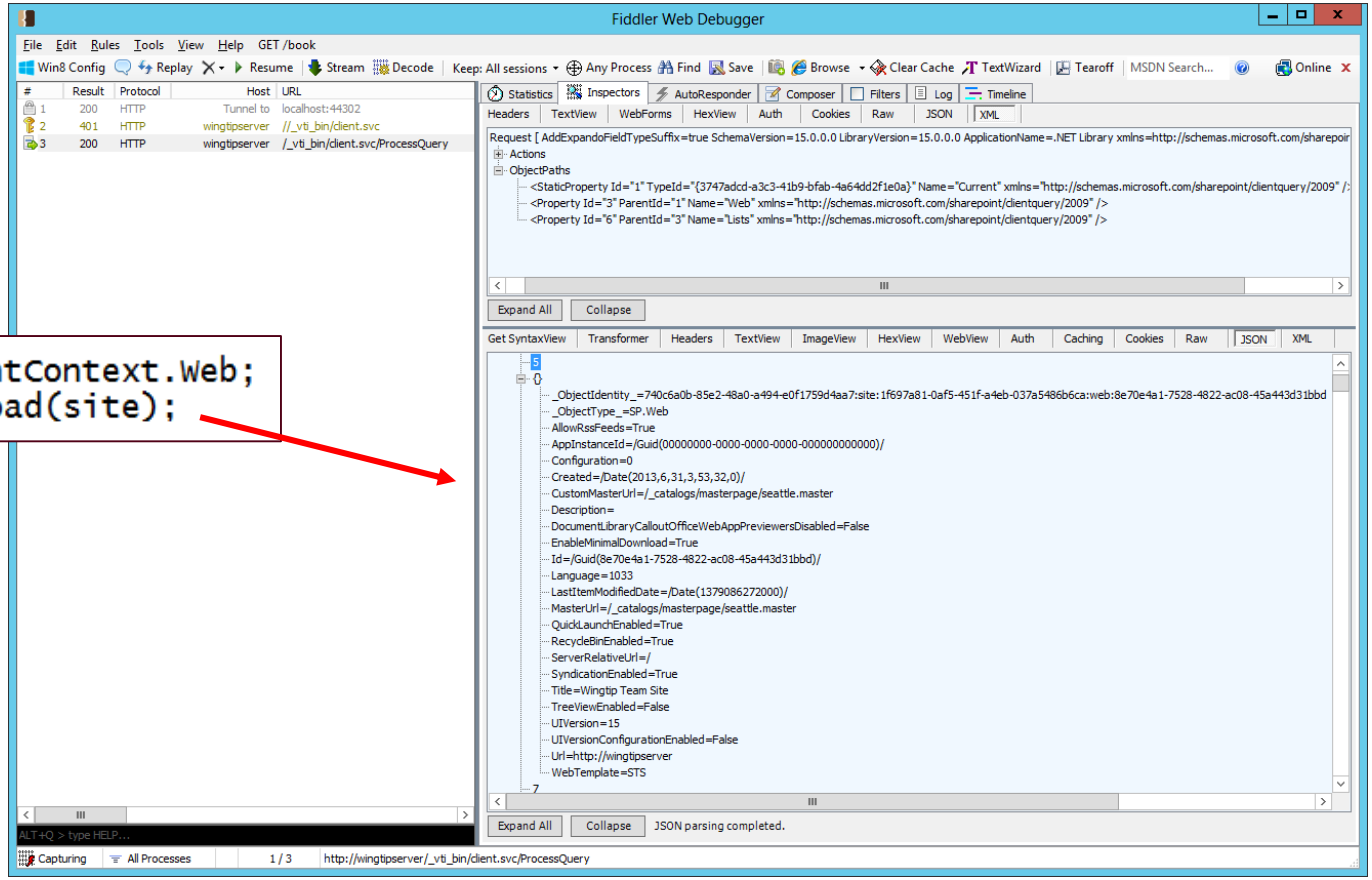
- ✓ CSOM Fundamentals
- ✓ User and App Authentication
- ➢ CSOM Code Optimization
- Remote Exception Handling
- Creating Content Types and Lists
- Managed Metadata and Publishing

What's Wrong with This Code?

```
  Web site = clientContext.Web;
  clientContext.Load(site);
  clientContext.Load(site.Lists);

  clientContext.ExecuteQuery();

  string html = "<h2>List in host web</h2>";

  html += "<ul>";

  foreach (var list in site.Lists) {
    if (list.Hidden != true) {
      html += "<li>" + list.Title + "</li>";
    }
  }

  html += "</ul>";

  WriteContentToPage(html);
}
```

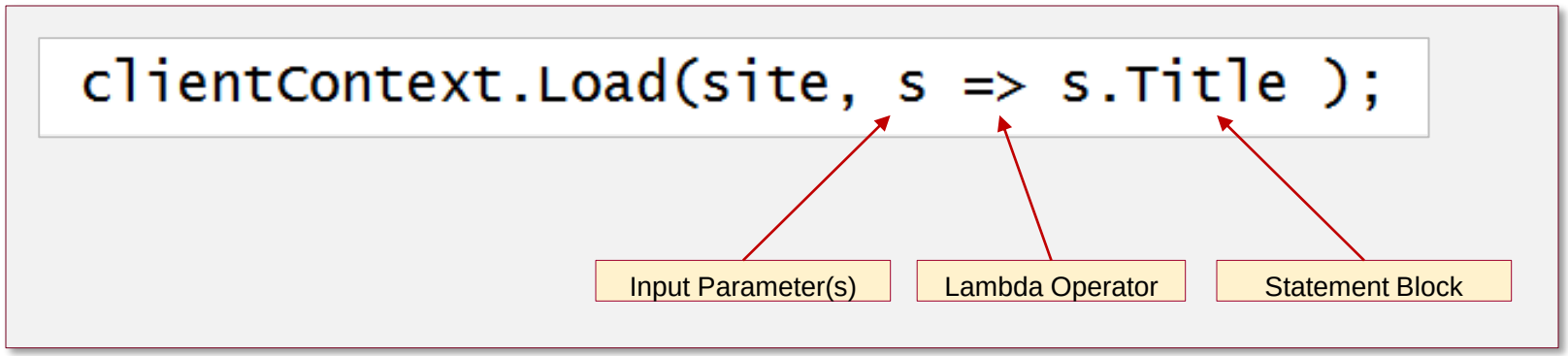
Inspecting CSOM Calls using Fiddler

```
Web site = clientContext.Web;
clientContext.Load(site);
```



Coding with Lambda Expressions

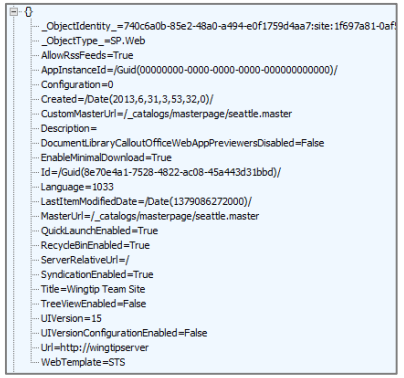
- C# supports the use of lambda expressions
  - Syntax Introduced as part of LINQ with .NET 3.5
  - Can (and should) be used with CSOM
- Lambda expression is anonymous function
  - It defines a parameter list and a function body



Using Lambda Expressions

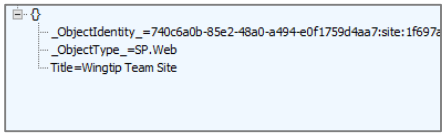
- Loading an object populates all scalar property values
  - Can result in inefficient use of network bandwidth

```
Web site = clientContext.Web;
clientContext.Load(site);
clientContext.ExecuteQuery();
```



- Lambda expressions can be used to optimize
  - You can indicate which properties you want populated

```
Web site = clientContext.Web;
clientContext.Load(site, s => s.Title);
clientContext.ExecuteQuery();
```



Using Where() and Include()

- Where lets you pass filter criteria to server

```
// instead of this
clientContext.Load(site.Lists);

// use this instead
clientContext.Load(site.Lists, lists => lists.Where(list => !list.Hidden));
```

