# Demo 1: Hello, Apps

In this Demo, you will create a simple SharePoint App using Visual Studio. This Demo will familiarize you with the development process, toolset, and patterns for simple App development.

## Step 1 – Create a New App

In this Step, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. **Open** Microsoft Visual Studio 2012
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for SharePoint 2013** project template
   4. Name the new project **HelloApp**
   5. Click the **OK** button
   6. In the **SharePoint Customization Wizard**, name the new App **Hello, App**.
   7. Specify the site you will use for this lab.
   8. Select **SharePoint-hosted** as the hosting type.
   9. Click **Finish**.
2. Code the App
   1. Open **Pages\Default.aspx** for editing.
   2. **Add** the code below within the **PlaceHolderMain** Content Placeholder control:

<div id="displayDiv"></div>

<input type="button" value="Push me!" onclick="hello();"/>

* 1. Open **Scripts\App.js** for editing.
  2. Add the following code to the library:

function hello() {

$get("displayDiv").innerHTML = "<p>Hello, Apps!</p>";

}

## Step 2 – Run the App in Debug mode

In this Step, you will run the App and see it hosted in SharePoint.

1. Run the App
   1. In Visual Studio, select **Debug⮚Start Debugging**, observe the Output Window for installation messages.
   2. Launch the App
   3. In the App, click the **Push Me** button.
2. Shutdown the App
   1. In Visual Studio, select **Debug⮚Stop Debugging**, observe the Output Window for retraction messages.

## Step 3 – Add a Client Web Part to the App

In this Step, you will add a Client Web Part and code it. This web part can be used on pages outside of the App to add functionality to the hosting SharePoint site.

1. Add a Client Web Part
   1. In the **Solution Explorer**, right click the **HelloApp** project node and select **Add⮚New Item** from the context menu.
   2. In the **Add New Item** dialog, select **Client Web Part (Host Web)**.
   3. Name the new item **AppPart** and click **Add**.
   4. In the Create Client Web Part dialog, choose **Create a new client web part page** and give it the name **AppPartSource**.
   5. **Replace** the entire contents of the **Elements** file with the following:

<?xml version="1.0" encoding="utf-8"?>

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

<ClientWebPart Name="AppPart"

Title="App Part"

Description="A simple client web part"

DefaultWidth="200"

DefaultHeight="200">

<Content

Type="html"

Src="~appWebUrl/Pages/AppPartSource.aspx?

Message=\_DisplayMessage\_" />

<Properties>

<Property Name="DisplayMessage"

Type="string"

WebBrowsable="true"

WebDisplayName="Display Message"

WebDescription="A message to display"

WebCategory="Configuration"

DefaultValue="Hello, Client Web Part!"

RequiresDesignerPermission="true" />

</Properties>

</ClientWebPart>

</Elements>

1. In the **Pages** folder, open the item **AppPartSource.aspx. Replace** the contents below the page directives with the following:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head id="Head1" runat="server">

<title>Client Web Part</title>

<WebPartPages:AllowFraming ID="AllowFraming" runat="server" />

<SharePoint:ScriptLink ID="ScriptLink" name="sp.js"

runat="server" OnDemand="true" LoadAfterUI="true"

Localizable="false" />

<script type="text/javascript" src="../Scripts/App.js"></script>

</head>

<body>

<form id="form1" runat="server">

<div>

<div id="appPartDiv"></div>

<input type="button" value="Push Me!" onclick="helloAppPart();" />

</div>

</form>

</body>

</html>

1. Open **Scripts\App.js** for editing.
2. **Add** the following code to the library:

function helloAppPart() {

var message = getQueryStringValue("Message");

document.getElementById("appPartDiv").innerHTML = "<p>" + message + "</p>";

}

function getQueryStringValue(paramName){

var params = document.URLUnencoded.split("?")[1].split("&");

var strParams = "";

for (var i = 0; i < params.length; i = i + 1) {

var singleParam = params[i].split("=");

if (singleParam[0] == paramName)

return decodeURIComponent(singleParam[1]);

}

}

## Step 4 – Add the Client Web Part to a Page

In this Step, you will run the App and add the Client Web Part to the home page of the portal.

