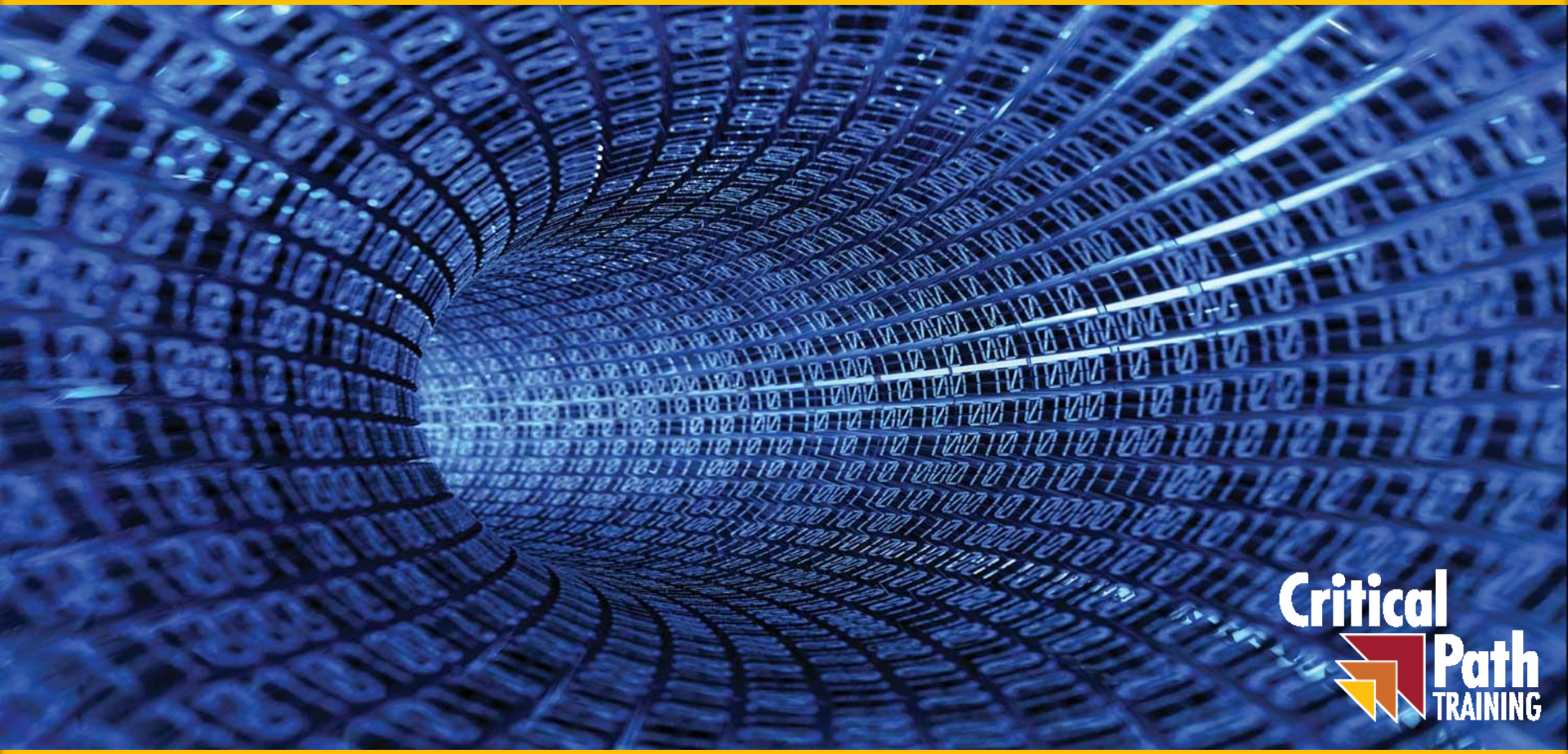
JavaScript Injection and Remote Provisioning



Agenda

- Enabling Scripting in SharePoint 2016
- Understanding JavaScript Injection
- JSOM Programming
- Remote Provisioning using CSOM
- Designing MDS-enabled Pages
- JSLink and Client-side Rendering

Scripting Capabilities in SharePoint Online

- SharePoint 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 **enabled** for
  - All sites in SharePoint On-premises
- The default scripting capabilities **disabled** for
  - Personal sites in SharePoint Online
  - Self-service created sites in SharePoint Online
  - Root site collection of the tenant in SharePoint Online

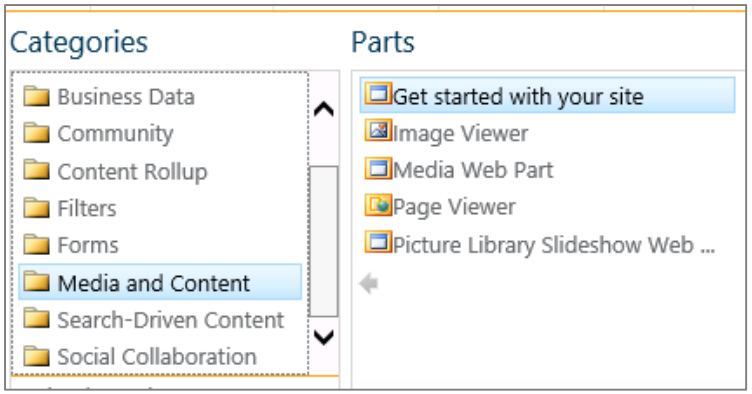
Features Affected with Scripting Disabled

- 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

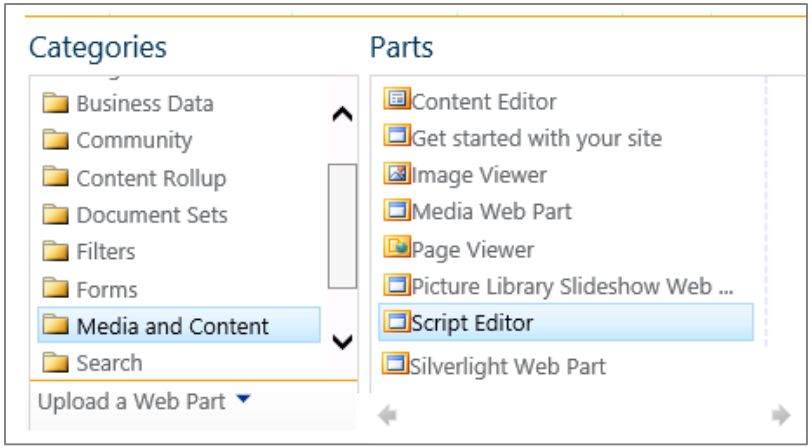
- Scripting must be enabled at the site level
  - Can be done by configuring SPO tenancy policy
  - Can be done using PowerShell or CSOM

