## Programming with JavaScript and the jQuery Library

**Lab Time**: 45 minutes

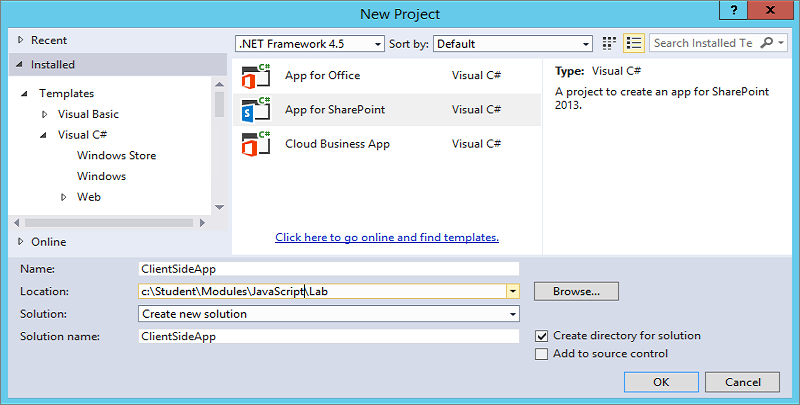
**Lab Folder**: C:\Student\Modules\JavaScript\Lab

**Lab Overview**: In this lab you will get hands-on experience writing JavaScript code with the jQuery library. Along the way you will work with event handlers, dynamically created HTML elements and AJAX calls across the network. In this lab you will also create a JavaScript Module which will teach you the best practice for creating reusable and maintainable JavaScript libraries.

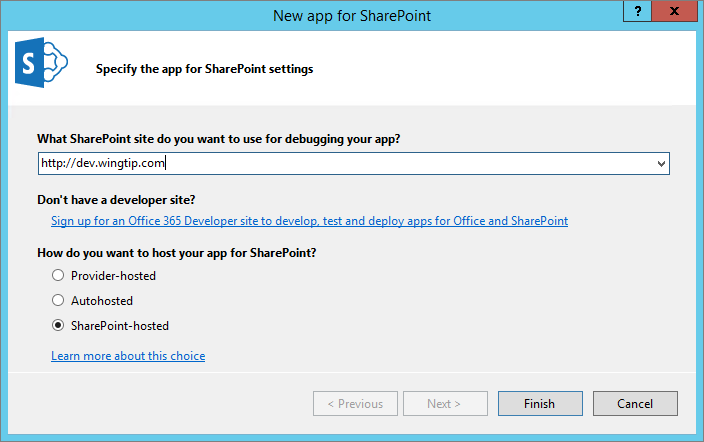
### Exercise 1: Adding client-side behavior to pages using JavaScript and jQuery

In this exercise you will create a new SharePoint-Hosted App in which you will be writing client-side code with JavaScript and jQuery.

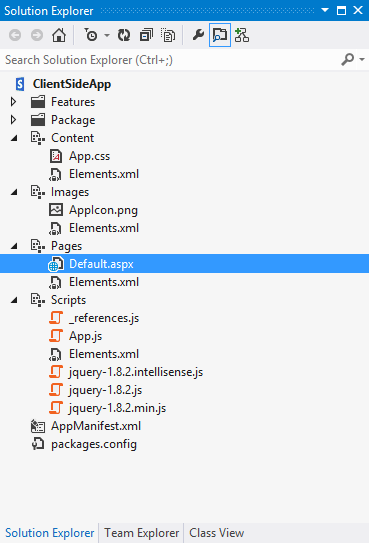
1. Create a new project in Visual Studio 2013:
   1. Launch **Visual Studio 2013** as administrator:
      1. Windows Keyboard Key 🡪 Right click on the Visual Studio 2013 tile and select Run as administrator.
   2. In Visual Studio select **File 🡪 New 🡪 Project**.
   3. In the **New Project** dialog:
      1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
      2. **Name**: ClientSideApp
      3. **Location:** C:\Student\Modules\JavaScript\Lab.



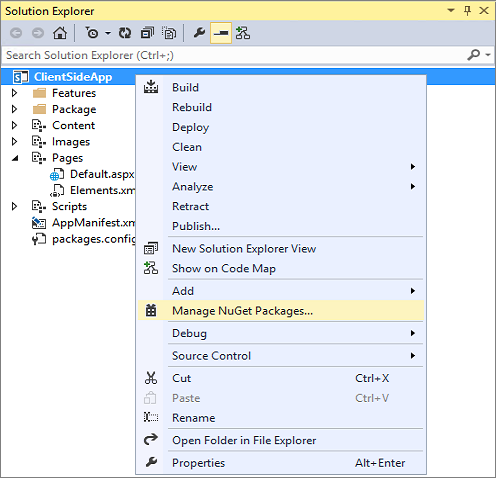
* + 1. Click **OK.**
  1. Next, the **New app for SharePoint** dialog appears. Fill it out like the one shown in the following screenshot. You can use any local test site for testing including the one you created in the last lab at <http://dev.wingtip.com>. When you are done, click **Finish** to create the project.