- Include lets you pick fields on item in a collection

```
// indicate which list properties you want to populate for each list
clientContext.Load(site.Lists,
                   lists => lists.Include(list => list.Title, list => list.DefaultViewUrl));
```

- Syntax is powerful but tricky to read and write

```
ListCollection Lists = clientContext.Web.Lists;
clientContext.Load(Lists, lists => lists.Where(list => !list.Hidden)
                                        .Include(list => list.Title,
                                                 list => list.DefaultViewUrl));

clientContext.ExecuteQuery();
```

Check Whether List Exists

- How do you determine if a list already exists
  - CSOM doesn't provide simple approach
  - Query for the list by it's title or URL
  - Check to see if match list exists

```
Web site = clientContext.Web;
ListCollection listCollection = clientContext.Web.Lists;
clientContext.Load(listCollection,
                   lists => lists.Include(list => list.Title)
                                 .Where(list => list.Title == "Customers"));

clientContext.ExecuteQuery();

bool listDoesNotExist = (listCollection.Count == 0);

if (listDoesNotExist) {
  // create customers list if it does not exist
}
```

Retrieving Data using LoadQuery

- LoadQuery can be used instead of Load
  - Allows you to write LINQ query expressions

```
ClientContext clientContext = new ClientContext("http://intranet.wingtip.com");

ListCollection Lists = clientContext.Web.Lists;

// retrieve standard lists that are not hidden
IQueryable<List> query = from list in Lists
                         where (list.Hidden == false) && (list.BaseType == 0)
                         select list;

var queryResults = clientContext.LoadQuery(query);
clientContext.ExecuteQuery();

foreach (List list in queryResults) {
  Console.WriteLine(list.Title);
}
```

Retrieving with a CamlQuery

```
ClientContext clientContext = new ClientContext("http://intranet.wingtip.com");

List list = clientContext.Web.Lists.GetByTitle("Customers");
CamlQuery query = new CamlQuery();
query.ViewXml =
  @"<View>
      <Query>
        <Where>
          <BeginsWith>
            <FieldRef Name='FirstName' />
            <Value Type='Text'>B</Value>
          </BeginsWith>
        </Where>
        <OrderBy>
          <FieldRef Name='Title' />
        </OrderBy>
      </Query>
      <ViewFields>
        <FieldRef Name='FirstName'/>
        <FieldRef Name='Title' />
        <FieldRef Name='WorkPhone' />
      </ViewFields>
    </View>";

ListItemCollection queryResults = list.GetItems(query);
clientContext.Load(queryResults);
clientContext.ExecuteQuery();

foreach (ListItem item in queryResults) {
  Console.WriteLine(item["Title"] + ", " + item["FirstName"] + " - " + item["WorkPhone"]);
}
```

Batching Commands

```
private void CreateCustomers(ClientContext clientContext, int customerCount, int batchSize) {
  List list = clientContext.Web.Lists.GetByTitle("Customers");

  int batchCount = 0;
  foreach (var customer in CustomerFactory.GetCustomerList(customerCount, false)) {
    batchCount += 1;
    var lici = new ListItemCreationInformation();
    ListItem item = list.AddItem(new ListItemCreationInformation());
    item["FirstName"] = customer.FirstName; item["Title"] = customer.LastName;
    item["Company"] = customer.Company; item["WorkPhone"] = customer.WorkPhone;
    item["HomePhone"] = customer.HomePhone; item["Email"] = customer.EmailAddress;
    item.Update();
    // call ExecuteQuery only when reaching batch size
    if (batchCount == batchSize) {
      clientContext.ExecuteQuery();
      batchCount = 0;
    }
  }
  // make sure all items have been committed
  if (batchCount > 0) {
    clientContext.ExecuteQuery();
  }
}
```

Agenda

- ✓ SharePoint Online Development Strategies
- ✓ Understanding Modern Team Site and Modern Pages
- ✓ Programming the Client-side Object Model (CSOM)
- ➢ Creating Site Columns, Content Types and Lists
- JavaScript Injection and the SharePoint REST API

Creating a List

```
Web site = clientContext.Web;
clientContext.Load(site);

// create and initialize ListCreationInformation object
ListCreationInformation listInformation = new ListCreationInformation();
listInformation.Title = "Announcements";
listInformation.Url = "Lists/Announcements";
listInformation.QuickLaunchOption = QuickLaunchOptions.On;
listInformation.TemplateType = (int)ListTemplateType.Announcements;

// Add ListCreationInformation to lists collection and return list object
List list = site.Lists.Add(listInformation);

// modify additional list properties and update
list.OnQuickLaunch = true;
list.EnableAttachments = false;
list.Update();

// send command to server to create list
clientContext.ExecuteQuery();
```

Checking Whether List Already Exists

```
Web site = clientContext.Web;
clientContext.Load(site);

string listTitle = "Announcements";

// delete list if it exists
ExceptionHandlingScope scope = new ExceptionHandlingScope(clientContext);
using (scope.StartScope()) {
  using (scope.StartTry()) {
    site.Lists.GetByTitle(listTitle).DeleteObject();
  }
  using (scope.StartCatch()) { }
}

// create and initialize ListCreationInformation object
ListCreationInformation listInformation = new ListCreationInformation();
listInformation.Title = listTitle;
listInformation.Url = "Lists/Announcements";
listInformation.QuickLaunchOption = QuickLaunchOptions.On;
listInformation.TemplateType = (int)ListTemplateType.Announcements;

// Add ListCreationInformation to lists collection and return list object
List list = site.Lists.Add(listInformation);

// modify additional list properties and update
list.OnQuickLaunch = true;
list.EnableAttachments = false;
list.Update();

// send command to server to create list
clientContext.ExecuteQuery();
```

Creating List Items

```
ListItemCreationInformation lici = new ListItemCreationInformation();

var item1 = list.AddItem(lici);
item1["Title"] = "SharePoint introduces new app model";
item1["Body"] = "<div>Developers wonder what happened to solutions.</div>";
item1["Expires"] = DateTime.Today.AddYears(10);
item1.Update();

var item2 = list.AddItem(lici);
item2["Title"] = "All SharePoint developers must now learn JavaScript";
item2["Body"] = "<div>Some developers are more excited then others.</div>";
item2["Expires"] = DateTime.Today.AddYears(1);
item2.Update();

var item3 = list.AddItem(lici);
item3["Title"] = "CSOM programming is super fun";
item3["Body"] = "<div>Just ask my mom.</div>";
item3["Expires"] = DateTime.Today.AddDays(7);
item3.Update();

clientContext.ExecuteQuery();
```

Creating Site Columns - Part 1

```
static Field CreateSiteColumn(string fieldName, string fieldDisplayName, string fieldType) {

  Console.WriteLine("Creating " + fieldName + " site column...");

  // delete existing field if it exists
  try {
    Field fld = site.Fields.GetByInternalNameOrTitle(fieldName);
    fld.DeleteObject();
    clientContext.ExecuteQuery();
  }
  catch { }

  string fieldXML = @"<Field Name='" + fieldName + "' " +
                             "DisplayName='" + fieldDisplayName + "' " +
                             "Type='" + fieldType + "' " +
                             "Group='Wingtip' > " +
                    "</Field>";

  Field field = site.Fields.AddFieldAsXml(fieldXML, true, AddFieldOptions.DefaultValue);
  clientContext.Load(field);
  clientContext.ExecuteQuery();
  return field;
}
```