1. Deploy the App
   1. In Visual Studio, right click the **HelloApp** node and select **Deploy** from the context menu.
   2. When the App is deployed, open your browser to the home page of the SharePoint site hosting the App (not the home page of the App itself!).
2. Add the Client Web Part
   1. In the ribbon, select **Page⮚Edit Page**.
   2. Click **Add a Web Part** in any Zone.
   3. Add the **AppPart** Web part to the page.
   4. Click the **Push Me** button to test the functionality.
3. Retract the App
   1. In Visual Studio, right click the **HelloApp** node and select **Retract** from the context menu.
   2. Note that the **Client Web Part** is removed from the home page.

## Step 5 – Add a button to the Ribbon

In this Step, you will add a custom action to the ribbon of all Document Libraries to launch the App.

1. In Visual Studio, right click the **HelloApp** node and select **Add⮚New Item** from the Context menu.
2. In the **New Item** dialog, select **Ribbon Custom Action**.
3. Name the item **Launcher** and click **Add**.
4. In the Create Custom Action for Ribbon window, choose to expose the custom action to the **Host Web.** The Custom Action should be scoped to **List Template**, and the particular item the custom action is scoped to is **Document Library**.
5. In the next window, we will specify settings to generate a button control for the ribbon. Use the following settings:
   1. Where is the control located? **Ribbon.Documents.Manage**
   2. What is the label text for the button control? **Go!**
   3. Where does the button control navigate to? **HelloApp\Pages\Default.aspx**

## Step 6 – Launch an App from the Ribbon

In this Step, you will deploy the App and use the button on the Ribbon to launch it.

1. In Visual Studio, right click the **HelloApp** node and select **Deploy** from the Context menu.
2. After the App is deployed, open the home page of the portal where the App is hosted (not the App itself!).
3. On the home page, click **Site Contents**.
4. Click **Add an App**.
5. Select to **Add a Document Library**.
6. Name the Document Library **Shared Documents**.
7. On the Apps page, click the newly-created **Shared Documents** library.
8. In the ribbon, click the **Files** tab.
9. **Launch** the App using the button you created.

# Demo 2: Membership App

In this Demo, you will create a SharePoint App that accesses the User Information list of the hosting site. This Demo will help you understand how to use request permissions in an App and access lists in the hosting SharePoint web. You will also make use of jQuery in the App.

## Step 1 – Create a New App

In this Step, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. **Open** Microsoft Visual Studio 2012
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for SharePoint 2013** project template
   4. Name the new project **MembershipApp**
   5. Click the **OK** button
   6. In the **New App for SharePoint Wizard**, name the new App **Membership App**.
   7. In the **New App for SharePoint Wizard**, specify the site you created prior to starting the lab for hosting Apps.
   8. Select **SharePoint-hosted** as the hosting type.
   9. Click **Finish**.
2. Code the App
   1. Open **Default.aspx** in Visual Studio for editing
   2. **Add** the following code in the **PlaceHolderMain** Content Placeholder control with the following:

<div id="peopleDiv"></div>

* 1. **Open** **App.js** for editing.
  2. **Add** the following code to **App.js** to query the User Information list of the hosting site:

$(document).ready(function () {

Membership = {

element: '',

url: '',

init: function (element) {

Membership.element = element;

Membership.url = \_spPageContextInfo.webAbsoluteUrl + "/\_api/site/rootweb/lists/getByTitle('User%20Information%20List')/items?$select=Title,Name";

},

load: function () {

$.ajax(

{

url: Membership.url,

method: "GET",

headers: {

"Accept": "application/json;odata=verbose",

},

success: Membership.onSuccess,

error: Membership.onError

}

);

},

onSuccess: function (data) {

var results = data.d.results;

var html = "<table>";

for (var i = 0; i < results.length; i++) {

html += "<tr><td>";

html += results[i].Title;

html += "</td><td>"

html += results[i].Name;

html += "</td><tr>";

}

html += "</table>";

Membership.element.html(html);

},

onError: function (err) {

alert(JSON.stringify(err));

}

}

Membership.init($('#peopleDiv'));

Membership.load();

});

* 1. Open the **AppManifest.xml** designer.
  2. In the **Permission Requests** table, select **Site Collection** as the Scope.
  3. Select **Read** as the Permission.

## Step 2 – Run the App in Debug mode

In this Step, you will run the App and see it hosted in SharePoint.

1. Run the App
   1. Select **Debug⮚Start Debugging** in Visual Studio, observe the **Output Window** for installation messages, and verify that **Internet Explorer** opens to the SharePoint Apps site you created before starting the lab.
   2. When prompted by SharePoint to grant permissions, click **Trust It**.
   3. From the Apps page, launch the Membership App.
   4. You should now see a list of all the members of the Site Collection hosting the app.