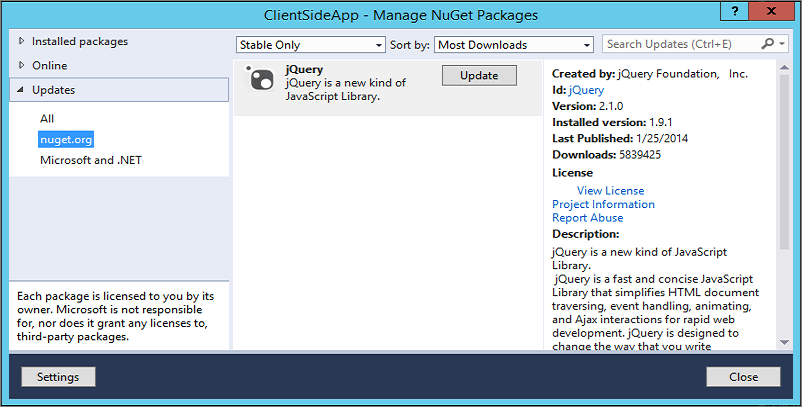
* 1. **Take a moment and inspect the new app project and all the files inside.**



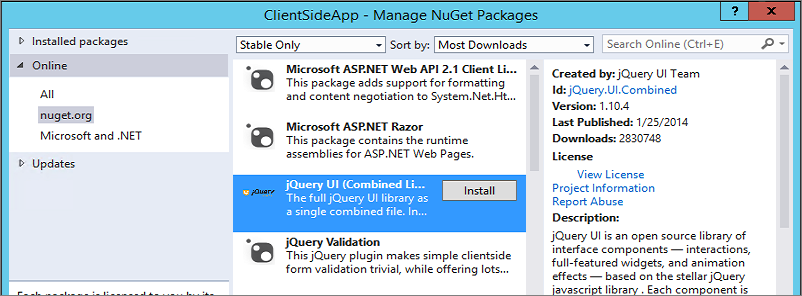
1. **Open the AppManifest.xml and set the Title: Client-Side App**
2. **Update to the latest version of the jQuery library and add the jQuery UI library.**
   1. **Right-click on the ClientSideApp project and click the menu command Manage NuGet Packages… to display the Manage NuGet Packages dialog.**



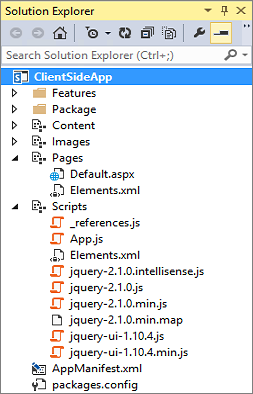
* 1. **Click the Updates tab on the left side of the Manage NuGet Packages dialog. Next, click the Update button for the jQuery package. This will update the jQuery sources files in your app project to the latest version.**



* 1. **Click the Online tab on the left side of the Manage NuGet Packages dialog. Look through the available NuGet packages and locate jQuery UI (Combined Library) and click the Install button. This will install the package into you project.**

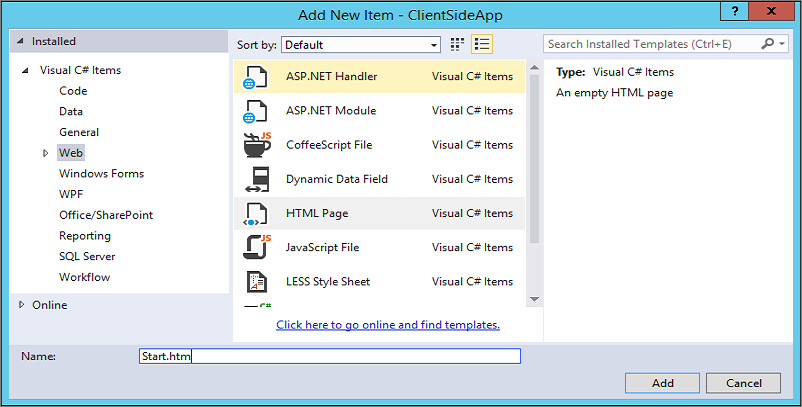


* 1. **Click the Close button at the bottom right to close the Manage NuGet Packages dialog.**
  2. **The Scripts folder in your project should now contain the updated files.**