Effects of Scripting Being Disabled

- Media and Content Web Parts (scripting disabled)

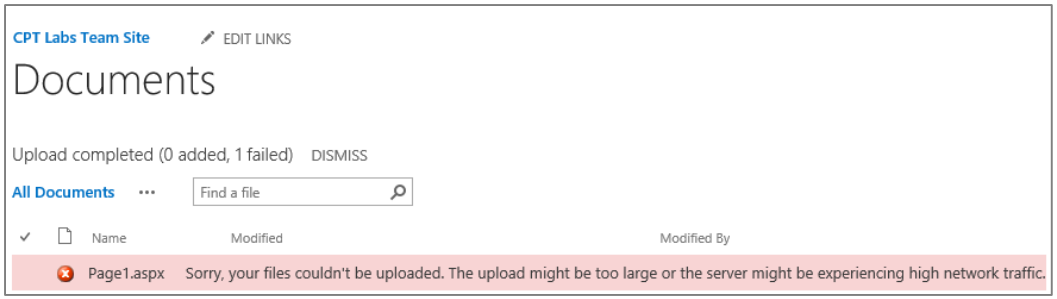


- Media and Content Web Parts (scripting enabled)



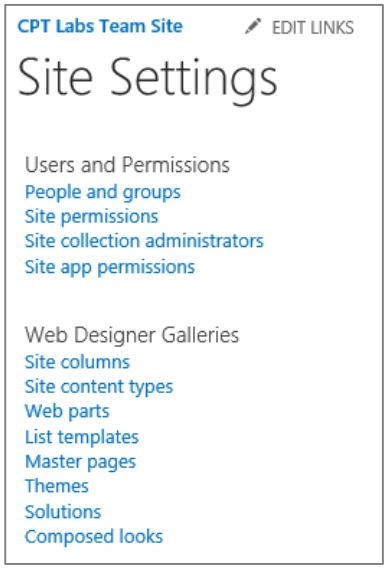
More Effects of Scripting Being Disabled

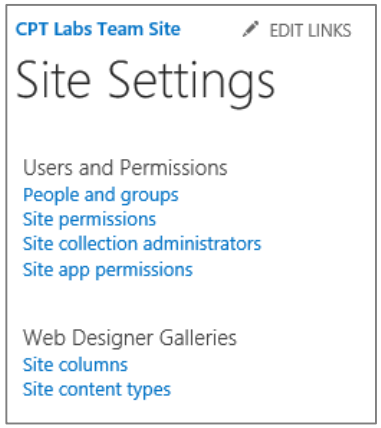
- You cannot upload a .ASPX file to a document library



- Many Administrative Links removed from Site Settings page

Scripting Enabled

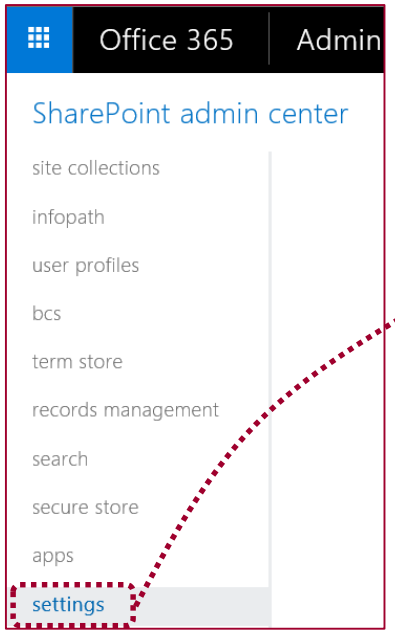


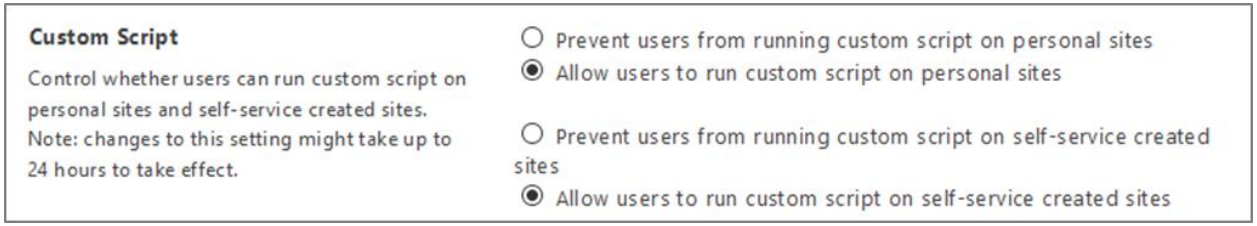


Scripting Disabled

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

```
# establish authenticated connection to tenant admin site collection
$credential = Get-Credential
Connect-SPOService -Url https://CptLabs-admin.sharepoint.com -Credential $credential


# enable scripting for a specific site collection
Set-SPOSite https://CptLabs.sharepoint.com -DenyAddAndCustomizePages 0

```

Agenda

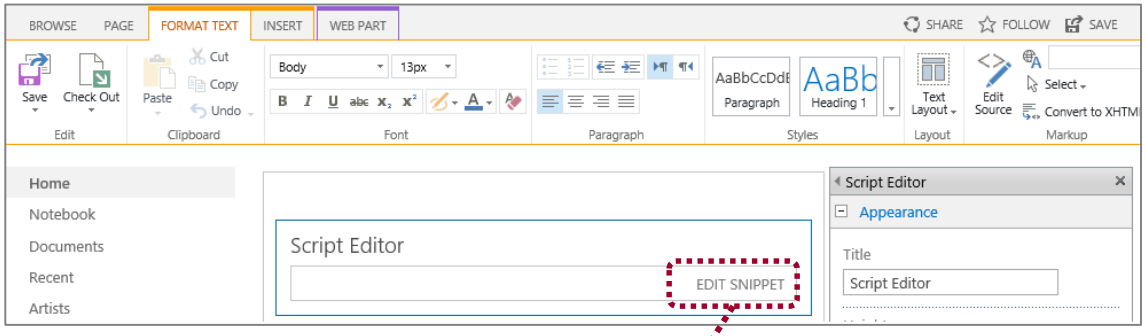
- ✓ Enabling Scripting in SharePoint 2016.
- ➢ Understanding JavaScript Injection
- JSOM Programming
- Remote Provisioning using CSOM
- Designing MDS-enabled Pages
- JSLink and Client-side Rendering

