|  |
| --- |
| Microsoft SharePoint 2013 - Hands-on Lab |
| Working with the CSOM and REST APIs |
| Verified Against Build 15.0.4420.1017 |

|  |
| --- |
| Microsoft  Version 1.0  August 14, 2012 |

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright © 2012 Microsoft Corporation. All rights reserved.

Microsoft and SharePoint are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Technical Background

This lab will walk you through creating SharePoint apps where the app is primarily hosted outside of SharePoint. Hosting options include, but are not limited to, an ASP.NET Web site hosted in IIS, Windows Azure or any other Web accessible infrastructure of choice.

## Create Lab SharePoint Site Collection

In the files provided with the hands on lab, run the batch file called **SetupModule.bat** by double clicking it. This file will execute a PowerShell script that will create a new site collection at [http://intranet.contoso.com/sites/](http://wave15-sp/sites/IntroSpApps)RESTLab.

|  |
| --- |
| Description: C:\Users\vesaj\Pictures\DVD_ART36\Artwork_Imagery\Icons - Illustrations\_ SUPER VISTA STYLE\yield sign red white exclamation point.png **Important** |
| *It is important you run this batch file to create the site collection before working through any of the exercises as the exercises contain instructions for working with this specific site collection at the specific URL created by the script.* |

The script will first check to see if there is already a site collection at the specified address. If there is it will delete the site collection before creating it. Therefore if you run into problems with the lab, feel free to rerun the batch file to reset the environment and restart the exercise.

# Introduction

## Estimated time to complete this lab

60 minutes

## Objectives

After completing this lab, you will be able to:

* Learn the fundamentals of new REST API for the CSOM
* Learn to call the REST API using the browser, JavaScript and jQuery
* Learn to present JSON resultsets using jQuery Templates
* Learn to use Visual Studio to write JavaScript code behind site pages

## Overview of Lab

In this lab you will learn how to access the new the SharePoint CSOM through the new REST API. You will write code behind site pages using JavaScript which leverages the AJAX capabilities of the jQuery library. You will also learn how to use the jsRender library and templates to format JSON results into HTML for display on a page.

## Virtual Machine Technology

The computers in this lab are virtual machines that are implemented using Microsoft Hyper-V. Before starting each virtual machine, ensure you apply the **Start-Lab** snapshot. When you have started a virtual machine, log on by pressing **CTRL+ALT+END** and supply the credentials listed in the lab instructions.

## Computers in this lab

This lab uses virtual machines as described in the following table. Before you begin the lab, you must start the virtual machines and then log on to the computers.

|  |  |
| --- | --- |
| **Virtual Machine** | **Role** |
| {Supplied by Instructor} | Domain Controller |
| {Supplied by Instructor} | Actual SharePoint environment with Office client and other required software. |