Creating Site Columns - Part 2

```
fieldProductCode = CreateSiteColumn("ProductCode", "Product Code", "Text");
fieldProductCode.EnforceUniqueValues = true;
fieldProductCode.Indexed = true;
fieldProductCode.Required = true;
fieldProductCode.Update();
clientContext.ExecuteQuery();
clientContext.Load(fieldProductCode);
clientContext.ExecuteQuery();

fieldProductDescription =
  clientContext.CastTo<FieldMultiLineText>(CreateSiteColumn("ProductDescription", "Product Description", "Note"));
fieldProductDescription.NumberOfLines = 4;
fieldProductDescription.RichText = false;
fieldProductDescription.Update();
clientContext.ExecuteQuery();

fieldProductListPrice =
  clientContext.CastTo<FieldCurrency>(CreateSiteColumn("ProductListPrice", "List Price", "Currency"));
fieldProductListPrice.MinimumValue = 0;
fieldProductListPrice.Update();
clientContext.ExecuteQuery();

fieldProductCategory =
  clientContext.CastTo<TaxonomyField>(CreateSiteColumn("ProductCategory", "Product Category", "TaxonomyFieldType"));
fieldProductCategory.SspId = localTermStoreID;
fieldProductCategory.TermSetId = termSetId;
fieldProductCategory.AllowMultipleValues = false;
fieldProductCategory.Update();
clientContext.ExecuteQuery();

fieldProductColor =
  clientContext.CastTo<FieldMultiChoice>(CreateSiteColumn("ProductColor", "Product Color", "MultiChoice"));
string[] choicesProductColor = { "White", "Black", "Grey", "Blue", "Red", "Green", "Yellow" };
fieldProductColor.Choices = choicesProductColor;
fieldProductColor.Update();
clientContext.ExecuteQuery();
```

Creating Content Types - Part 1

```
static ContentType CreateContentType(string contentTypeName, string baseContentType) {

  DeleteContentType(contentTypeName);

  ContentTypeCreationInformation contentTypeCreateInfo = new ContentTypeCreationInformation();
  contentTypeCreateInfo.Name = contentTypeName;
  contentTypeCreateInfo.ParentContentType = site.ContentTypes.GetById(baseContentType); ;
  contentTypeCreateInfo.Group = "Wingtip";
  ContentType ctype = site.ContentTypes.Add(contentTypeCreateInfo);
  clientContext.ExecuteQuery();
  return ctype;

}

static void DeleteContentType(string contentTypeName) {
  try {
    foreach (var ct in site.ContentTypes) {
      if (ct.Name.Equals(contentTypeName)) {
        ct.DeleteObject();
        Console.WriteLine("Deleting existing " + ct.Name + " content type...");
        clientContext.ExecuteQuery();
        break;
      }
    }
  }
  catch { }
}
```

Creating Content Types - Part 2

```
ctypeProduct = CreateContentType("Product", "0x01");

// add site columns
FieldLinkCreationInformation fieldLinkProductCode = new FieldLinkCreationInformation();
fieldLinkProductCode.Field = fieldProductCode;
ctypeProduct.FieldLinks.Add(fieldLinkProductCode);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductDescription = new FieldLinkCreationInformation();
fieldLinkProductDescription.Field = fieldProductDescription;
ctypeProduct.FieldLinks.Add(fieldLinkProductDescription);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductListPrice = new FieldLinkCreationInformation();
fieldLinkProductListPrice.Field = fieldProductListPrice;
ctypeProduct.FieldLinks.Add(fieldLinkProductListPrice);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductCategory = new FieldLinkCreationInformation();
fieldLinkProductCategory.Field = fieldProductCategory;
ctypeProduct.FieldLinks.Add(fieldLinkProductCategory);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductColor = new FieldLinkCreationInformation();
fieldLinkProductColor.Field = fieldProductColor;
ctypeProduct.FieldLinks.Add(fieldLinkProductColor);
ctypeProduct.Update(true);

clientContext.ExecuteQuery();
```

Creating List with Content Type

```
ListCreationInformation listInformationProducts = new ListCreationInformation();
listInformationProducts.Title = "Products";
listInformationProducts.Url = "Lists/Products";
listInformationProducts.QuickLaunchOption = QuickLaunchOptions.On;
listInformationProducts.TemplateType = (int)ListTemplateType.GenericList;
listProducts = site.Lists.Add(listInformationProducts);
listProducts.OnQuickLaunch = true;
listProducts.Update();

clientContext.Load(listProducts);
clientContext.Load(listProducts.ContentTypes);
clientContext.ExecuteQuery();

// configure list to use custom content type
listProducts.ContentTypesEnabled = true;
listProducts.ContentTypes.AddExistingContentType(ctypeProduct);
ContentType existing = listProducts.ContentTypes[0]; ;
existing.DeleteObject();
listProducts.Update();
clientContext.ExecuteQuery();

// add custom site columns to default veiw of list
View viewProducts = listProducts.DefaultView;
viewProducts.ViewFields.Add("ProductCode");
viewProducts.ViewFields.Add("ProductListPrice");
viewProducts.ViewFields.Add("ProductCategory");
viewProducts.ViewFields.Add("ProductColor");
viewProducts.Update();

clientContext.ExecuteQuery();
```

Creating a Document Library

```
ListCreationInformation listInformationProductImages = new ListCreationInformation();
listInformationProductImages.Title = "Product Images";
// make sure to set URL to root of site - not in /Lists folder
listInformationProductImages.Url = "ProductImages";
listInformationProductImages.QuickLaunchOption = QuickLaunchOptions.On;
listInformationProductImages.TemplateType = (int)ListTemplateType.PictureLibrary;
listProductImages = site.Lists.Add(listInformationProductImages);
listProductImages.OnQuickLaunch = true;
listProductImages.Update();

clientContext.ExecuteQuery();
```

Uploading Files to a Library

- Create a utility upload function with common CSOM code

```
static void UploadProductImage(byte[] imageContent, string imageFileName) {
  Console.WriteLine("  uploading " + imageFileName);
  FileCreationInformation fileInfo = new FileCreationInformation();
  fileInfo.Content = imageContent;
  fileInfo.Overwrite = true;
  fileInfo.Url = listProductImagesUrl + imageFileName;
  File newFile = listProductImages.RootFolder.Files.Add(fileInfo);
  clientContext.ExecuteQuery();
}
```

- Call function passing file name and byte array

```
UploadProductImage(Properties.Resources.WP0001, "WP0001.jpg");
UploadProductImage(Properties.Resources.WP0002, "WP0002.jpg");
UploadProductImage(Properties.Resources.WP0003, "WP0003.jpg");
UploadProductImage(Properties.Resources.WP0004, "WP0004.jpg");
UploadProductImage(Properties.Resources.WP0005, "WP0005.jpg");
UploadProductImage(Properties.Resources.WP0006, "WP0006.jpg");
UploadProductImage(Properties.Resources.WP0007, "WP0007.jpg");
UploadProductImage(Properties.Resources.WP0008, "WP0008.jpg");
UploadProductImage(Properties.Resources.WP0009, "WP0009.jpg");
UploadProductImage(Properties.Resources.WP0010, "WP0010.jpg");
```

Agenda

- ✓ SharePoint Online Development Strategies
- ✓ Understanding Modern Team Site and Modern Pages
- ✓ Programming the Client-side Object Model (CSOM)
- ✓ Creating Site Columns, Content Types and Lists
- ➢ JavaScript Injection and the SharePoint REST API

JavaScript Injection

- JavaScript injection based on central concept…
  1. upload custom JavaScript code to SharePoint Online
  2. execute code using identity and permissions of current user
- Approaches for using JavaScript injection
  - Script Editor Web Part
  - Adding JavaScript code behind SharePoint site pages
  - Full-blown Visual Studio project development
- Why create solution using JavaScript Injection?
  - Provides more flexibility than SharePoint add-in model
  - Poses fewer constraints than SharePoint add-in model

Scripting Capabilities in SharePoint Online

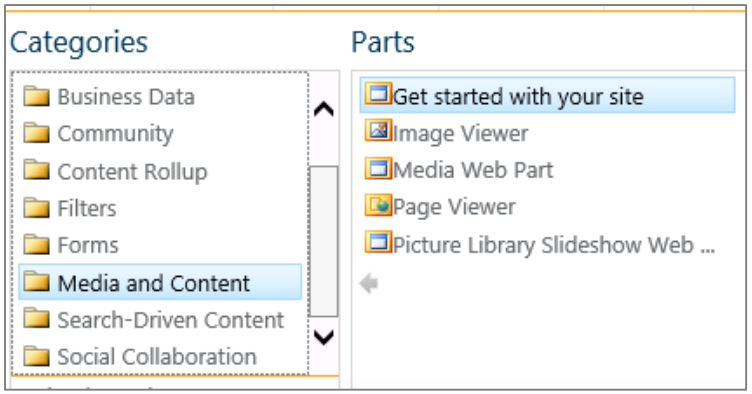
- SharePoint Online has powerful scripting features
  - It’s powerful when used by the good guys
  - It’s powerful when used by the bad guys
  - SharePoint Online disables scripting by default
- The default scripting capabilities disabled for
  - Personal sites
  - Self-service created sites
  - Root site collection of the tenant