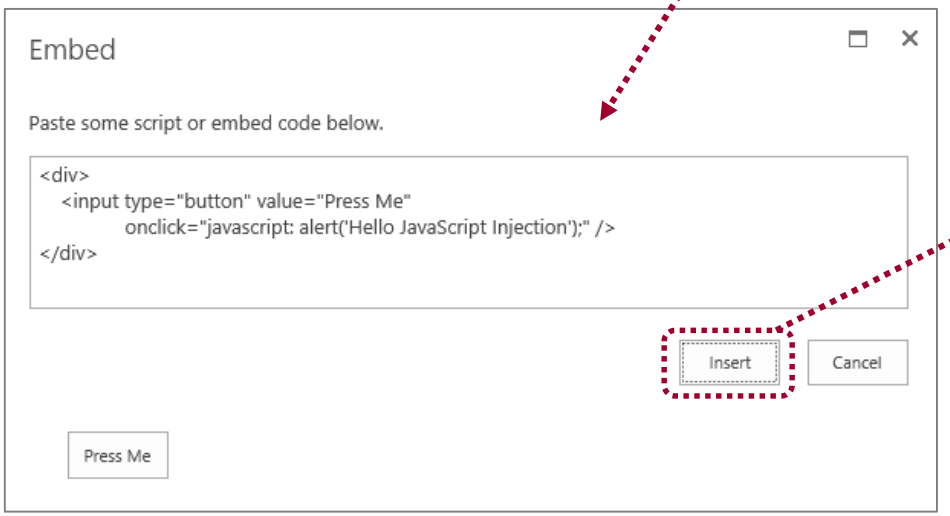
JavaScript Injection

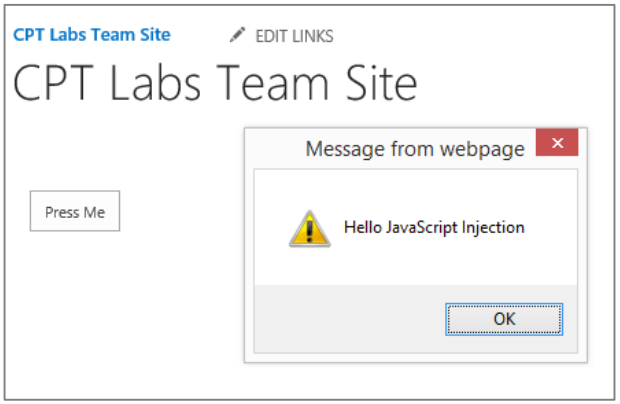
- JavaScript injection based on central concept…
  1. upload custom JavaScript code to SharePoint Online
  2. execute code using identity and permissions of current user
- Approaches for using JavaScript injection
  - Script Editor Web Part
  - Uploading Custom pages and JavaScript files
  - Use remote provisioning to deploy files to target site
- Why create solutions using JavaScript Injection?
  - Provides more flexibility than SharePoint add-in model
  - Poses fewer constraints than SharePoint add-in model

Script Editor Web Part

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







DEMO

Adding Custom Site Logic using the Script Editor Web Part

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

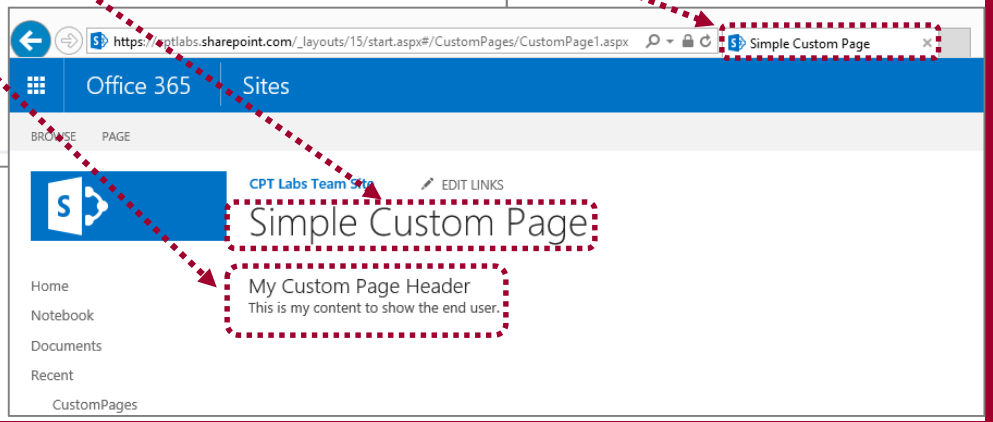
```
<% @Page MasterPageFile="~masterurl/default.master"
         Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage,Microsoft.SharePoint,Version=16.0

<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

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>
```



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))
      );
  });
}
```

Get Site Properties | Get Lists

- ID: 9bc612a2-9df4-44aa-8342-a0f87eb79379
- Title: CPT Labs Team Site
- Url: https://cptlabs.sharepoint.com

```
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);
  });
}
```

Get Site Properties | Get Lists

- CustomPages
- Documents
- Form Templates
- MicroFeed
- Site Assets
- Site Pages
- Style Library

Agenda

- ✓ Enabling Scripting in SharePoint 2016.
- ✓ Understanding JavaScript Injection
- ➢ JSOM Programming
- Remote Provisioning using CSOM
- Designing MDS-enabled Pages
- JSLink and Client-side Rendering

JavaScript Object Model (JSOM)

- SharePoint's Client-side JavaScript API
  - Provides equivalent to CSOM available to .NET code
  - Provides extra functionality for browser-based apps
  - Automatic authentication – no ability to impersonate
  - Requires execution within valid SharePoint context

- Core JSOM libraries
  - MicrosoftAjax.js
  - SP.Runtime.js
  - SP.js

The _spPageContextInfo Variable

- _spPageContextInfo
  - Culture/Locale information
  - Server-relative URL for site
  - Absolute URL for Site
  - Current page relative URL
  - Pages ListID
  - Web Title
  - Web UI Version

Scope | Watch