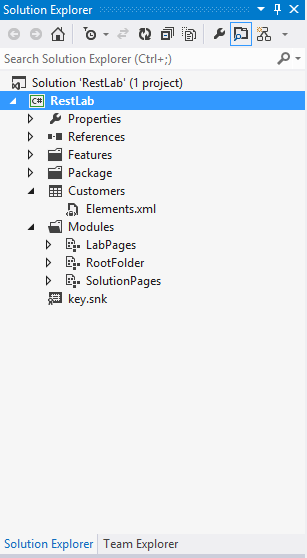
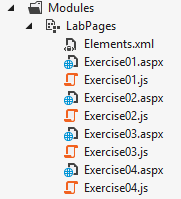
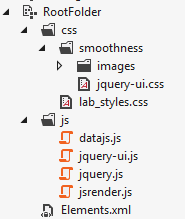
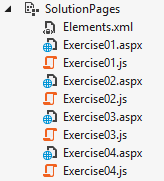
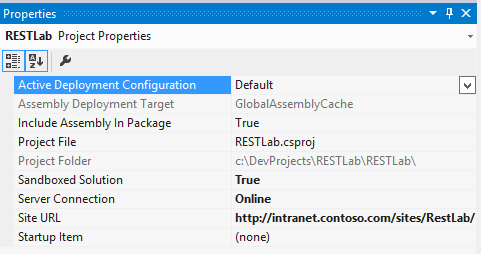
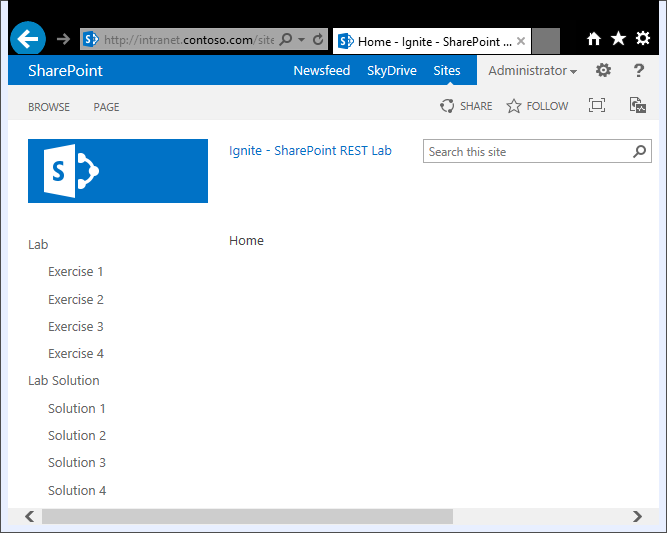
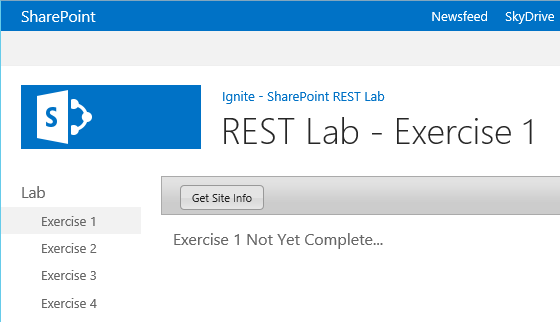
All user accounts in this lab use the password {Supplied by Instructor}.

## Lab Setup Script

In the files provided with the hands on lab, run the batch file called **SetupModule.bat** by double clicking it. This file will execute a PowerShell script that will create a new site collection at [http://intranet.contoso.com/sites/](http://wave15-sp/sites/IntroSpApps)RESTLab.

# Exercise 1: Calling the CSOM Rest API

* 1. In this exercise, you will have a walkthrough which shows how to form URLs to make REST API calls using the browser.

1. Launch Visual Studio 2012.
2. Look inside the folder for this lab and locate the folder named **RestLab**. You should be able to see that there is a Visual Studio solution inside this folder with a solution file named **RestLab.sln**. Use the **File > Open Project/Solution** menu command in Visual Studio to open the solution. You should be able to see that this solution contains a project named **RestLab**.
   * 1. 
3. Note there is a project item to create a list named **Customers**. When you deploy the project later in this exercise, it creates a Contacts list with the title of **Customers** and adds a few default items that will be used in testing your code.
4. Inside the **Modules** folder, there are three existing modules named **LabPages**, **RootFolder** and **SolutionPages**.
   * 1. Expand the Module named **LabPages** and see what files are inside. There are four site pages and a JavaScript file associated with each of the site pages. Most of the work you will do in this lab will involve writing JavaScript code inside these JavaScript source files to retrieve data using the CSOM REST API.
        + 1. 
     2. Expand the Module named **RootFolder** and see what files are inside. This modules contains JavaScript libraries for **jquery.js**, **jquery-ui.js**, **datajs.js** and **jsrender.js**. There is also a **css** folder with .css files and images which are part of the jQuery UI library. Note that the links to these library source files have already been added to the pages you will be working with. Therefore, the code you write in this lab can automatically use the jQuery and jQuery UI libraries without you having to worry about linking to their source files.
        + 1. 
     3. Expand the Module named **SolutionPages** and see what files are inside. There are four site pages and an associated JavaScript file for each site page. They four JavaScript files represent the solutions to the exercises for this lab.
        + 1. 
5. The **RestLab** project contains a features named **MainSite** which has an associated feature receiver. When this features activates, the code inside the feature receiver creates navigation links to the pages for the lab as well as the solution pages.
6. Within the Solution Explorer, select the top-level **RestLab** project node and inspect the property sheet and ensure the Site URL property for the project is configured correctly to use the test site for this lab at <http://intranet.contoso.com/sites/RestLab>.
   * + 1. 
7. Within the Solution Explorer, right-click on the top-level **RestLab** project node and click on the **Deploy** command to deploy this solution to the test site. Once the solution has been deployed and its feature has been activated, return to the test site in the Internet Explorer and refresh the home page. You should now see several links in the Quick Launch bar that make it possible to navigate to all the pages for each of the four exercises and for the solutions as well.
   * + 1. 
8. Click on the **Exercise 1** link in the Quick Launch bar to navigate to the page **Exercise1.aspx**. Examine what this page looks like in the Internet Explorer. Note there is a command button on the page. Click the command button and notice that the code currently behind this button writes some text to an element on the page.
   * + 1. 
9. Return to the **RestLab** project in Visual Studio and locate the site page named **Exercise01.aspx** inside the module named **LabPages**. Note that you will not be required to edit this file but you should open it in the code editor so you can see what is inside. You should notice that the placeholder named **PlaceHolderAdditionPageHead** already contains links to **jquery.js**, **jquery-ui.js** and the JavaScript file you will be editing in this exercise named **Exercise01.js**.