Features Affected with Scripting Disabled

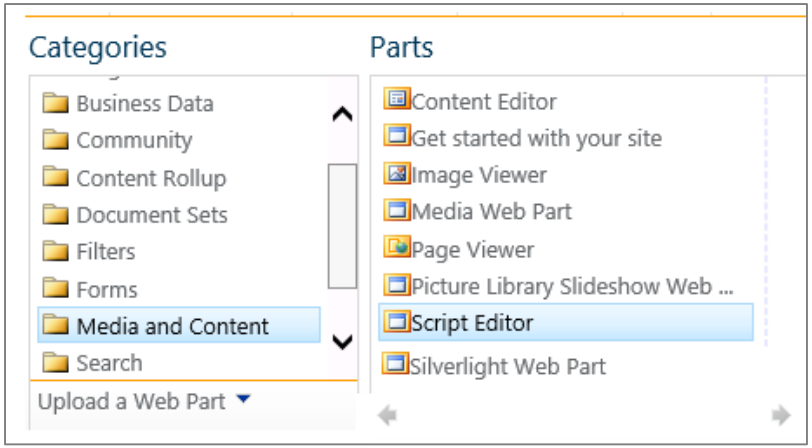
- When scripting is disabled…
  - Many links removed from Site Settings page
  - SharePoint Designer capabilities reduced
  - You cannot edit master pages or page layouts
  - You cannot edit theme for current site
  - Many Web Parts are missing (e.g. Script Editor)
  - Users cannot upload .aspx files to document libraries
- Scripting must be enabled at the site level
  - Can be done by configuring SPO tenancy policy
  - Can be done using PowerShell or CSOM

Effects of Scripting Being Disabled

- Media and Content Web Parts (scripting disabled)

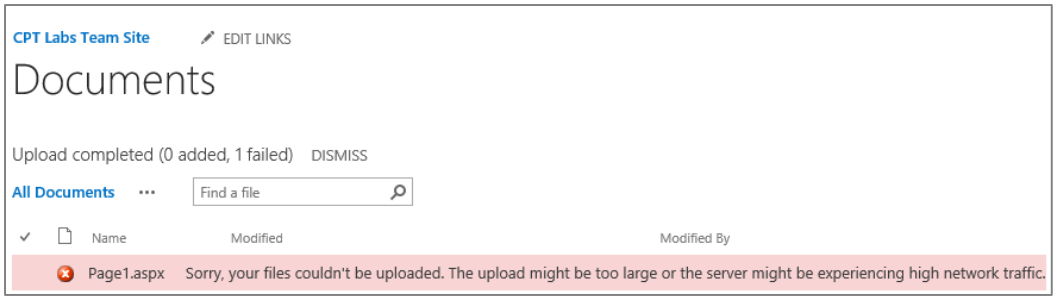


- Media and Content Web Parts (scripting enabled)



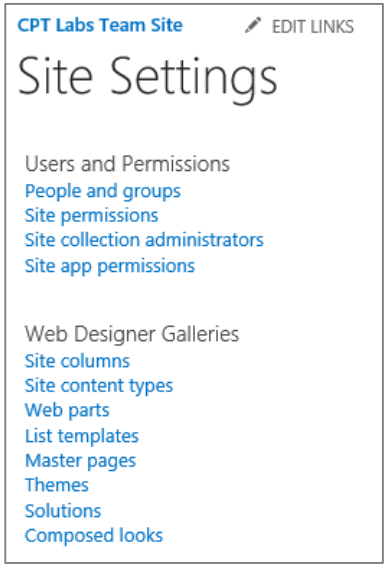
More Effects of Scripting Being Disabled

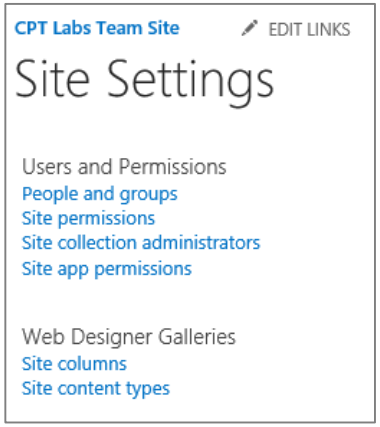
- You cannot upload a .ASPX file to a document library



- Many Administrative Links removed from Site Settings page

Scripting Enabled

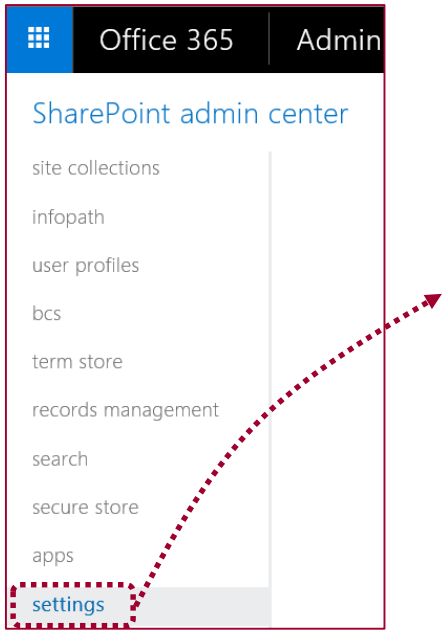


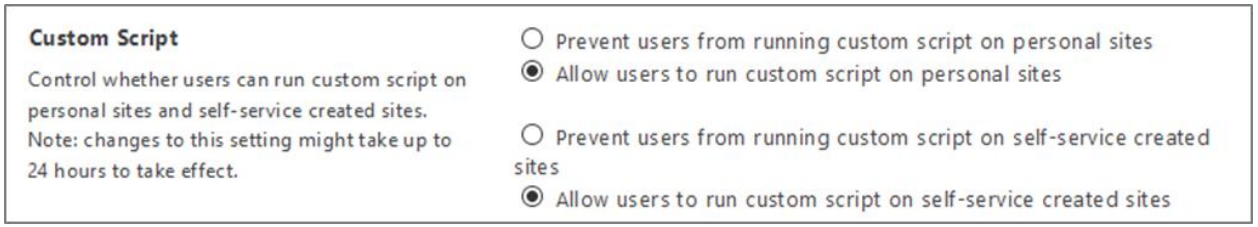


Scripting Disabled

Enabling Scripting in SharePoint Admin Center

- Settings configurable in SharePoint admin center
  - Sets policy for sites created in future
  - Sets policy for existing sites created within tenancy
  - Can take up to 24 hours to propagate changes to existing sites





Enabling Scripting using PowerShell

- Site scripting setting can be enabled using PowerShell
  - Use **`Set-SPOSite`** cmdlet to update **`DenyAddAndCustomizePages`**
  - Changes take affect immediately