```
▼ _spPageContextInfo: Object
    alertsEnabled: true
    allowSilverlightPrompt: "True"
    clientServerTimeDelta: 2640
    crossDomainPhotosEnabled: true
    currentCultureName: "en-US"
    currentLanguage: 1033
    currentUICultureName: "en-US"
    env: "Prod"
    isAppWeb: false
    isSiteAdmin: true
    layoutsUrl: "_layouts/15"
    pageListId: "{16c8a1fd-3b20-4ad9-bfa8-80372c210a0a}"
    pagePersonalizationScope: 1
    serverRequestPath: "/Lists/Customers/AllItems.aspx"
    siteAbsoluteUrl: "https://cptlabs.sharepoint.com"
    siteClientTag: "0$$16.0.4524.1212"
    siteServerRelativeUrl: "/"
    systemUserKey: "i:0h.f|membership|10033fff93ace105@live.com"
    tenantAppVersion: "259279818"
  ▶ updateFormDigestPageLoaded: Wed Oct 14 2015 13:50:35 GMT-0400 (Eastern
    userId: 10
    userLoginName: "student@cptlabs.onmicrosoft.com"
    webAbsoluteUrl: "https://cptlabs.sharepoint.com"
    webLanguage: 1033
    webLogoUrl: "https://cptlabs.sharepoint.com/CPT/content/AppIcon.png"
  ▶ webPermMasks: Object
    webServerRelativeUrl: "/"
    webTemplate: "1"
    webTitle: "CPT Labs"
    webUIVersion: 15
```

The ctx Variable

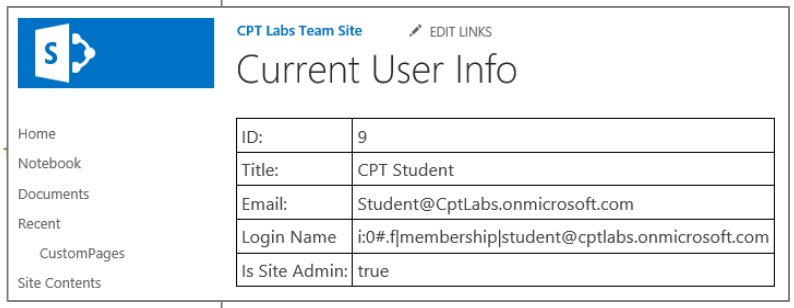
- ListData on ListViews
  - ListGuid, ListTitle, ListUrl
  - Row collection property
  - Column values
  - ContentTypeId
  - FSObjType (0=ListItem, 1=Folder)
  - displayFormUrl
  - editFormUrl
  - newFormUrl
  - ListSchema
  - IsDocLib
  - Field Information
  - LocaleID
  - PagePath


```
Scope  Watch
+  ↻
▶ _spPageContextInfo: Object
▼ ctx: a
    AllowGridMode: true
  ▶ BasePermissions: Object
    BaseViewID: 1
    CascadeDeleteWarningMessage: null
    ContentTypesEnabled: false
    ControlMode: 4
    CurrentCultureName: "en-US"
    CurrentItem: null
    CurrentItemIdx: -1
    CurrentLanguage: 1033
    CurrentSelectedItems: null
    CurrentUICultureName: "en-US"
    CurrentUserId: 10
    CurrentUserIsSiteAdmin: true
    EnableMinorVersions: false
    ExternalDataList: false
    HasRelatedCascadeLists: 0
    HttpPath: "https://cptlabs.sharepoint.com/_vti_
    HttpRoot: "https://cptlabs.sharepoint.com"
    IsAppWeb: false
    IsClientRendering: true
    LastSelectableRowIdx: null
  ▶ ListData: Object
    ListDataJSONItemsKey: "Row"
  ▶ ListSchema: Object
    ListTemplateType: 105
    ListTitle: "Customers"
    ModerationStatus: 0
```

Using SharePoint JavaScript Libraries

- Many SharePoint library files use lazy loading
  - Based on script-on-demand (SOD)
  - SPO library files downloaded using `ExecuteOrDelayUntilScriptLoaded`

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

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

  <script>

    // delay script execution until SOD file has downloaded
    ExecuteOrDelayUntilScriptLoaded(initializePage, "sp.js");

    function initializePage() {
      // create variables to track CSOM objects
      var context = SP.ClientContext.get_current();
      var user = context.get_web().get_currentUser();
      // use jQuery Document Ready event to call to CSOM and load user
      $(function () {
        context.load(user);
        context.executeQueryAsync(onDisplayUserInformation, onError);
      });
      // update page ith user information
      function onDisplayUserInformation() {
        var htmlTable = $("<table>");
        htmlTable.append(createTableRow("ID:", user.get_id()));
        htmlTable.append(createTableRow("Title:", user.get_title()));
        htmlTable.append(createTableRow("Email: ", user.get_email()));
        htmlTable.append(createTableRow("Login Name", user.get_loginName()));
        htmlTable.append(createTableRow("Is Site Admin:", user.get_isSiteAdmin()));
        $("#content_box").empty().append(htmlTable);
      }

      function createTableRow(name, value) {
        return $("<tr>").append($("<td>").text(name)).append($("<td>").text(value));
      }

      function onError(err, err_args) {
        alert('Failed to get user name. Error:' + err_args.get_message());
      }
    }
  </script>
```



Agenda

- ✓ Enabling Scripting in SharePoint 2016.
- ✓ Understanding JavaScript Injection
- ✓ JSOM Programming
- ➢ Remote Provisioning using CSOM
- Designing MDS-enabled Pages
- JSLink and Client-side Rendering

Deploying Custom Pages

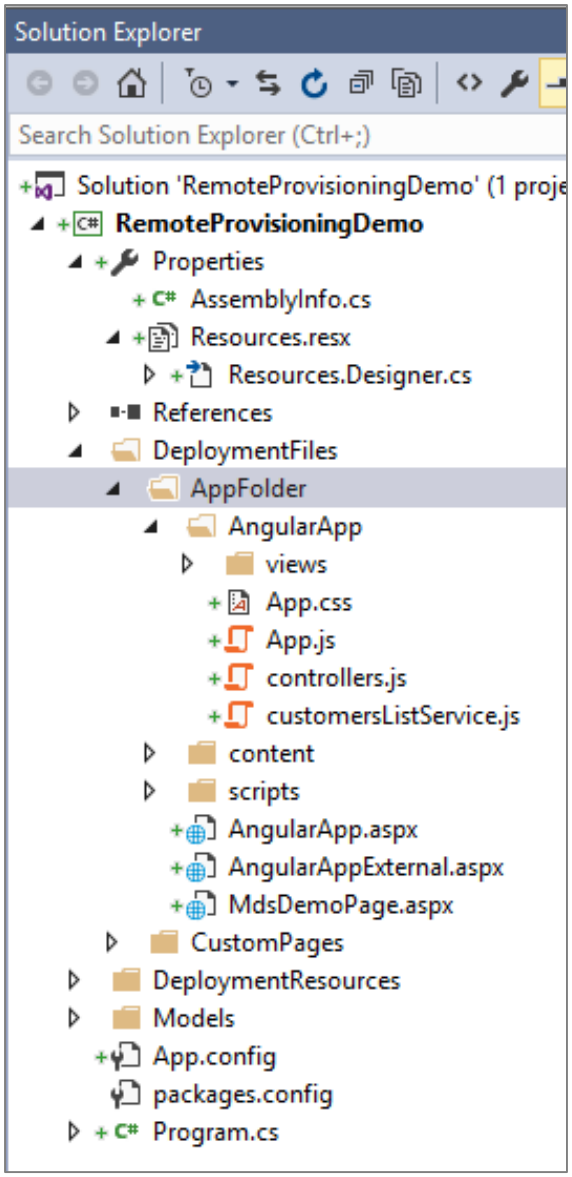
- Several approaches can be used to deploy custom pages
  - Upload pages to document library using browser
  - Upload pages to document library using Windows Explorer
  - Upload pages to document library using SharePoint Designer
  - Upload pages to document library using custom installer program
- Creating custom installer program benefits
  - Automates copying files to target SharePoint site
  - Eliminates copy-by-hand approach that is error-prone and tedious
  - Provides opportunity for more than just copying files
  - Provides opportunity to use CSOM for remote provisioning