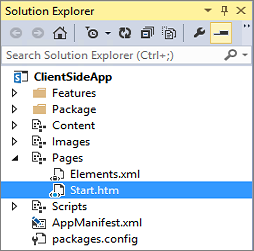


**Note the version numbers of the library files in your project may not match the version numbers shown in the screenshot. The version numbers might be more recent because these JavaScript library packages are updated on a regular basis.**

1. Create the start page for your app with a new HTML file named **start.htm**.
   1. Locate the **Default.aspx** file inside the Pages folder and delete it from the project.
   2. Right-click the **Pages** folder and select **Add New Item**.
   3. In the **Add New Item** dialog, select **Visual C# Items >> Web** on the left-hand side as shown in the following screenshot. Select **HTML Page** and give it a name of **start.htm.**



* 1. Verify that the **start.htm** file was added into the **Pages** folder.



* 1. You should see Visual Studio has added some initial HTML content to **start.htm**. Modify the content of **start.htm** to look like the following code listing.

<!DOCTYPE html>

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

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=10"/>

<title>My Client-side App</title>

</head>

<body>

<h2>My Client-side App</h2>

<p>Hello World</p>

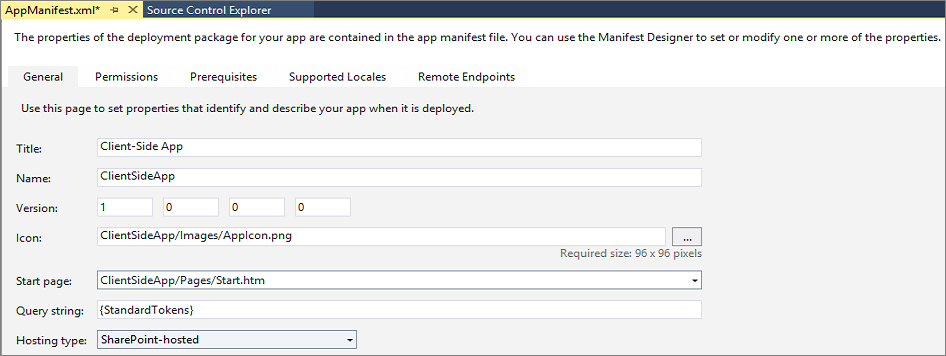
</body>

</html>

* 1. **Save Changes** and Close **start.htm**.

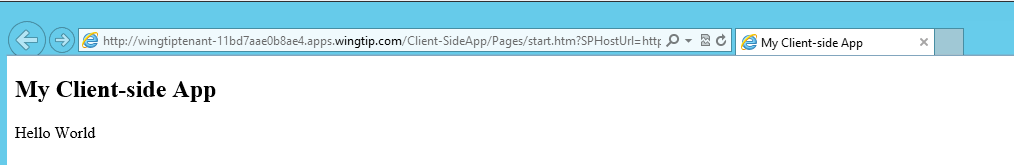
Now that you have created a start page, you must update the app manifest to use it.

1. Update the **AppManifest.xml** file to point to start.htm.
   1. In the Solution Explorer, locate and double-click on the **AppManifest.xml** file to open the app manifest in the Visual Studio designer.
   2. Update the app’s **Start page** property to point to **start.htm**.



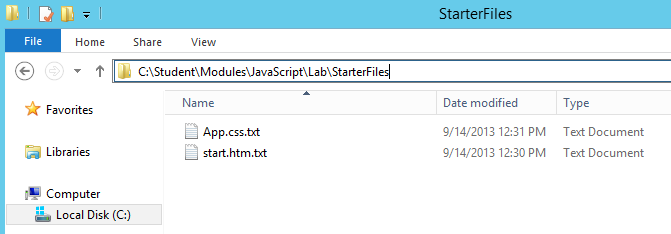
* 1. Close **AppManifest.xml**.

1. Test the **ClientSideApp** project
   1. In Visual Studio, launch a debugging session for the SharePoint app project by pressing the **{F5}** key. While there will not be too much excitement at this point, the app should at least install without any errors and display **start.htm** in the browser.



* 1. Stop the debugging session by closing the browser window.

1. Create the user interface for the app using HTML and CSS.
   1. Inside the Windows Explorer, look inside the folder at **C:\Student\Modules\JavaScript\Lab\StarterFiles** and locate the two files named **start.htm.txt** and **App.css.txt**.