<SharePoint:ScriptLink Name="~sitecollection/js/jquery.js"   
 Defer="false" runat="server" />

<SharePoint:ScriptLink Name="~sitecollection/js/jquery-ui.js"   
 Defer="false" runat="server" />

<script src="Exercise01.js" type="text/javascript" ></script>

1. Also note that **Exercise01.aspx** contains a placeholder named **PlaceHolderMain** which contains an input control with an **id** of **cmdGetSiteInfo** which acts as a command button and a div with an **id** of **results**. When you write your code over the next few steps, you use this div with an **id** of **results** to display content on the page.

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

<div id="toolbar" class="ui-widget-header" >

<input id="cmdGetSiteInfo" type="button" value="Get Site Info" />

</div>

<div id="results" />

</asp:Content>

1. Open the JavaScript source file named **Exercise01.js** and inspect the code inside.

// register onPageLoad function with jquery Document Ready event handler

$(onPageLoad);

function onPageLoad() {

// use jquery-ui to format command button

$(":button", "#toolbar").button();

// register event handler for click event

$("#cmdGetSiteInfo").click(onGetSiteInfo);

}

function onGetSiteInfo() {

// TODO: write code for exercise 1

$("#results").html("Exercise 1 Not Yet Complete...");

}

1. You will complete this exercise by rewriting the implementation of the **onGetSiteInfo** function. Begin by removing the existing code inside the function and replacing it with jQuery code which clears the content of the **results** div element and adds an img element which displays the standard SharePoint image with the spinning gears.

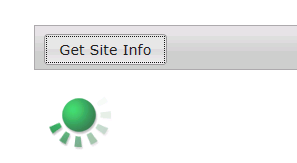
function onGetSiteInfo() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

}

1. Now it's time to test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 1** link to navigate to the page named **Exercise01.aspx**. Click the command button and ensure you can see the spinning gears.
   * 1. 
2. Place your cursor at the bottom of the **onGetSiteInfo** function so you can write more code. Create a variable named **requestUri** to track the site-relative URL to the REST-based API. Calculate the path for this Uri by appending **\_spPageContextInfo.webAbsoluteUrl** together with **\_/api/Web**.

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web";

1. Next, execute an AJAX request which returns a JSON result by calling the jQuery function named **ajax**. Make sure to setup a callback method for success named **onDataReturned** and a callback method for failure named **onError**. Note that the callback method for success should accept a single parameter which you should name **data**.

function onGetSiteInfo() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

// begin work to call across network

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web";

// execute AJAX request

$.ajax({

type: "GET",

url: requestUri,

contentType: "application/json",

headers: { Accept: "application/json;odata=verbose" },

success: OnDataReturned,

error: onError

});

}

function onDataReturned(data) {

}

function onError(err) {

}

1. Begin implementing the **onDataReturned** function by clearing all content from the **results** div. Next, create a variable named odataResults to track the JSON result set from the AJAX call and initialize it by assigning the **d** property of the **data** parameter.

function OnDataReturned(data) {

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

var odataResults = data.d

}

1. Next, write the jQuery code required to display site properties including **Id**, **Url**, **Title**, **MasterUrl** and **Language**. The following code shows one possible implementation.

function OnDataReturned(data) {

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

var odataResults = data.d

$("<h2>").html("Site Properties")