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

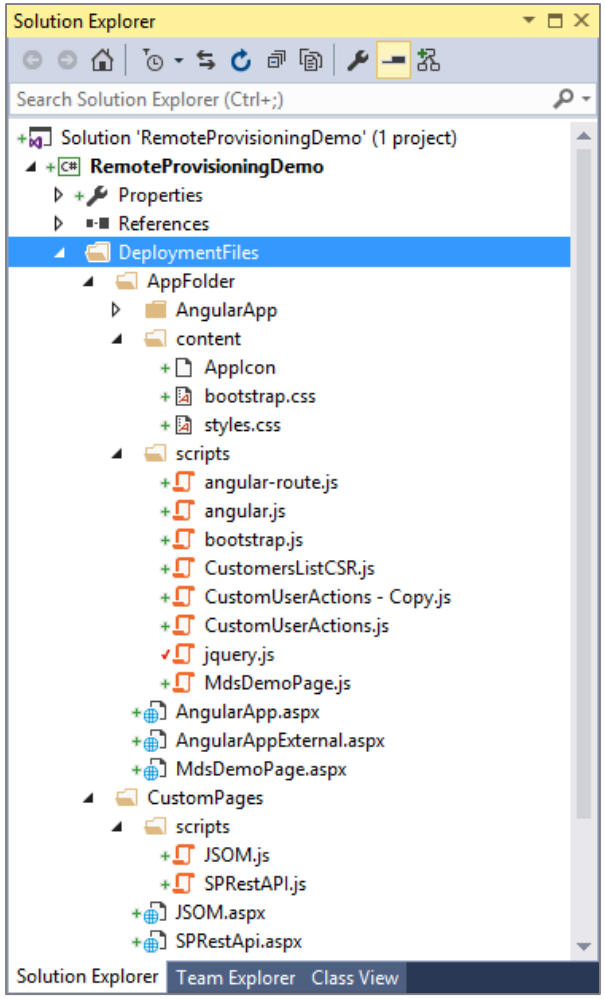
Remote Provisioning Demo Console App

- 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



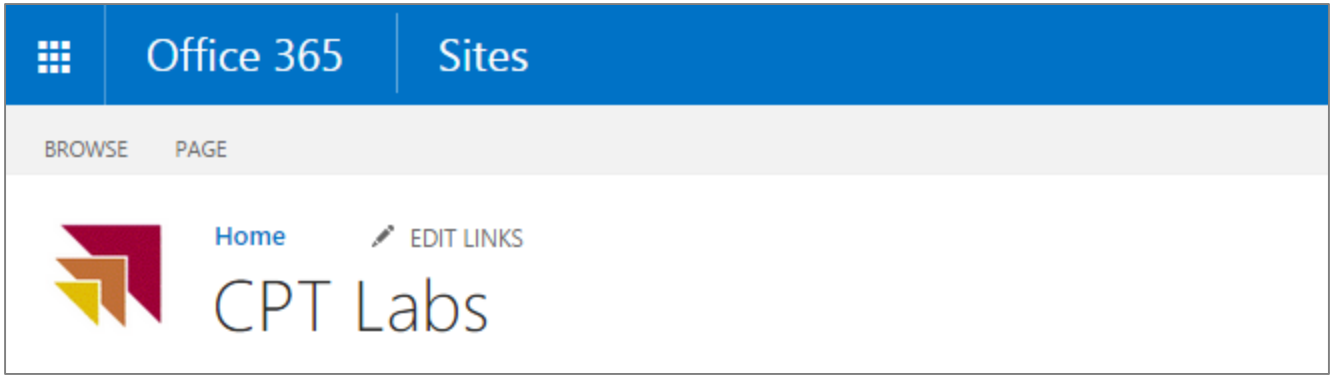
- Sample CSOM Code for uploading file

```
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

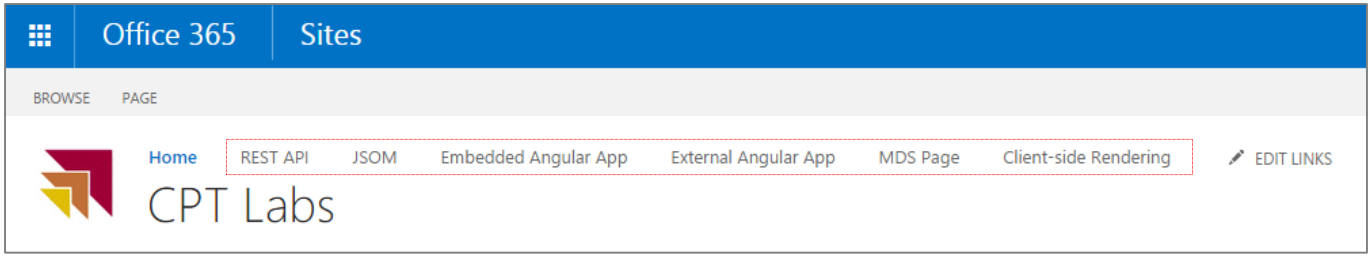
- Adding styling to an SPO Site
  - AlternateCssUrl links one style sheet to all pages in SPO site
  - SiteLogoUrl used to substitute custom site icon