- PowerShell syntax

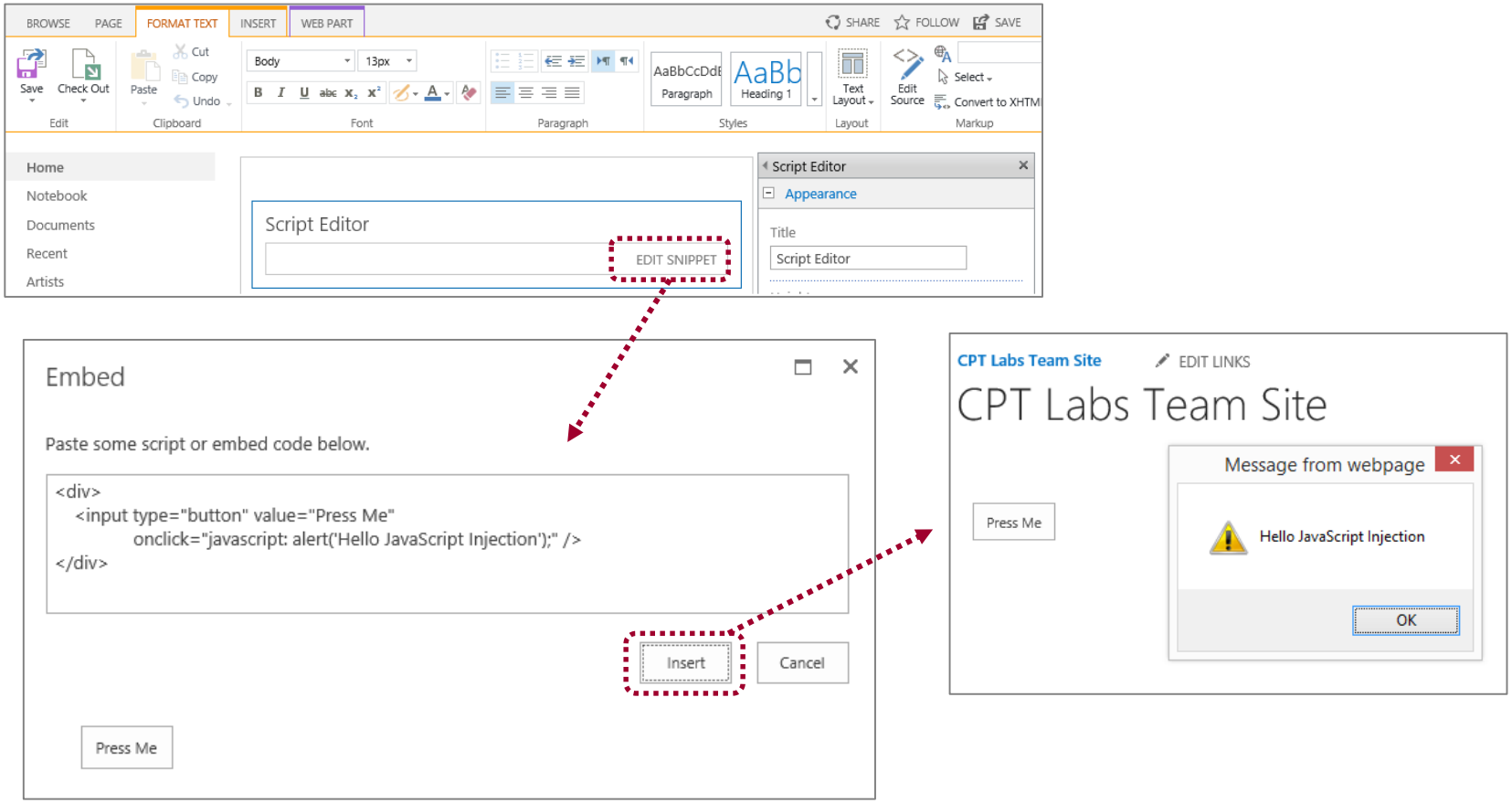
`Set-SPOsite` `<_YOUR_SITE_URL_>` **`-DenyAddAndCustomizePages 0`**

EnableScripting.ps1

```
1  # establish authenticated connection to tenant admin site collection
2  $credential = Get-Credential
3  Connect-SPOService -Url https://CptLabs-admin.sharepoint.com -Credential $credential
4
5
6  # enable scripting for a specific site collection
7  Set-SPOSite https://CptLabs.sharepoint.com -DenyAddAndCustomizePages 0
8
```

Script Editor Web Part

- Allows user to add custom script logic in ad-hoc fashion



Creating and Uploading Custom Pages

- Uploading Custom Pages
  - Scripting must be enabled for target SPO site
  - Page file must be ASPX file (HTML files do not work)
  - Page can be uploaded to any document library
  - Page can link to same master page as other site pages
  - Page can link to custom CSS files and JavaScript files
- What about the SharePoint sites running in MDS mode?
  - Minimal Download Strategy (MDS) affects how pages run
  - MDS-enabled pages run in MDS mode through start.aspx
  - MDS mode redirects unsupported pages back to non-MDS URLs

Adding a Script Link for jQuery

- SharePoint does not load jQuery library
  - It must be explicitly for Script Editor Web Part

```
<div>
  <input id="cmdPressMe" value="Press Me" type="button"  />
</div>

<!-- add script link to jQuery library-->
<script src="https://code.jquery.com/jquery-2.1.4.js"></script>

<script language="JavaScript" >
  // add jQuery document ready event handler
  $(function () {
    $("#cmdPressMe").click(function () {
      alert('Hello JavaScript Injection - This is working!')
    });
  });
</script>
```

Creating a Simple Site Pages for SPO

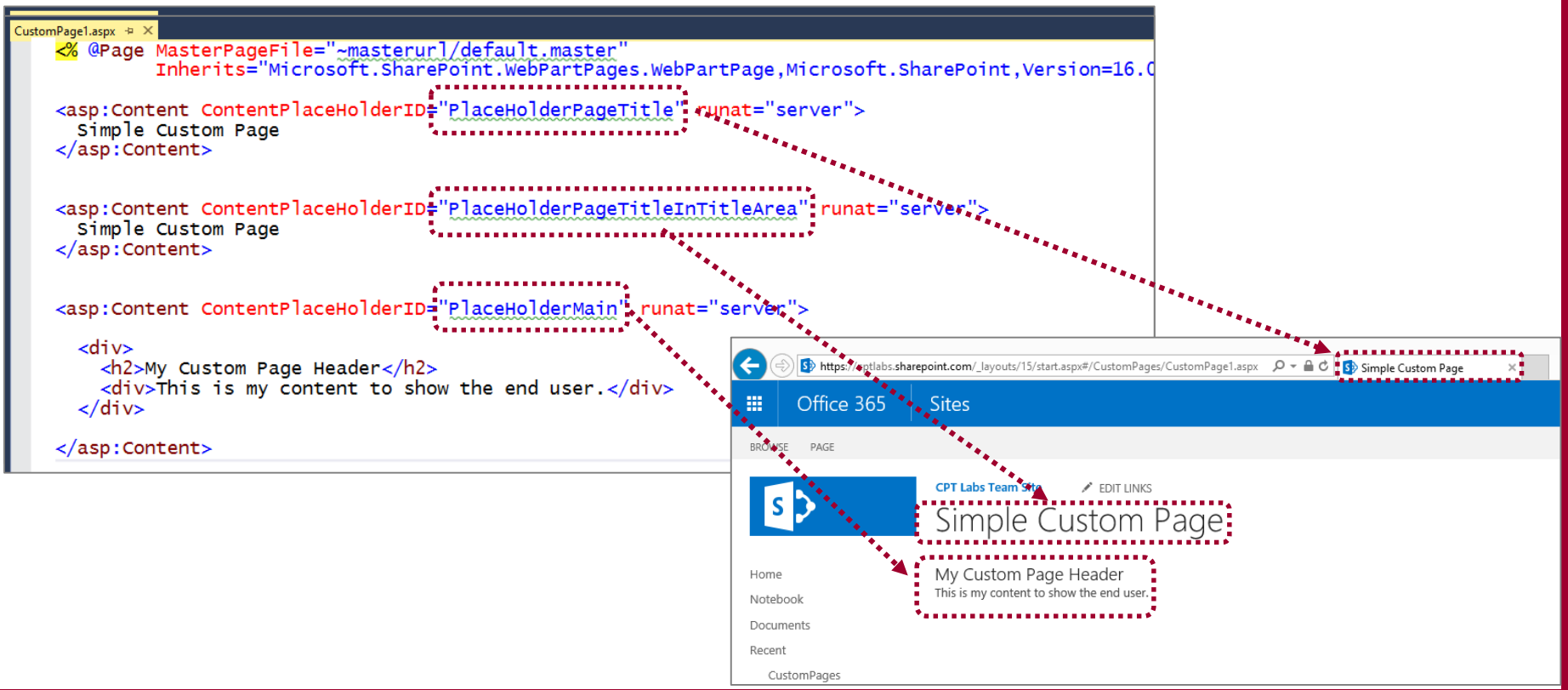
- Custom pages should link to current site's master page
  - Set **`MasterPageFile`** to dynamic token **`~masterurl.default.master`**
- Custom Page should inherit from **`WebPartPage`**
  - Required to work correctly with Minimal Download Strategy feature
  - Required if you want to add support for Web Parts