* 1. Open **start.htm.txt** and copy its contents into the Windows clipboard.
  2. Return to Visual Studio. Open **start.htm** in Code View and replace the existing html code by selecting all and pasting the contents from the clipboard. The **start.htm** file should now contain the following contents.

<!DOCTYPE html>

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

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=10"/>

<title>My Client-side App</title>

<!-- links for CSS and JavaScript files go here -->

</head>

<body>

<div id="page\_width">

<div id="nav\_bar">

&nbsp;

</div>

<div id="top\_banner">

<div id="site\_title">Client-side App</div>

</div>

<div class="toolbar">

<input id="cmdHelloDOM" type="button" value="Hello DOM" />

<input id="cmdHellojQuery" type="button" value="Hello jQuery" />

<input id="cmdCreateList" type="button" value="Create List" />

<input id="cmdLinkHostWeb" type="button" value="Link to Host Web" />

<input id="cmdAjax" type="button" value="AJAX" />

<input id="cmdPromises" type="button" value="Promises" />

</div>

<div id="content\_box">

<div id="results"></div>

</div>

</div>

</body>

</html>

* 1. Save your changes to **start.htm.** However, you should leave this file open in the Visual Studio HTML editor as you will continue to work on it.
  2. Back in Windows Explorer, open **app.css.txt** and copy its contents into the Windows clipboard.
  3. Back in Visual Studio, open **App.css** in the **Content** folder and paste the contents from the clipboard. The **App.css** file should now contain the following contents.

body {

background-color: #CC6633;

margin: 0px;

padding: 0px;

}

#page\_width {

width: 960px;

margin-left: auto;

margin-right: auto;

border: 1px #000 solid;

background-color: white;

}

#nav\_bar {

background-color: black;

height: 20px;

}

#top\_banner {

background-color: #FFF;

padding: 10px;

}

#site\_title {

font-size: 36px;

color: #990033;

}

#content\_box {

min-height: 520px;

padding: 0px;

}

.toolbar {

background-color: #CCC;

padding: 8px;

}

.toolbar input {

font-size: 12px;

padding: 4px;

width: 140px;

}

#results {

padding: 8px;

}

#results li {

font-size: 18px;

margin-top: 8px;

margin-bottom: 2px;

color: #333;

}

* 1. Save and close **app.css**.
  2. Return to the **start.htm** file in the Visual Studio HTML editor and add a link to the CSS file **app.css**. (Note that you can accomplish this by dragging-and-dropping, or copying and pasting the CSS file from the Solution Explorer into the head section of **start.htm** in the Visual Studio HTML editor.)

<head>

<meta charset="utf-8" />

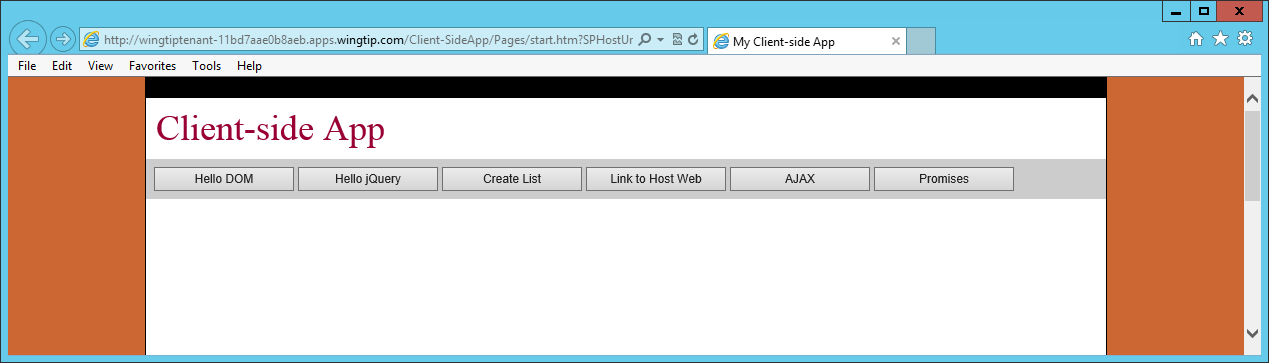
<meta http-equiv="X-UA-Compatible" content="IE=10"/>

<title>My Client-side App</title>

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

</head>

* 1. Now test your work by pressing the **{F5}** key to begin a debugging session. The app should appear with a better UI.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

1. Add JavaScript code to handle the click events of the six command buttons on the toolbar.
   1. Inside the Solution Explorer, open the **App.js** file which is located inside the **Scripts** folder.
   2. Delete the existing code inside **App.js** and replace it with the following JavaScript code which contains the document ready event handler which will invoke a dialog box with the message ‘Hello’ when the start page is loaded. Don’t forget to add the **use strict** statement at the top of **App.js**.