```
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");
}
```

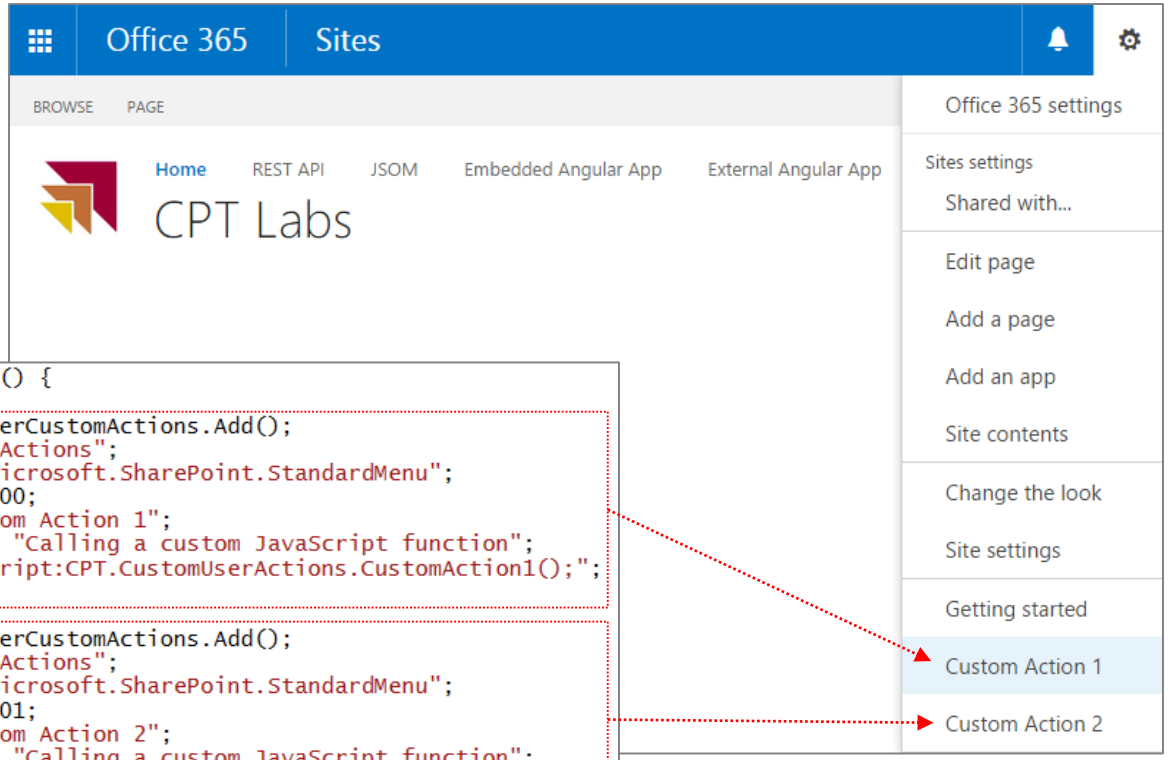
Adding ScriptLinks to Site

- 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

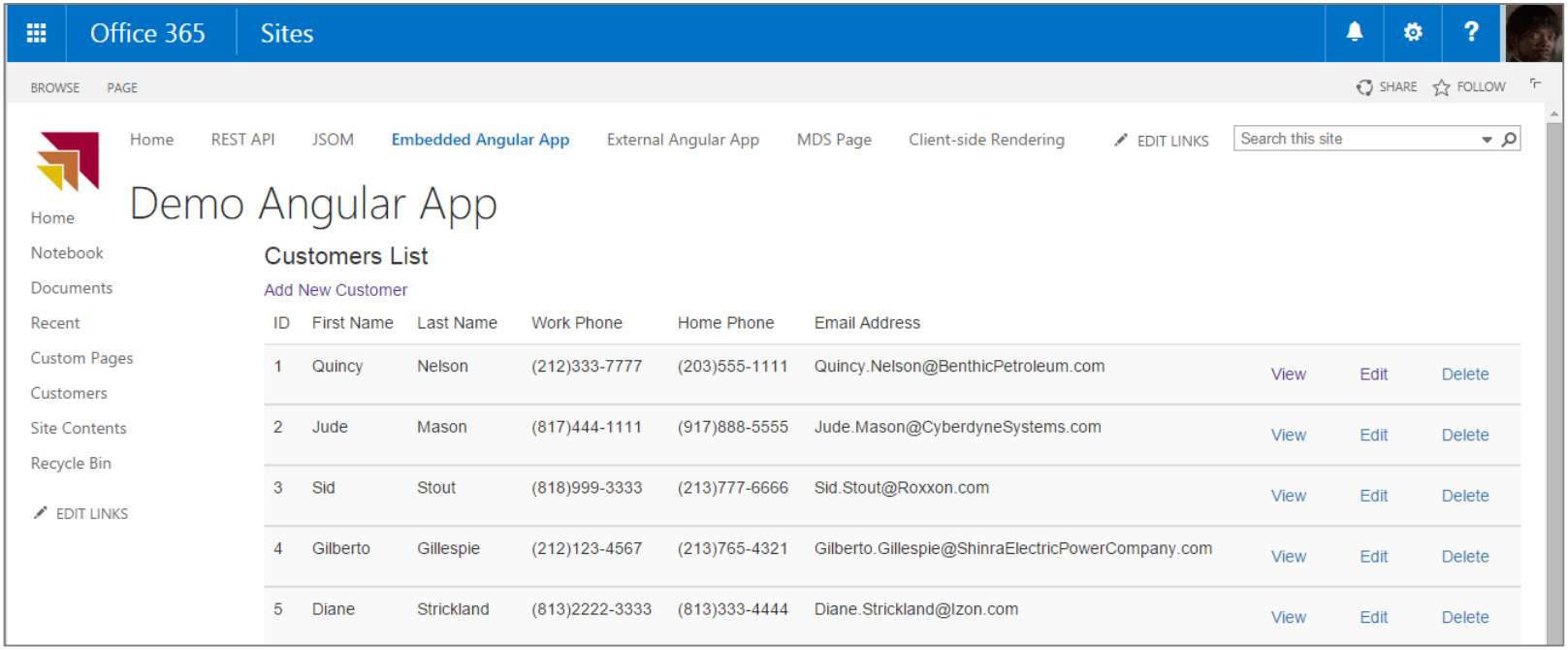
- Adding menu commands to SiteActions menu




```
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



DEMO

Remote Provisioning Demo Console App

Agenda

- ✓ Enabling Scripting in SharePoint 2016.
- ✓ Understanding JavaScript Injection
- ✓ JSOM Programming
- ✓ Remote Provisioning using CSOM
- ➢ Designing MDS-enabled Pages
- JSLink and Client-side Rendering