CustomPage1.aspx

```
<% @Page MasterPageFile="~masterurl/default.master"
         Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage,Microsoft.SharePoint,Version=16.0

<asp:Content ContentPlaceHolderID="PlaceHolderPageTitle" runat="server">
  Simple Custom Page
</asp:Content>


<asp:Content ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
  Simple Custom Page
</asp:Content>


<asp:Content ContentPlaceHolderID="PlaceHolderMain" runat="server">

  <div>
    <h2>My Custom Page Header</h2>
    <div>This is my content to show the end user.</div>
  </div>

</asp:Content>
```

Creating a Simple Site Pages for SPO

- Essential SharePoint Master Page Placeholders
  - PlaceHolderPageTitle
  - PlaceHolderPageTitleInTitleArea
  - PlaceHolderMain



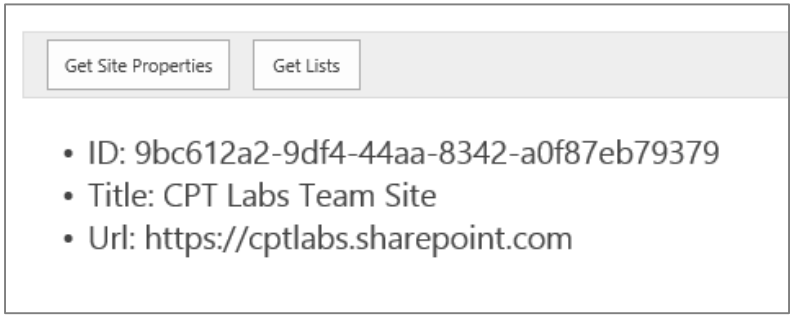
Adding Scripting to a Custom Page

- Adding scripts and links using PlaceHolderAdditionalPagehead

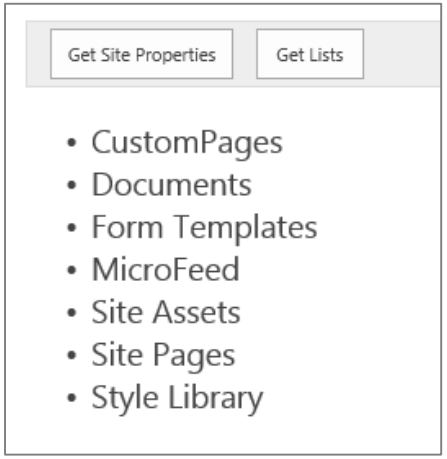
```
<asp:Content ContentPlaceHolderID="PlaceHolderAdditionalPageHead" runat="server">

  <script src="https://code.jquery.com/jquery-2.1.4.js" ></script>

  <script>

    $(function () {
      $("#getSiteProperties").click(onGetSiteProperties);
      $("#getLists").click(onGetLists);
    });

    function onGetSiteProperties()...

    function onGetLists()...

  </script>
</asp:Content>


<asp:Content ContentPlaceHolderID="PlaceHolderMain" runat="server">

  <div>
    <button id="getSiteProperties" type="button" >Get Site Properties</button>
    <button id="getLists" type="button" >Get Lists</button>
  </div>

  <div id="content_box" />

</asp:Content>
```

Programming the SharePoint REST API

```
function onGetSiteProperties() {
  var urlRest = "../_api/web/?$select=Id,Title,Url";
  $.ajax({
    url: urlRest,
    method: "GET",
    headers: {"accept": "application/json;odata=verbose"}
  }).then(function (data) {
    $("#content_box")
      .empty()
      .append($("<ul>")
        .append($("<li>").text("ID: " + data.d.Id))
        .append($("<li>").text("Title: " + data.d.Title))
        .append($("<li>").text("Url: " + data.d.Url))
      );
  });
}
```




```
function onGetLists() {
  var urlRest = "../_api/web/lists/?$filter=(Hidden eq false)";
  $.ajax({
    url: urlRest,
    method: "GET",
    headers: { "accept": "application/json;odata=verbose" }
  }).then(function (data) {
    var lists = data.d.results;
    var listOfLists = $("<ul>");
    for (var i = 0; i < lists.length; i++) {
      listOfLists.append( $("<li>").text(lists[i].Title) );
    }
    $("#content_box").empty().append(listOfLists);
  });
}
```



Remote Provisioning

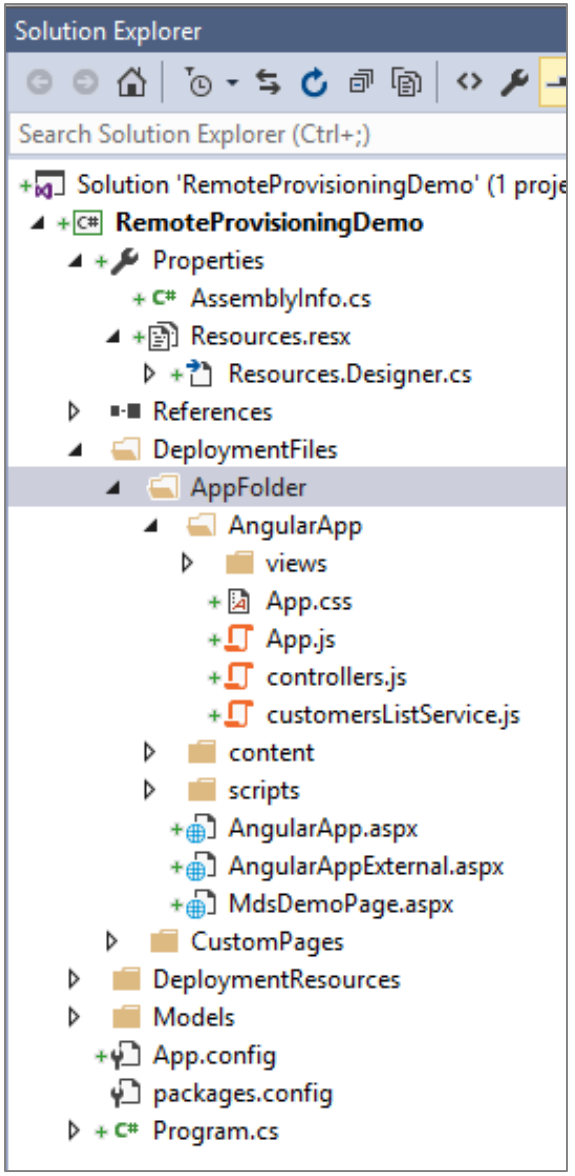
- Remote provisioning in SPO
  - Use CSOM to create SPO site elements
  - Recommended over SharePoint solutions & features
- What can you create with Remote Provisioning
  - New child sites, lists and document libraries
  - Site columns, content types and remote event receivers
  - New pages with custom JavaScript logic
  - User custom actions with custom JavaScript logic

Remote Provisioning using CSOM

- What can you do to a SPO site using CSOM?
  - Upload custom ASPX pages and JavaScript files
  - Add navigation nodes on the top navigation bar
  - Create child sites, lists and document libraries
  - Create site columns, content types and term sets
  - Create user custom actions and script links