'use strict';

$(function () {

alert("Hello");

});

* 1. Save your changes to **App.js**.
  2. Open **start.htm** in code view and add links to the following source file.
     1. The jQuery source JavaScript file from the **Scripts** folder.
     2. The jQuery UI source JavaScript file from the **Scripts** folder.
     3. The jQuery UI source CSS file from the **Content/themes/base** folder.
     4. The **App.js** file from the **Scripts** folder.

Remember that the easiest way to add JavaScript file links to an HTML page is to drag-and-drop the source file from the **Scripts** folder in the Solution Explorer into the head section of the target HTML page.

* 1. When you are done making changes to the **head** section of the **start.htm**, it should appear as follows.

head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=10" />

<title>My Client-side App</title>

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

<script src="../Scripts/jquery-2.1.0.js"></script>

<script src="../Scripts/jquery-ui-1.10.4.js"></script>

<link href="../Content/themes/base/all.css" rel="stylesheet" />

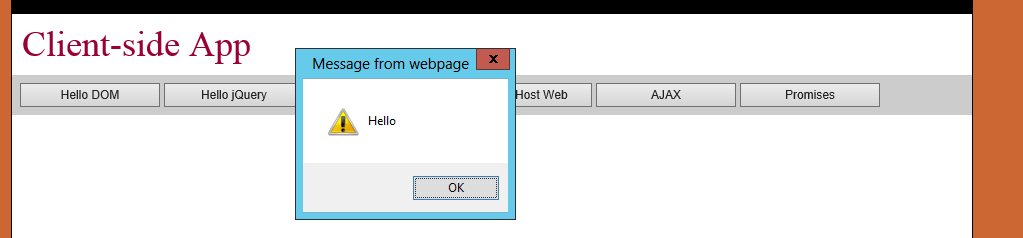
<script src="../Scripts/App.js"></script>

</head>

Note that the main CSS used by jQuery-UI used to be named **jquery-ui.css** but was renamed to **all.css** in a recent package update.

* 1. Save your changes to **start.htm**.

1. Now test your work by pressing the **{F5}** key. When the start page loads, you see a dialog greeting you with a friendly message.



1. Close the browser window to stop the debugging session and return to Visual Studio.
2. Add a jQuery event handler for each of the command buttons on the start page.
   1. Open **App.js** and modify it to look like the following code listing.

'use strict';

$(function () {

$("#cmdHelloDOM").click(onHelloDOM);

$("#cmdHellojQuery").click(onHellojQuery);

$("#cmdCreateList").click(onCreateList);

$("#cmdLinkHostWeb").click(onLinkHostWeb);

$("#cmdAjax").click(onAjax);

$("#cmdPromises").click(onPromises);

});

function onHelloDOM(){

alert("This function needs to be implemented");

}

function onHellojQuery(){

alert("This function needs to be implemented");

}

function onCreateList(){

alert("This function needs to be implemented");

}

function onLinkHostWeb(){

alert("This function needs to be implemented");

}

function onAjax(){

alert("This function needs to be implemented");

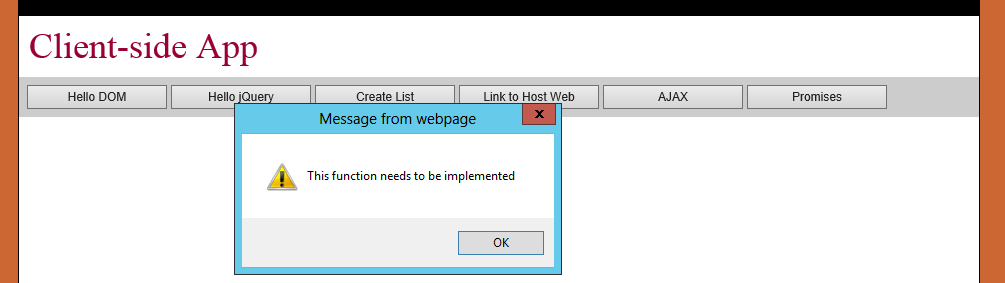
}

function onPromises() {

alert("This function needs to be implemented");

}

* 1. Now test your work by pressing the **{F5}** key. When the start page loads, try and click on each of the command buttons on the page. Each button should have a click event handler that brings up a dialog like the one shown in the following screenshot.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

1. Write *pure JavaScript* code in the click event handler for the **cmdHelloDOM** button to add a simple message to the page.
   1. Locate the **onHelloDOM** function inside **App.js** and remove the line of code that is there. Replace it with the following JavaScript code which will dynamically add content into the div element with the **id** of **results**.