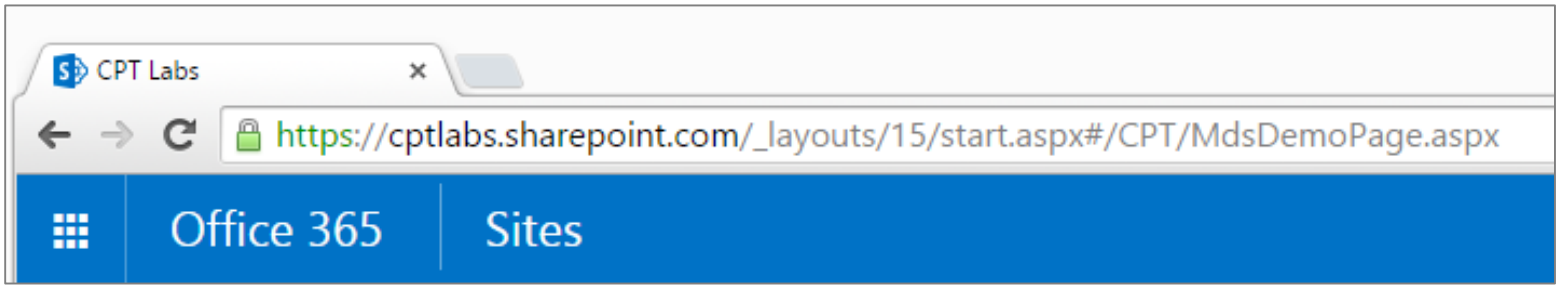
Minimal Download Strategy (MDS)

- MDS creates SPA architecture for SharePoint site
  - First page request loads MDS environment
  - Page transitions involve AJAX request for page deltas
  - Significantly complicates JavaScript programming
  - Incorrect code results in double page processing

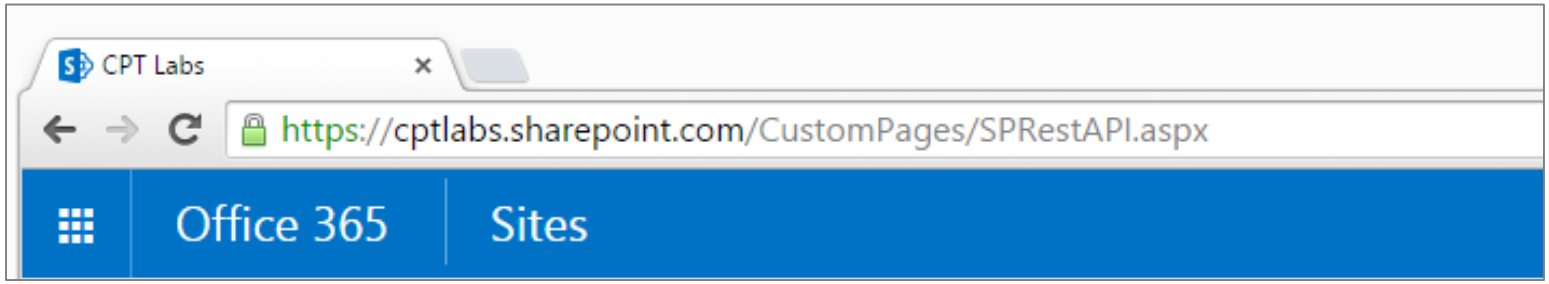
- When do you have to deal with MDS?
  - MDS enabled by default on Team sites
  - MDS can be disabled on Team sites
  - MDS not supported on Publishing Sites

Page URLs in MDS Mode

- When a site has MDS feature enabled…
  - MDS-enabled pages processed using MDS URLs



  - Pages not supporting MDS redirected to non-MDS URLs



Using the SharePoint:ScriptLink Control

- When designing pages for MDS mode
  - Do not use script tags because they break MDS mode
  - Use the SharePoint:ScriptLink control instead
  - Link to JavaScript files uploaded inside to the site

MdsDemoPage.aspx

```
<%@ Assembly Name="Microsoft.SharePoint,Version=16.0.0.0,Culture=neutral,PublicKeyToken=71e9

<%  @Page MasterPageFile="~masterurl/default.master"
          Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage" %>


<asp:Content ContentPlaceHolderID="PlaceHolderAdditionalPageHead" runat="server">
  <SharePoint:ScriptLink ID="Ajax" Name="MicrosoftAjax.js" runat="server" />
  <SharePoint:ScriptLink ID="jQuery" Name="~site/CPT/scripts/jquery.js" runat="server" />
  <SharePoint:ScriptLink ID="App" Name="~site/CPT/scripts/MdsDemoPage.js" runat="server" />
</asp:Content>


<asp:Content ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
  MDS Demo Page
</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>
```

What Happens When in MDS Mode?

- On the first page load
  - Processing occurs just like non-MDS page load – all events fire
- On subsequent MDS page transitions
  - JavaScript files are not reparsed
  - Inline code does not execute
  - jQuery document ready event handler does not run
  - SharePoint load event handler does not run
  - MDS performs garbage collection to purge global variables

MDS Garbage Collection

- MDS performs garbage collection
  - Global variables are purged during page transitions
  - MDS garbage collection prevents memory bloat
  - MDS garbage collection requires your attention

- Creating global variables that survive garbage collections
  - Create global variable by registering it as an ASP.NET namespace

```
// create global variable with standard JavaScript approach
var CptCanary = window.CptCanary || {};
CptCanary.greeting = "tweet, tweet";


// create global variable by registering namespace with ASP.NET AJAX
Type.registerNamespace("CPT");
CPT.greeting = "Hi, I can live across MDS page transitions";
```

Managing MDS and Non-MDS Page Loads

```
// create global variable by registering namespace with ASP.NET AJAX
Type.registerNamespace("CPT");

CPT.MdsDemoPage = function () {
  console.log("executing main MdsDemoPage function");

  var initializeMDS = function ()...;

  var onMdsAfter = function (source, args)...;

  var registerEventHandlers = function ()...

  var onGetSiteProperties = function ()...

  var onGetLists = function ()...

  // public interface
  return {
    initializeMDS: initializeMDS,
    registerEventHandlers: registerEventHandlers
  };

}();


if (typeof asyncDeltaManager !== 'undefined') {
  console.log("Initialize page in MDS mode");
  CPT.MdsDemoPage.initializeMDS();
}
else {
  console.log("Initialize page in standard (non-MDS) mode");
  $(CPT.MdsDemoPage.registerEventHandlers);
}
```

Understanding MDS Events

- MDS events occur during page transitions
  - Request Initialized
  - Begin Request
  - End Request

- `asyncDeltaManager` used to add event handlers
  - `add_initializeRequest`
  - `add_beginRequest`
  - `add_endRequest`