.appendTo("#results");

$("<p>").html("<b>Id</b>: " + odataResults.Id)

.appendTo("#results");

$("<p>").html("<b>Url</b>: " + odataResults.Url)

.appendTo("#results");

$("<p>").html("<b>Title</b>: " + odataResults.Title)

.appendTo("#results");

$("<p>").html("<b>MasterUrl</b>: " + odataResults.MasterUrl)

.appendTo("#results");

$("<p>").html("<b>Language</b>: " + odataResults.Language)

.appendTo("#results");

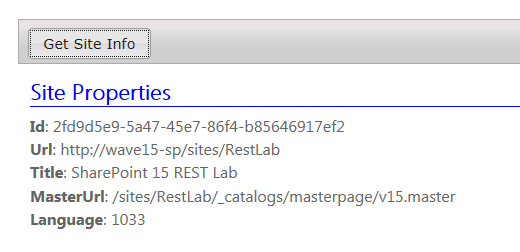
}

1. Implement the **onError** function as follows to display the error message to the page.

function onError(err) {

$("#results").text("ERROR: " + JSON.stringify(err));

}

1. Test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 1** link to navigate to the page named **Exercise01.aspx**. Click the command button and ensure you can see that your code has called into the CSOM REST API to determine site properties.
   * 1. 
2. At this point you have successfully made your first REST call into the SharePoint 2013 CSOM using JavaScript and jQuery.

# Exercise 2: Executing OData Queries to Discover the Lists

* 1. In this exercise, you will call into the REST APIs as you did in the previous exercise. However, now you will deal with a JSON result set that has mutliple items. In particular, you must write to code to enumerate through the existing set of lists in the current site.

1. Inside the **LabPages** module, locate and open the site page named **Exercise02.aspx**. Note that you will not be required to edit this file but you should open it in the code editor so you can see what is inside. You should see that this page has content that is almost identical to **Exercise1.aspx**.Itcontains a placeholder named **PlaceHolderMain** which contains a command button and a div with an **id** of **results**. As in the previous exercise, you will use the **results** div to display content on the page.
2. Open the JavaScript source file named **Exercise02.js** and inspect the code inside. Your work for this exercise will involve rewriting the function named **onGetLists**.

function onGetLists() {

// TODO: write code for exercise 2

$("#results").html("Exercise 2 Not Yet Complete...");

}

1. Begin by removing the existing code inside the function and replacing it with jQuery code to clear the content of the **results** div element and add an img element which displays the standard SharePoint image with the spinning gears.

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, write code to calculate the proper Uri to retrieve the lists from the current site.

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/Lists";

1. Next, execute an AJAX request which returns a JSON result by calling the jQuery function named **ajax**. Make sure to setup a callback method for success named **onDataReturned** and a callback method for failure named **onError**. Note that the callback method for success should accept a single parameter which you should name **data**.

function onGetLists() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

// begin work to call across network

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/Lists";

$.ajax({

type: "GET",

url: requestUri,

contentType: "application/json",

headers: { Accept: "application/json;odata=verbose" },

success: onDataReturned,

error: onError

});

}

1. Begin implementing the **onDataReturned** function by clearing all content from the **results** div. Next, create a variable named **odataResults** to track the JSON result set from the AJAX call and initialize it by assigning the JavaScript array in the **d.results** property of the **data** parameter. Note that this is different from the previous exercise where you used the **d** property not the **d.results** property.

function onDataReturned(data) {

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

var odataResults = data.d.results;

}

1. Complete the implementation of the **onDataReturned** function by enumerating through **odataResults** array to discover the **Title** property of each list and to display these list titles on the page.

function onDataReturned(data) {

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

var odataResults = data.d.results;

$("<h2>").html("Lists in this site").appendTo("#results");

var ul = $("<ul>");

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

$("<li>").html(odataResults[i].Title).appendTo(ul);

}

ul.appendTo("#results");

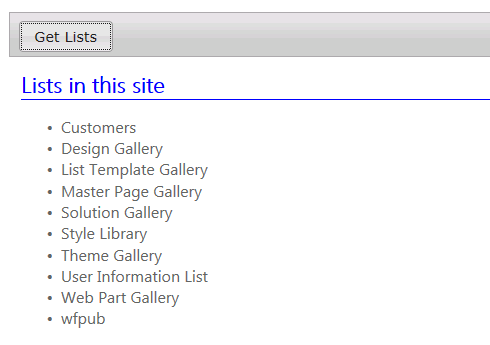
}

1. Implement the **onError** function as follows to display the error message to the page.

function onError(err) {

$("#results").text("ERROR: " + JSON.stringify(err));

}

1. Test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 2** link to navigate to the page named **Exercise02.aspx**. Click the command button and ensure you can see that your code has called into the CSOM REST API to enumerate through all the lists in the current site.
   * 1. 
2. Now you have learned how to enumerate through a JSON result set returned by the REST API.