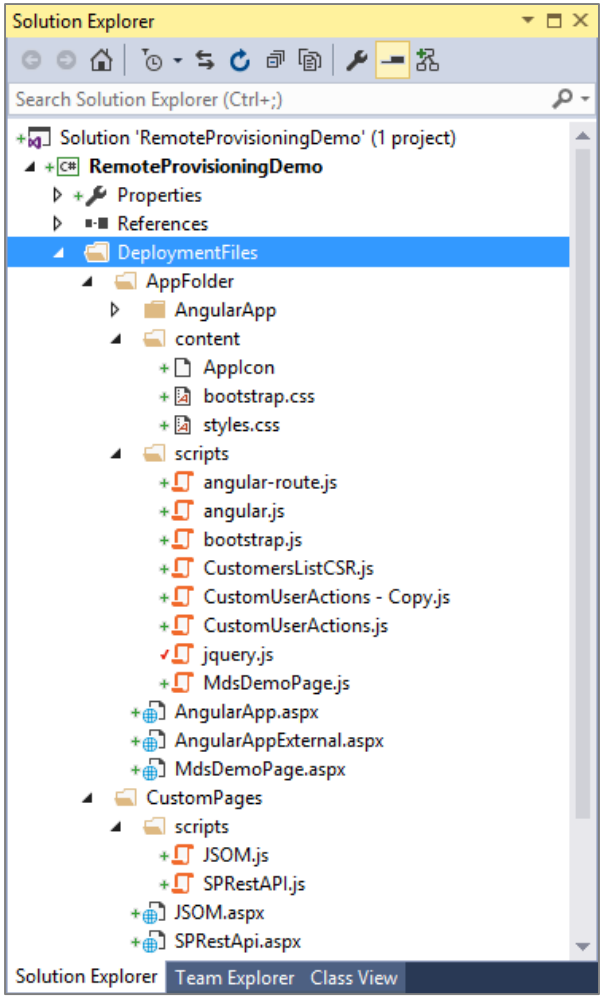
Remote Provisioning Demo Console App

- What does this sample app demonstrate?
  - Connects to an SPO site
  - Creates private folder at root of site
  - Uploads custom pages, scripts and style sheets
  - Sets Alternate CSS URL for the current site
  - Registers ScriptLinks for jQuery and custom script
  - Adds custom actions to site Actions menu
  - Creates and populates sample Customer list
  - Embeds an Angular app into SharePoint UX
  - Uses JSLink and custom client-side rendering



Uploading Pages and Scripts using CSOM

- Where can you upload custom pages and scripts?
  - Master Page Gallery
  - Style Library
  - Standard document library
  - New folder created at site root



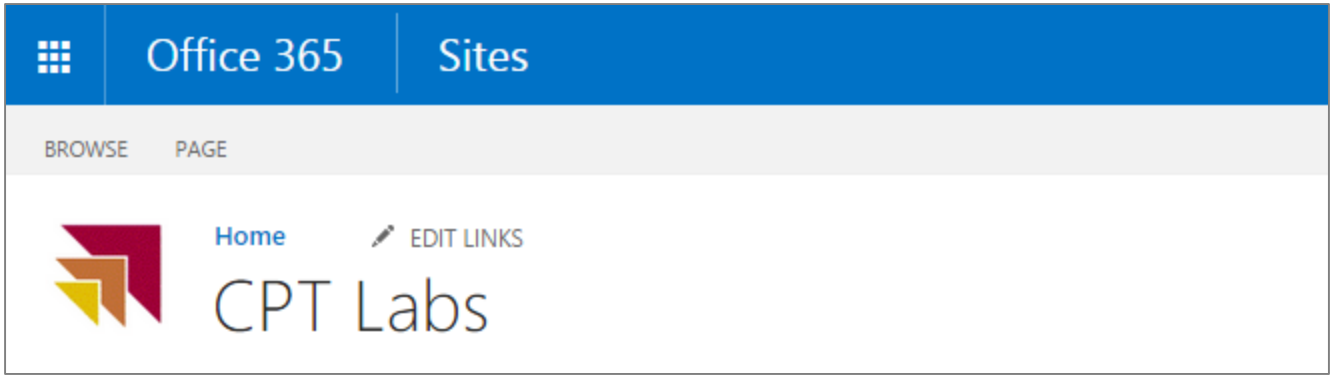
- Sample CSOM Code for uploading file

```
static void UploadToAppRootFolder(string path, byte[] content) {
  string filePath = AppRootFolderAbsoluteUrl + path;
  Console.WriteLine("Uploading to App Root Folder: " + path);
  FileCreationInformation fileInfo = new FileCreationInformation();
  fileInfo.Content = content;
  fileInfo.Overwrite = true;
  fileInfo.Url = filePath;
  File newFile = AppRootFolder.Files.Add(fileInfo);
  clientContext.ExecuteQuery();
}
```

AlternateCssUrl and Site Icon

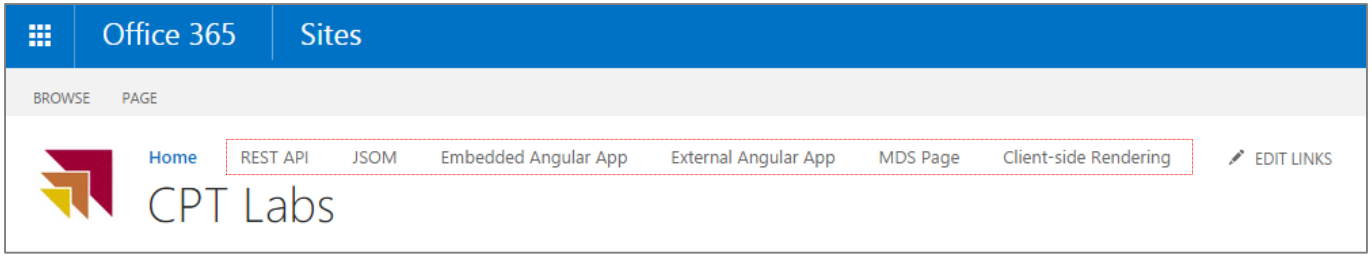
- Adding styling to an SPO Site
  - AlternateCssUrl links one style sheet to all pages in SPO site
  - SiteLogoUrl used to substitute custom site icon

```
static void SetAlternateCssAndSiteIcon() {
  site.AlternateCssUrl = AppRootFolderAbsoluteUrl + "content/styles.css";
  site.SiteLogoUrl = AppRootFolderAbsoluteUrl + "content/AppIcon.png";
  site.Update();
  clientContext.ExecuteQuery();
}
```



Creating Top Nav Nodes

- CSOM allows you to create Top Nav Nodes
  - Provides easy way to provide navigation to custom pages




```
static void CreateTopNavNode(string title, string path) {
  string nodeUrl = site.Url + path;
  NavigationNodeCreationInformation newNode = new NavigationNodeCreationInformation();
  newNode.IsExternal = false;
  newNode.Title = title;
  newNode.Url = nodeUrl;
  newNode.AsLastNode = true;
  TopNavNodes.Add(newNode);
  clientContext.ExecuteQuery();
}


static void ConfigureTopNav() {
  DeleteAllTopNavNodes();
  AddHomeTopNavNode();
  CreateTopNavNode("REST API", "/CustomPages/SPRestAPI.aspx");
  CreateTopNavNode("JSOM", "/CustomPages/JSOM.aspx");
  CreateTopNavNode("Embedded Angular App", "/CPT/AngularApp.aspx");
  CreateTopNavNode("External Angular App", "/CPT/AngularAppExternal.aspx");
  CreateTopNavNode("MDS Page", "/CPT/MdsDemoPage.aspx");
  CreateTopNavNode("Client-side Rendering", "/Lists/Customers");
}
```