Handling MDS Events on Page Transitions

```
if (typeof asyncDeltaManager !== 'undefined') {
  console.log("Initialize page in MDS mode");
  CPT.MdsDemoPage.initializeMDS();
}
else {
  console.log("Initialize page in standard (non-MDS) mode");
  $(CPT.MdsDemoPage.registerEventHandlers);
}
```

```
var initializeMDS = function () {
  console.log("executing initializeMDS function");
  if (CPT.MdsDemoPage.MdsEnabled == undefined) {
    // add handler for MDS page transitions
    asyncDeltaManager.add_endRequest(onMdsAfter);
    // set variable to inidicate event handler has been registered
    CPT.MdsDemoPage.MdsEnabled = true;
  }
};

var onMdsAfter = function (source, args) {
  console.log("onMdsAfter handler executing");
  var currentPage = source._currentUrl;
  console.log("current url: " + currentPage);
  var currentPageIsMdsDemoPage = (currentPage.indexOf("MdsDemoPage.aspx") > -1);
  if (currentPageIsMdsDemoPage) {
    registerEventHandlers();
  }
};

var registerEventHandlers = function () {
  console.log("registerEventHandlers executing");
  // register event handlers
  $("#getSiteProperties").click(onGetSiteProperties);
  $("#getLists").click(onGetLists);
}
```

Agenda

- ✓ Enabling Scripting in SharePoint 2016.
- ✓ Understanding JavaScript Injection
- ✓ JSOM Programming
- ✓ Remote Provisioning using CSOM
- ✓ Designing MDS-enabled Pages
- ✓ JSLink and Client-side Rendering

Attaching Custom Scripts using JSLink

- JSLink property can attach custom script to
  - List Views
  - List Forms (e.g. New / Edit / Display forms)
  - List View Web Parts and List Form Web Parts
  - Site Columns and Content Types

```
string listTitle = "Customers";
string listUrl = "Lists/Customers";

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

ListCreationInformation lci = new ListCreationInformation();
lci.Title = listTitle;
lci.Url = listUrl;
lci.TemplateType = (int)ListTemplateType.Contacts;
listCustomers = site.Lists.Add(lci);
listCustomers.OnQuickLaunch = true;
listCustomers.Update();

// attach JSLink script to default view for client-side rendering
listCustomers.DefaultView.JSLink = AppRootFolderRelativeUrl + "scripts/CustomersListCSR.js";
listCustomers.DefaultView.Update();
listCustomers.Update();
clientContext.Load(listCustomers);
clientContext.Load(listCustomers.Fields);
clientContext.ExecuteQuery();
```

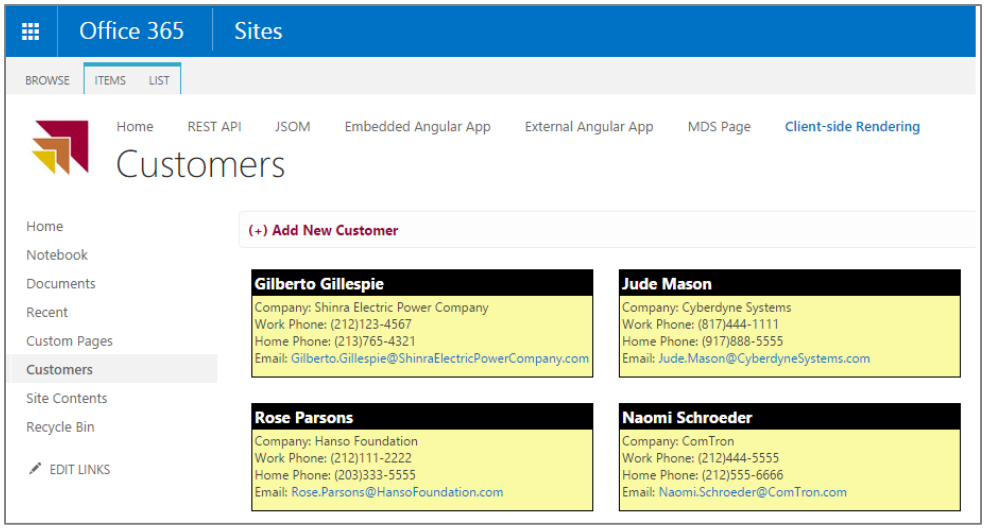
Custom Client-side Rendering (CSR)

CustomersListCSR.js

<global>

```
"use strict";

// determine path to the current script
var pathToThisScript = _spPageContextInfo.siteServerRelativeUrl +
                       "CPT/scripts/CustomersListCSR.js";

// Register function named RegisterTemplate to execute on MDS events
RegisterModuleInit(pathToThisScript, RegisterTemplate);

// call RegisterTemplate when site not running in MDS mode
RegisterTemplate();

function RegisterTemplate() {
  // create new custom template object
  var overrideCtx = {};
  overrideCtx.ListTemplateType = 105;
  overrideCtx.OnPreRender = onPreRender;
  overrideCtx.OnPostRender = onPostRender;
  overrideCtx.Templates = {};
  overrideCtx.Templates.Header = customHeader;
  overrideCtx.Templates.Item = customItem;

  // register custom template with SharePoint Template Manager
  SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
}

function customHeader(ctx)...

function customItem(ctx)...

function onPreRender(ctx)...

function onPostRender(ctx)...
```

Implementing Custom Rendering Logic




```
function RegisterTemplate() {
  // create new custom template object
  var overrideCtx = {};
  overrideCtx.ListTemplateType = 105;
  overrideCtx.OnPreRender = onPreRender;
  overrideCtx.OnPostRender = onPostRender;
  overrideCtx.Templates = {};
  overrideCtx.Templates.Header = customHeader;
  overrideCtx.Templates.Item = customItem;

  // register custom template with SharePoint Template Manager
  SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);
}
```

```
function customHeader(ctx) {

  return "<div class='csrHeaderDiv'>" +
           "<a href='" + ctx.newFormUrl + "' >(+) Add New Customer</a>" +
         "</div>";
}
```

```
function customItem(ctx) {

  var itemHTML = "<div class='csr1CustomerBlock' >" +
                   "<div class='csr1CustomerBlockHeader' >" +
                     ctx.CurrentItem.FirstName + " " + ctx.CurrentItem.Title +
                   "</div>" +
                   "<div class='csr1CustomerBlockBody' >" +
                     "Company: " + ctx.CurrentItem.Company +
                     "<br/>" +
                     "Work Phone: " + ctx.CurrentItem.WorkPhone +
                     "<br/>" +
                     "Home Phone: " + ctx.CurrentItem.HomePhone +
                     "<br/>" +
                     "Email: " + ctx.CurrentItem.Email +
                 "</div>" +
               "</div>";

  return itemHTML;
}
```

Summary

- ✓ Enabling Scripting in SharePoint 2016.
- ✓ Understanding JavaScript Injection
- ✓ JSOM Programming
- ✓ Remote Provisioning using CSOM
- ✓ Designing MDS-enabled Pages
- ✓ JSLink and Client-side Rendering