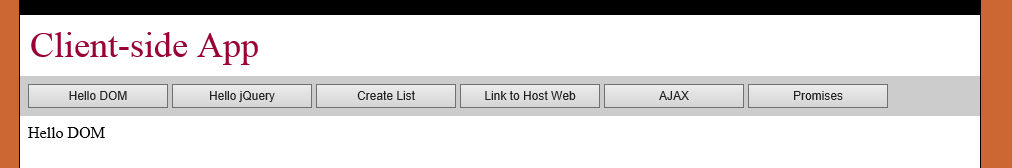
function onHelloDOM() {

var resultsDiv = document.getElementById("results");

resultsDiv.innerHTML = "Hello DOM";

}

* 1. Test your work by pressing **{F5}** to start the app and click the **Hello DOM** button. When you click the **Hello DOM** button, you should see the "Hello DOM" message on the page.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

1. Write *jQuery* code in the click event handler for the **cmdHellojQuery** button to add a simple message to the page.
   1. Locate the **onHellojQuery** function inside **App.js** and remove the line of code that is there. Replace it with the following JavaScript code which use the jQuery library to add content into the div element with the **id** of **results**.

function onHellojQuery(){

$("#results").text("Hello jQuery");

}

* 1. Test your work by pressing **{F5}** to start the app and click the **Hello jQuery** button. When you click the **Hello jQuery** button, you should see the "Hello jQuery" message on the page.
  2. Close the browser window to stop the debugging session and return to Visual Studio.
  3. Now, take it a bit further by adding some CSS styles to your message. Update the **onHellojQuery** function as follows to make the message display in blue and in a larger font.

function onHellojQuery(){

$("#results")

.text("Hello jQuery")

.css({"color":"blue","font-size": "48px"});

}

* 1. Test your work by pressing **{F5}** to start the app and click the **Hello jQuery** button. When you click the **Hello jQuery** button this time, you should see the "Hello jQuery" message on the page in a larger blue font.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

You have now completed the initial steps required to develop a SharePoint hosted app by adding HTML, CSS and JavaScript. Now you will move ahead and write some more JavaScript code which is a bit more involved.

### Exercise 2: Using jQuery to Dynamically Create HTML Elements

In this exercise you will learn how to use jQuery syntax to dynamically create HTML elements and add them to the DOM of the current page. Along the way you will also create a reusable JavaScript library file using the Module pattern.

1. Write code in the click event handler for the **cmdCreateList** button to dynamically create an HTML **<ol>** element.
   1. Locate the **onCreateList** function inside **App.js** and remove the line of code that is there. Replace it with the following code which declares a JavaScript array and uses its contents to add a dynamically created **<ol>** element to the page.

function onCreateList() {

// declare JavaScript array of strings

var people = ["Betty", "Frank", "Sandra", "Ricco"];

// create <ol> element

var list = $("<ol>");

// add li child elements

for (var index = 0; (index < people.length) ; index++) {

list.append($("<li>").text(people[index]));

}

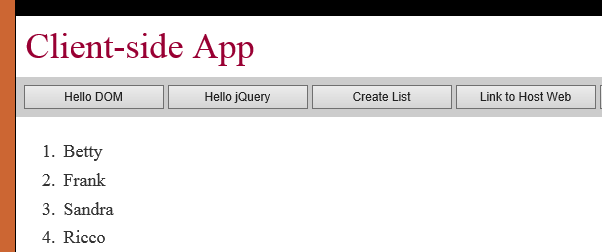
// add list element to page

$("#results").empty();

$("#results").append(list);

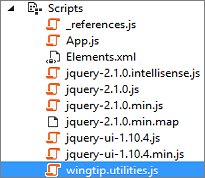
}

* 1. Test your work by pressing **{F5}** to start the app and click the **Create List** button. When you click the **Create List** button, you should see output on the page that looks like the following screenshot.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

1. Create a reusable JavaScript library file using the Module pattern.
   1. Add a new JavaScript file to the **Scripts** folder and give it a name of **wingtip.utilities.js**.
      1. Right Click on **Scripts** folder and select **Add 🡪 New Item**
      2. Select **JavaScript File** under the **Visual C# 🡪 Web** category
      3. Name the file **wingtip.utilities.js**



* 1. Add a use strict statement and define a new JavaScript namespace named **Wingtip**.