# **Demo 3: List-Powered Apps**

In this Demo, you will create a SharePoint App that uses SharePoint lists. This Demo will familiarize you with basic SharePoint concepts and how they can be used in Apps.

## Step 1 – Create a New App

In this Step, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. **Open** Microsoft Visual Studio 2012
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for SharePoint 2013** project template
   4. Name the new project **ListApp**
   5. Click **OK**.
   6. In the **SharePoint Customization Wizard**, name the App, **List App**.
   7. Specify the site you will use for this lab.
   8. Select **SharePoint-hosted** as the hosting type.
   9. Click **Finish**.
2. In the **Solution Explorer**, Right click on the **ListApp** node, select Add, select **Add New Item.**
3. In the **Add New Item** dialog, select **List**.
4. Enter **QuickStatus** for the name and click **Add**.
5. In the Choose List Settings dialog, change the display name to **Quick Status.**
6. Click **Finish**.
7. In the **List Schema Designer**, add a field called **Priority**, and change the Type to **Choice**.
8. Code the App.
   1. Open **Content/App.css** in Visual Studio for editing
   2. **Add** the following:

.messages

{

margin: 0px;

padding: 0px;

float: left;

border-top: 1px #D8D8D0;

width: 300px;

}

.messages tr

{

padding: 5px 5px 10px 10px;

list-style-type: none;

background-color: #FAFAF4;

color: #000033;

}

.subtle

{

color: #999999

}

* 1. Open **Scripts/App.js** in Visual Studio for editing
  2. **Add** the following code:

var \_surlWeb = "";

$(document).ready(function()

{

\_surlWeb = \_spPageContextInfo.webServerRelativeUrl;

if (\_surlWeb.length > 0 && \_surlWeb.substring(

\_surlWeb.length - 1, \_surlWeb.length ) != "/") {

\_surlWeb += "/"; }

loadMessages();

});

function loadMessages()

{

$.ajax({

url: \_surlWeb +

"\_api/lists/getbytitle('Quick Status')/items?$select=ID,Title,Author/Title&$expand=Author&$orderby=ID desc",

headers: {

"accept": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val()

},

success: postMessageListRetrieve,

error: oops

});

}

function postMessageListRetrieve(data)

{

var items = [];

items.push("<table class='messages'>");

$.each(data.d.results, function(key, val)

{

items.push('<tr id="' + val.ID + '"><td><div>' +

val.Title + '</div><div class="subtle">' +

val.Author.Title + '</div></td></tr>');

});

items.push("</table>");

$('#messages').html(items.join(''));

}

function oops(data) {

alert(data.responseText);

}

function addMessage()

{

$.ajax({

url: \_surlWeb + "\_api/lists/getbytitle('Quick Status')/items",

type: "POST",

data: JSON.stringify(

{ '\_\_metadata': {

'type': 'SP.Data.Quick\_x0020\_StatusListItem'},

'Title': $('#messageInput').val()

}),

headers: {

"Content-Type" : "application/json;odata=verbose",

"accept": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val()

},

success: loadMessages,

error: oops

});

}

* 1. Open **Pages/Default.aspx** for editing
  2. **Add** the following within the **PlaceHolderMain** Content Placeholder control:

<div>

<div id="messageInputArea">

<input type="text" id="messageInput"></input>

<input type="button" value="Add"

onclick="addMessage()"></input>

</div>

<div id="messages">

Please wait, loading...

</div>

</div>

## Step 2 – Run the App in Debug mode

In this Step, you will run the App and see it hosted in SharePoint.

1. Run the App
   1. Select **Debug⮚Start Debugging** in Visual Studio, observe the **Output Window** for installation messages, and verify that **Internet Explorer** opens to the SharePoint Apps site you created before starting the lab.
   2. From the Apps page, launch the **List App**.
   3. Try adding new items to the list using the App.

# **Demo 4: The Chrome Control**

In this Demo, you will create a SharePoint App that uses the chrome control. The chrome control is used to brand a SharePoint App similarly to the hosting SharePoint web.

## Step 1 – Create a New App

In this Step, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. **Open** Microsoft Visual Studio 2012
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for SharePoint 2013** project template
   4. Name the new project **ChromeApp**
   5. Click **OK**.
   6. In the **SharePoint Customization Wizard**, name the App, **Chrome App**.
   7. Specify the site you will use for this lab.
   8. Select **SharePoint-hosted** as the hosting type.
   9. Click **Finish**.