# Exercise 3: Using OData Filters and Templates

* 1. In this exercise, you will make calls to the REST APIs using the jQuery **getJSON** function as you did in the previous exercise. However, when calling into the REST API to get the list of the site, you will add a filter expression to filter out hidden system lists such as the MasterPage Gallery and the List Template Gallery. You will also begin working with a JavaScript utility library named jsRender which makes it much easier to parse a JSON resultset into a complex HTML element for display.

1. Inside the **LabPages** module, locate and open the site page named **Exercise03.aspx**. Note that you will not be required to edit this file but you should open it in the code editor so you can see what is inside. You should see that it is almost identical to **Exercise01.aspx**.Itcontains a placeholder named **PlaceHolderMain** which contains a command button and a div with an **id** of **result**. As in the previous exercise, you will use the **result** div to display content on the page. One important difference is that the page contains an extra **ScriptLink** control which links to the JavaScript source file for the jsRender library.

<SharePoint:ScriptLink Name="~sitecollection/js/jsrender.js"

Defer="false" runat="server" />

1. Open the JavaScript source file named **Exercise03.js** and inspect the code inside. You work for this exercise will involve rewriting the function named **onGetUserLists**.

function onGetUserLists() {

// TODO: write code for exercise 3

$("#results").html("Exercise 3 Not Yet Complete...");

}

1. Begin by removing the existing code inside the function and replacing it with jQuery code which clears the content of the **results** div element and adds an img element which displays the standard SharePoint image with the spinning gears.

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, write code to calculate the proper Uri to retrieve the lists from the current site as you did in the previous exercise. However, add a query string to end of the Uri using the **$filter** parameter to filter out hidden system lists which have a **hidden** property equal to **true**.

var requestUri = \_spPageContextInfo.webAbsoluteUrl +

"/\_api/Web/Lists?$filter=Hidden eq false";

1. Now call the **ajax** function and setup up the callback methods as you did in the previous exercise.

function onGetUserLists() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

// begin work to call across network

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/Lists?$filter=Hidden eq false";

// execute AJAX request

$.ajax({

type: "GET",

url: requestUri,

contentType: "application/json",

headers: { Accept: "application/json;odata=verbose" },

success: onDataReturned,

error: onError

});

}

1. Begin implementing the **onDataReturned** function by clearing all content from the **results** div. Next, create a variable named **odataResults** to track the JSON result set from the AJAX call and initialize it by assigning the JavaScript array in the **d.results** property of the **data** parameter. Note that this is different from the previous exercise where you used the **d** property not the **d.results** property. This work is exactly what you did in exercise 2.

function onDataReturned(data) {

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

var odataResults = data.d.results;

}

1. Now, you are going to do things differently from exercise 2 because you are going to leverage the jsRender library to create a HTML ordered list from the JavaScript array held within a JSON result set. Complete the implementation of onDataReturned using the following code which creates a jsRender template using the **$.template** method and generates formatted output from the array and the template using the **$.render** method.

function onDataReturned(data) {

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

var odataResults = data.d.results;

$("<h2>").html("Lists in this site").appendTo("#results");

// create a template using jsRender library

renderingTemplate = "<li>{{=Title}}</li>";

$.template("tmplLists", renderingTemplate);

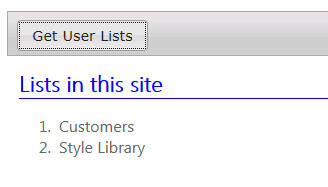
// generate output from array using template

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

ol.append($.render(odataResults, "tmplLists"));

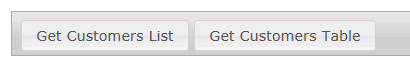
$("#results").append(ol);

}

1. Test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 3** link to navigate to the page named **Exercise03.aspx**. Click the command button and ensure you can see that your code displays all the lists in the site which are not hidden system lists.
   * 1. 
2. At this point, you have learned how to add filtering expression in the query string of the REST Uri as well as how to leverage the jsRender library to generate HTML output from an OData result set using templates.