'use strict';

var Wingtip = window.Wingtip || {};

* 1. Inside the **wingtip.utiltiies.js** file, create a JavaScript Module named **Wingtip.Utilities** with two functions named **getQueryStringParameter** and **executeGetRequest**. Accomplish this by creating a self-executing function with the following starting point.

'use strict';

var Wingtip = window.Wingtip || {};

Wingtip.Utilities = function () {

var getQueryStringParameter = function (param) {

// TODO

}

var executeGetRequest = function (urlContent) {

// TODO

}

return {

getQueryStringParameter: getQueryStringParameter,

executeGetRequest: executeGetRequest

};

}();

* 1. Now update the implementation of **getQueryStringParameter** with the following code which accepts the name of a parameter of interest and the query string parameters then searches the query string parameters for a match, which, if found, then returns the matching query string parameter’s value to the caller.

var getQueryStringParameter = function (param) {

var querystring = document.URL.split("?")[1];

if (querystring) {

var params = querystring.split("&");

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

var current = params[index].split("=");

if (param.toUpperCase() === current[0].toUpperCase()) {

return decodeURIComponent(current[1]);

}

}

}

}

* 1. Save your changes to **wingtip.utilities.js**.

1. Open **start.htm** and add a link to **wingtip.utilities.js**.
   1. Add the script link to **wingtip.utilities.js** so that it appears after the links to the jQuery library and jQuery UI library but before the link to **App.js**.

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=10" />

<title>My Client-side App</title>

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

<script src="../Scripts/jquery-2.1.0.js"></script>

<script src="../Scripts/jquery-ui-1.10.4.js"></script>

<link href="../Content/themes/base/jquery-ui.css" rel="stylesheet" />

<script src="../Scripts/wingtip.utilities.js"></script>

<script src="../Scripts/App.js"></script>

</head>

1. Now return back to the **App.js** file. Write code in the click event handler for the **cmdLinkHostWeb** command button to dynamically add a hyperlink to the page which links back to the host web.
   1. Locate the **onLinkHostWeb** function in **App.js** and remove the line of code that is there. Replace it with the following code which calls the **getQueryStringParameter** from the **Wingtip.Utilities** and uses the returned value to add a **<a>** element to the page.

function onLinkHostWeb() {

// get URL back to host web

var urlHostWeb = Wingtip.Utilities.getQueryStringParameter("SPHostUrl");

// create <a> elements and link it to host web

var linkHostWeb =

$("<a>")

.attr("href", urlHostWeb)

.css({ "color": "white", "text-decoration": "none" })

.text("Back to Host Web");

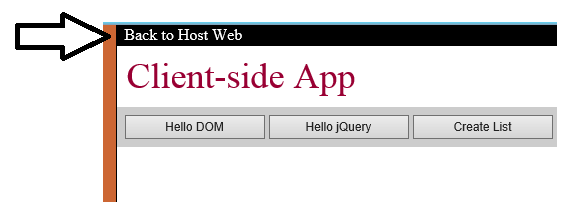
// append link into nav\_bar div

$("#nav\_bar").append(linkHostWeb);

}

Note that you should be dynamically adding the **<a>** element into the div with the **id** of **nav\_bar** as opposed to the div with the **id** of **results** as you have been doing in other steps in this lab.

* 1. Test your work by pressing **{F5}** to start the app and click the **Link to Host Web** button. You should see a new hyperlink appear in the top left portion of the page.



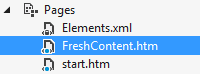
* 1. Make sure the **Back to Host Web** link works as it should by clicking on it. Clicking this hyperlink should take you back to the host web that you are using for your testing.
  2. Close the browser window to stop the debugging session and return to Visual Studio.

Now that you have learned to generate HTML elements with jQuery and how to create a JavaScript Module, it's time to move on to the next exercise in which you will use client-side JavaScript and the jQuery library to issue AJAX requests to retrieve HTML content from across the network.

### Exercise 3: Making AJAX Calls using the jQuery Library

In this exercise you will write jQuery code to execute HTTP GET requests to retrieve content from across the network. You will also learn how to use the JavaScript Promises pattern to separate the JavaScript code that executes calls across the network from the code that updates the user interface of a page.

1. Add a new HTML page named **FreshContent.htm** to the **Pages** folder so you have some sample content to retrieve.
   1. Add a new HTML page to the **Pages** folder and name it **FreshContent.htm**: Right Click on **Pages** in the Solutions Explorer **🡪 Visual C# Items 🡪 Web** category **🡪 HTML Page**



* 1. Delete all the content inside **FreshContent.htm** and replace it with the following content.