## Step 2 – Code the App

In this Step, you code the app to use the Chrome control.

1. In the **Solution Explorer**, Right click on the **Scripts** node, select **Add Existing Item.**
   1. **Navigate** to C:\Program Files\Common Files\Microsoft Shared\web server extensions\15\TEMPLATE\LAYOUTS.
   2. In the file dialog, select the file **sp.ui.controls.js** and click **Add**.
2. In the Solution Explorer, expand the Scripts node
   1. Open **App.js** for editing.
   2. **Replace** all of the code in the file with the following:

var hostweburl;

// Load the SharePoint resources.

$(document).ready(function () {

// Load the js file and continue to the

// success handler.

$.getScript("../scripts/sp.ui.controls.js", renderChrome)

});

function renderChrome() {

// The Help, Account, and Contact pages receive the

// same query string parameters as the main page.

var options = {

"appIconUrl": "../Images/AppIcon.png",

"appTitle": "Chrome control app",

"appHelpPageUrl": "Help.html?"

+ document.URL.split("?")[1],

"settingsLinks": [

{

"linkUrl": "Account.html?"

+ document.URL.split("?")[1],

"displayName": "Account settings"

},

{

"linkUrl": "Contact.html?"

+ document.URL.split("?")[1],

"displayName": "Contact us"

}

]

};

var nav = new SP.UI.Controls.Navigation(

"chrome\_ctrl\_container",

options

);

nav.setVisible(true);

}

1. In the Solution Explorer, expand the **Pages** node.
   1. Delete **Default.aspx**.
   2. Right click the **Pages** node and select **Add⮚New Item** from the context menu.
   3. In the **Add New Item** dialog, select **Web⮚HTML Page**.
   4. Name the new page **Home.html** and click **Add**.
   5. Repeat these steps to add pages **Account.html**, **Contact.html**, and **Help.html**.
2. Replace all of the code in **Home.html** with the following.

<!DOCTYPE html>

<html>

<head>

<title>Chrome controls declarative sample</title>

<script src="../Scripts/jquery-1.7.1.js" type="text/javascript"></script>

<script src="../Scripts/app.js" type="text/javascript"></script>

<link href="../Content/App.css" rel="stylesheet" />

</head>

<body>

<!-- Chrome control placeholder -->

<div id="chrome\_ctrl\_container"></div>

<!-- The chrome control also makes the SharePoint

Website stylesheet available to your page -->

<h1 class="ms-accentText">Main content</h1>

<h2 class="ms-accentText">The chrome control</h2>

<div id="MainContent" >

This is the page's main content.

You can use the links in the header to go to the help,

account or contact pages.

</div>

</body>

</html>

1. Replace all of the code in **Account.html** with the following.

<!DOCTYPE html>

<html>

<head>

<title>Account Settings</title>

<script src="../Scripts/sp.ui.controls.js" type="text/javascript">

</script>

</head>

<body>

<!-- Note the use of siteUrl to reference the site for the theming -->

<div id="chrome\_ctrl\_container" data-ms-control="SP.UI.Controls.Navigation" data-ms-options='{

"siteUrl" : "http://intranet.contoso.com/sites/SPHostedApps",

"appHelpPageUrl" : "../Pages/Help.html",

"appIconUrl" : "../Images/appIcon.png",

"appTitle" : "Chrome Control",

"settingsLinks" : [

{

"linkUrl" : "../Pages/Account.html",

"displayName" : "Account settings"

},

{

"linkUrl" : "../Pages/Contact.html",

"displayName" : "Contact us"

}

]

}'>

</div>

<div style="margin: 50px">

<h3>Account Settings Page</h3>

<table cellspacing="3" cellpadding="3" border="0">

<tr>

<td>Store Personal Infomartion</td>

<td>

<input type="checkbox" /></td>

</tr>

<tr>

<td>E-Mail Address</td>

<td>

<input type="text" value="administrator@contoso.com" /></td>

</tr>

</table>

</div>

</body>

</html>

1. Replace all of the code in **Contact.html** with the following.

<!DOCTYPE html>

<html>

<head>

<title>Contact Us</title>

<script src="../Scripts/sp.ui.controls.js" type="text/javascript">

</script>

</head>

<body>

<!-- Note the use of siteUrl to reference the site for the theming -->

<div id="chrome\_ctrl\_container" data-ms-control="SP.UI.Controls.Navigation" data-ms-options='{