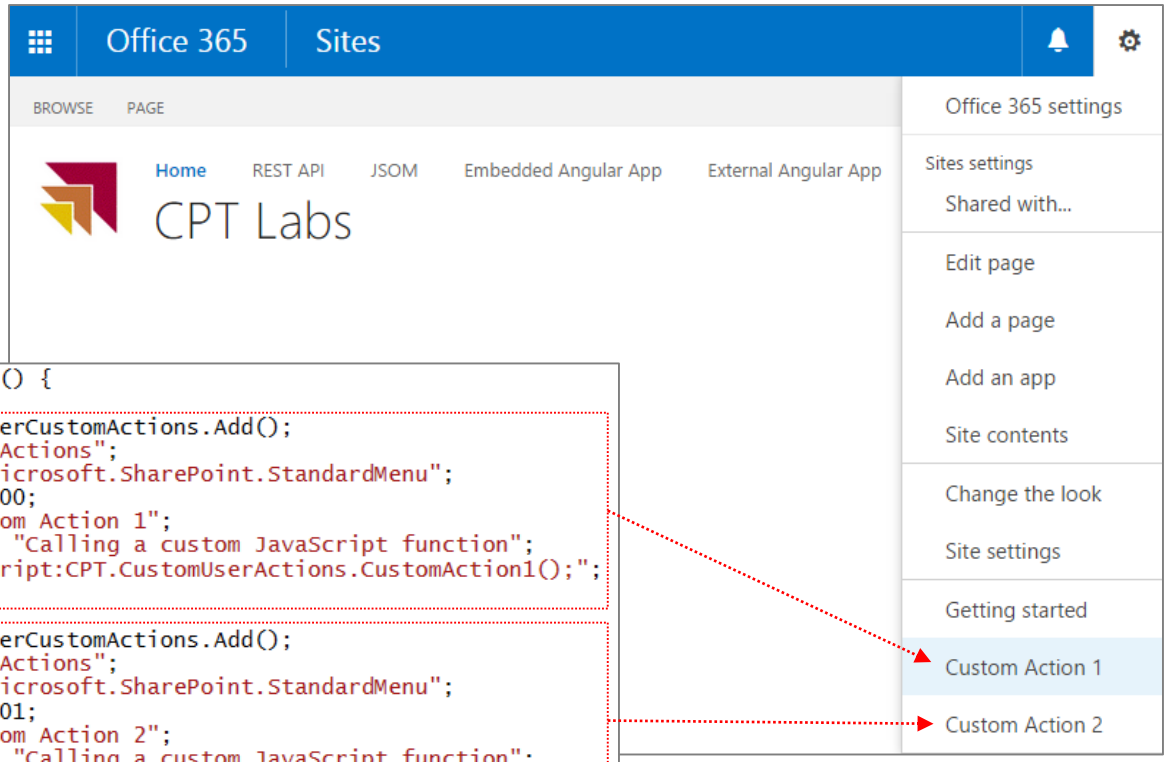
Adding ScriptLinks to Site

- ScriptLink added to site as UserCustomAction
  - Provides easy way to link all pages in site to common script file
  - Does not require modification to site's master page
  - Can be used to load common JavaScript libraries (e.g. jQuery)
  - Can be used to load custom scripts

```
static void CreateScriptLinks() {

  // Register ScriptLink for jQuery
  UserCustomAction customAction1 = site.UserCustomActions.Add();
  customAction1.Title = "jQuery";
  customAction1.Location = "ScriptLink";
  customAction1.ScriptSrc = "~SiteCollection/CPT/scripts/jquery.js";
  customAction1.Sequence = 10;
  customAction1.Update();

  // Register ScriptLink for custom javascript file
  UserCustomAction customAction2 = site.UserCustomActions.Add();
  customAction2.Title = "CustomUserActions";
  customAction2.Location = "ScriptLink";
  customAction2.ScriptSrc = "~SiteCollection/CPT/scripts/CustomUserActions.js";
  customAction2.Sequence = 11;
  customAction2.Update();

  clientContext.ExecuteQuery();
}
```

Adding Custom Actions to the SiteActions Menu

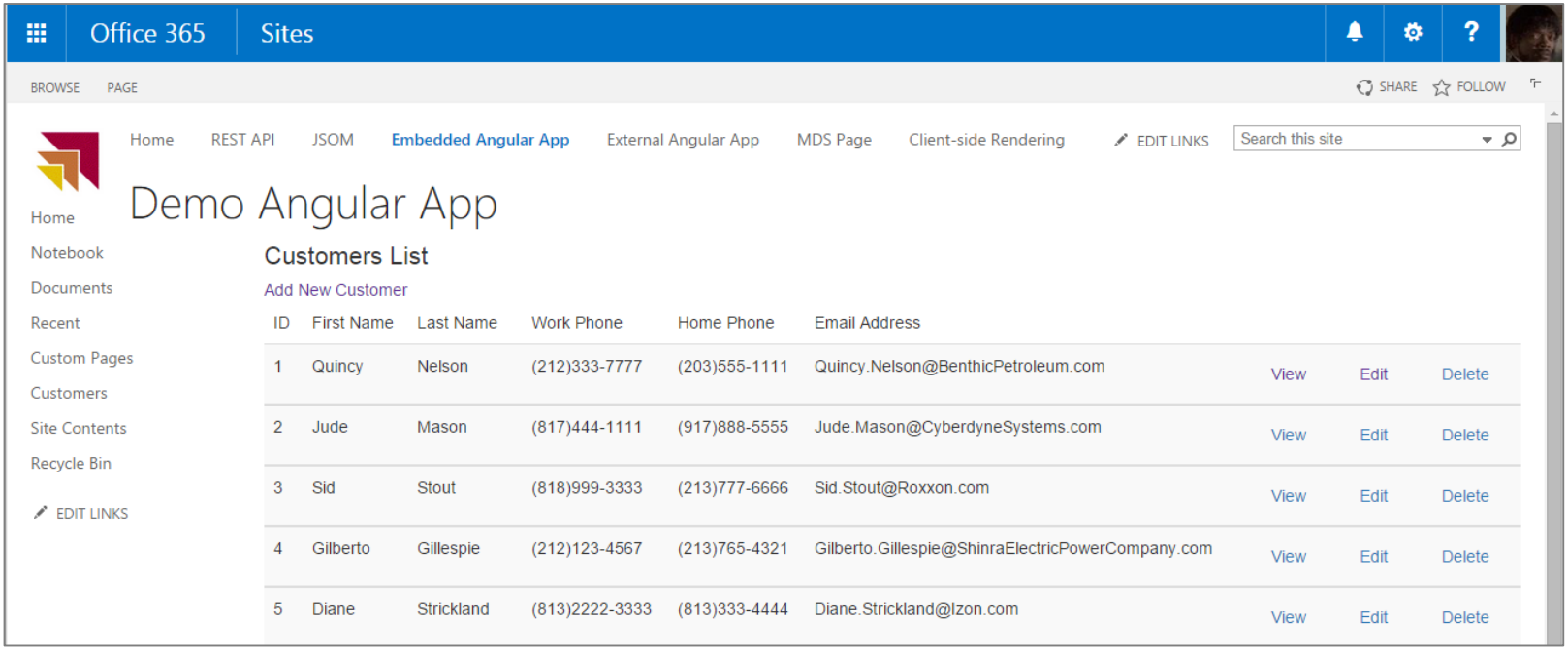
- Adding menu commands to SiteActions menu




```
static void CreateCustomSiteActions() {

  var siteActionsCommand1 = site.UserCustomActions.Add();
  siteActionsCommand1.Group = "SiteActions";
  siteActionsCommand1.Location = "Microsoft.SharePoint.StandardMenu";
  siteActionsCommand1.Sequence = 1000;
  siteActionsCommand1.Title = "Custom Action 1";
  siteActionsCommand1.Description = "Calling a custom JavaScript function";
  siteActionsCommand1.Url = "javascript:CPT.CustomUserActions.CustomAction1();";
  siteActionsCommand1.Update();

  var siteActionsCommand2 = site.UserCustomActions.Add();
  siteActionsCommand2.Group = "SiteActions";
  siteActionsCommand2.Location = "Microsoft.SharePoint.StandardMenu";
  siteActionsCommand2.Sequence = 1001;
  siteActionsCommand2.Title = "Custom Action 2";
  siteActionsCommand2.Description = "Calling a custom JavaScript function";
  siteActionsCommand2.Url = "javascript:CPT.CustomUserActions.CustomAction2();";
  siteActionsCommand2.Update();

  clientContext.ExecuteQuery();
}
```

Embedding an Angular App

- Angular apps can be injected using remote provisioning
  - Angular App can be embedded in SharePoint UU
  - Angular App can be designed external to SharePoint UI



Summary

- ✓ SharePoint Online Development Strategies
- ✓ Understanding Modern Team Site and Modern Pages
- ✓ Programming the Client-side Object Model (CSOM)
- ✓ Creating Site Columns, Content Types and Lists
- ✓ JavaScript Injection and the SharePoint REST API