<div>

<h2>Fresh content from across the network</h2>

<div>Hello AJAX</div>

</div>

* 1. Save and close **FreshContent.htm**.

1. Write code in the click event handler for the **cmdAJAX** button to retrieve **FreshContent.htm** using the jQuery **$.ajax** function.
   1. Locate the **onAjax** function inside **App.js** and remove the line of code that is there. Replace it with the following code.

function onAjax() {

// use ajax() function to execute an HTTP request

$.ajax({

url: "FreshContent.htm",

type: "GET",

dataType: "html",

success: onAjaxComplete,

error: onAjaxError

});

}

* 1. In **App.js** add two functions named **onAjaxComplete** and **onAjaxError** directly after the **onAjax** function with the code shown in the following listing.

function onAjaxComplete(responseData) {

$("#results").html(responseData);

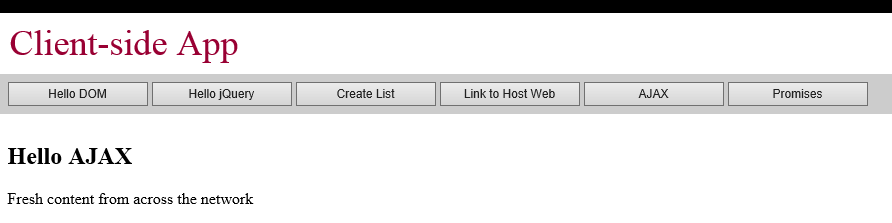
}

function onAjaxError(error) {

$("#results").text("ERROR: " + error);

}

* 1. Test your work by pressing **{F5}** to start the app and click the **AJAX** command button. When you click the **AJAX** command button, you should see the HTML contents from **FreshContent.htm** displayed on the start page.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

In the final steps you will refactor the code that calls across the network out into the **Wingtip.Utilities** Module. This will teach you how to use the *JavaScript Promises pattern* so you can separate the code that accesses data from across the network from the code that updates the user interface.

1. Implement the **executeGetRequest** function to make an AJAX call and return a promise.
   1. In **wingtip.utilities.js**, modify the implementation of the **executeGetRequest** function to execute an AJAX request using the jQuery **$.ajax** function. Capture the **XmlHttpRequest** object returned from the **$.ajax** function and use that object to return the promise to the caller using the code shown in the following listing.

var executeGetRequest = function (urlContent) {

var xhr = $.ajax({

url: urlContent,

type: "GET",

dataType: "html"

});

return xhr.promise();

}

return {

getQueryStringParameter: getQueryStringParameter,

executeGetRequest: executeGetRequest

};

}();

* 1. Save your changes to **wingtip.utilities.js**.

1. Write code in the click event handler for the **cmdPromises** button to call **executeGetRequest** and to properly deal with the promise that is returned.
   1. Locate the **onPromises** function inside **App.js** and remove the line of code that is there. Replace it with the following code to call the **executeGetRequest** function from **Wingtip.Utilities** Module and to capture the return value with the promise object.

function onPromises() {

var promise = Wingtip.Utilities.executeGetRequest("FreshContent.htm");

}

* 1. Call the **then** function of the promise object and provide two anonymous function for success and for failure as shown in the following code listing.

function onPromises() {

var promise = Wingtip.Utilities.executeGetRequest("FreshContent.htm");

promise.then(

// implement success function

function (responseData) {

$("#results").html(responseData);

},

// implement error function

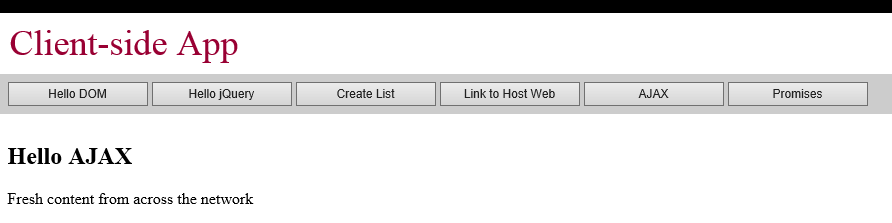
function (error) {

$("#results").text("ERROR: " + error);

});

}

* 1. Test your work by pressing **{F5}** to start the app and click the **Promises** commandbutton. When you click the **Promises** commandbutton, you should see the HTML contents from **FreshContent.htm** displayed on the start page.



* 1. Close the browser window to stop the debugging session and return to Visual Studio.

**You are now done with this lab and should feel more comfortable with programming JavaScript and jQuery.**