"siteUrl" : "http://intranet.contoso.com/sites/SPHostedApps",

"appHelpPageUrl" : "../Pages/Help.html",

"appIconUrl" : "../Images/appIcon.png",

"appTitle" : "Chrome Control",

"settingsLinks" : [

{

"linkUrl" : "../Pages/Account.html",

"displayName" : "Account settings"

},

{

"linkUrl" : "../Pages/Contact.html",

"displayName" : "Contact us"

}

]

}'>

</div>

<div style="margin: 50px">

<h3>Contact Page</h3>

<p><a href="mailto:administrator@contoso.com">Web Master</a></p>

</div>

</body>

</html>

1. Replace all of the code in **Help.html** with the following.

<!DOCTYPE html>

<html>

<head>

<title>Help Me</title>

<script src="../Scripts/sp.ui.controls.js" type="text/javascript">

</script>

</head>

<body>

<!-- Note the use of siteUrl to reference the site for the theming -->

<div id="chrome\_ctrl\_container" data-ms-control="SP.UI.Controls.Navigation" data-ms-options='{

"siteUrl" : "http://intranet.contoso.com/sites/SPHostedApps",

"appHelpPageUrl" : "../Pages/Help.html",

"appIconUrl" : "../Images/appIcon.png",

"appTitle" : "Chrome Control",

"settingsLinks" : [

{

"linkUrl" : "../Pages/Account.html",

"displayName" : "Account settings"

},

{

"linkUrl" : "../Pages/Contact.html",

"displayName" : "Contact us"

}

]

}'>

</div>

<div style="margin: 50px">

<h3>Help Page</h3>

<p>There's really no helping you.</p>

</div>

</body>

</html>

1. Open the **AppManifest.xml** designer.
   1. Set the **StartPage** element to be **Home.html**

## Step 3 – Deploy and Run the App

In this Step, you will deploy the App and run it.

1. In the **Solution Explorer**, Right click on the **ChromeApp** project, select **Add Deploy.**
2. **Navigate** to the Site hosting the App.
   1. Click **Site Contents**.
   2. Click **Chrome App**.

# Demo 5: Cross Domain Library

In this Demo, you will show how to use the cross-domain library to query data from a SharePoint-hosted app from a remote web.

## Step 1 – Show the CrossDomainApp project

1. Open the **CrossDomainLibrary** solution using Microsoft Visual Studio 2012.
2. In the **CrossDomainApp** project, open the Announcements list. This list will be deployed to the app web. Open the elements.xml file and show that it contains data.
3. Open the **AppManifest.xml** file in the XML editor (click F7 to view XML). Show the AppPrincipal has an **AllowedRemoteHostUrl** attribute that points to the remote web. This grants permission for the remote web to use the cross-domain capability.
4. In the **AppManifest.xml** file, show the start page is not the appweb, but a remote URL <http://localhost:40725/Pages/CrossDomainCallREST.html>. This tells SharePoint to redirect to the remote web, which will in turn make queries into SharePoint using the cross-domain library.
5. In the **AppManifest.xml** file, show the **AppPermissionRequest** to read from the web. Typically apps have full control to the app web, but the code that will read from the app web is remote and requires read permission to the app web itself.

## Step 2 – Show the CrossDomainWeb Project

1. In the **CrossDomainWeb** project, show that it runs in localhost by showing the project’s properties.
2. Show the script references:
   1. **MicrosoftAjax.js** - provides the decodeUriComponent implementation
   2. **SP.RequestExecutor.js** – copied from the LAYOUTS directory from a SharePoint 2013 server. This provides the cross-domain library implementation.
   3. **SP.js** - copied from the LAYOUTS directory from a SharePoint 2013 server. Used for CSOM calls.
   4. **SP.Runtime.js** - copied from the LAYOUTS directory from a SharePoint 2013 server. Used for CSOM calls.
3. Show the CrossDomainCallREST.html page
   1. We obtain the SPAppWebUrl from the querystring to know which web to make RESTful calls to.
   2. The **SP.RequestExecutor** is used to issue RESTful calls to the app web.
   3. The data is displayed upon success, otherwise an error is written to the page.
4. Show the CrossDomainCallCSOM.html page
   1. We obtain the SPAppWebUrl from the querystring to know which web to make CSOM calls to.
   2. The **SP.ProxyWebRequestExecutorFactory** is used to issue CSOM calls to the app web using the cross-domain library.
   3. The data is displayed upon success, otherwise an error is written to the page.