# Exercise 4: Querying SharePoint List Items

* 1. In this exercise, you will use all the skills gained in the previous exercises to read and display the list items from a list. You will use the REST API to run a query against a specific list which returns a specific set of columns. You will use the jsRender library and a template to format the item content as a HTML list. Then you will get more involved by creating a second template that displays the same list content as an HTML table.

1. Inside the **LabPages** module, locate and open the site page named **Exercise04.aspx**. Note that you will not be required to edit this file but you should open it in the code editor so you can see what is inside. You should see that it is almost identical to **Exercise3.aspx** because it includes a div with an **id** of **results**. As in the previous exercise, you will use the **results** div to display content on the page. However, this page has two command buttons instead of one.
   * 1. 
2. Open the JavaScript source file named **Exercise04.js** and inspect the code inside. Your work for this exercise will involve rewriting the two function named **onGetCustomersLists** and **onGetCustomersTable**.

function onGetCustomersList() {

// TODO: write code for exercise 4

$("#results").html("Exercise 4 Part A Not Yet Complete...");

}

function onGetCustomersTable() {

// TODO: write code for exercise 4

$("#results").html("Exercise 4 Part B Not Yet Complete...");

}

1. Begin modifying the **onGetCustomersLists** function by removing the existing code inside the function and replacing it with jQuery code which clears the content of the **results** div element and adds an img element which displays the standard SharePoint image with the spinning gears.
2. Next, write code to calculate the URI to retrieve the items from the list in the current site which has a **Title** of **Customers**. Specify the columns to retrieve by adding a query string with the **$select** parameter which request three site columns named **Title**, **FirstName** and **WorkPhone**. Note this list is a standard SharePoint **Contacts** list in which the site column named **Title** has been configured with a **DisplayName** property of **Last Name**.

var requestUri = \_spPageContextInfo.webAbsoluteUrl +

"/\_api/Web/Lists/GetByTitle('Customers')/Items" +

"?$select=Title,FirstName,WorkPhone";

1. Now call the **ajax** function and setup up the callback methods as you did in the previous exercise. The one difference is that you should name the callback method for success **onListDataReturned** instead of **onDataReturned** as you did in previous exercises.

function onGetCustomersList() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

// begin work to call across network

var requestUri = \_spPageContextInfo.webAbsoluteUrl +

"/\_api/Web/Lists/GetByTitle('Customers')/Items" +

"?$select=Title,FirstName,WorkPhone";

// execute AJAX request

$.ajax({

type: "GET",

url: requestUri,

contentType: "application/json",

headers: { Accept: "application/json;odata=verbose" },

success: onListDataReturned,

error: onError

});

}

function onListDataReturned(data) {

}

function onError(err) {

$("#results").text("ERROR: " + JSON.stringify(err));

}

1. Implement the **onListDataReturned** function as follows to create an HTML list from the items in the customers list which displays customers in a *{Last Name, First Name}* format.

function onListDataReturned(data) {

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

var odataResults = data.d.results;

$("<h2>").html("Customers").appendTo("#results");

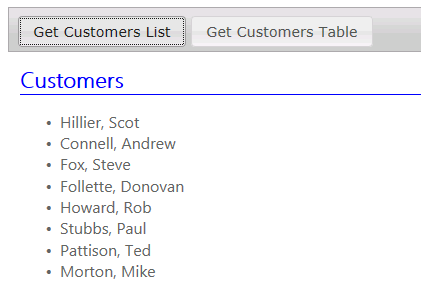
// set rendering template

var renderingTemplate = "<li>{{=Title}}, {{=FirstName}}</li>";

$.template("tmplLists", renderingTemplate);

$("#results").append($("<ul>").append($.render(odataResults, "tmplLists")));

}

1. Test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 4** link to navigate to the page named **Exercise4.aspx**. Click the **Get Customers List** command button and ensure you can see that your code displays a list of customers. Do you recognize any of them?
   * 1. 
2. Now that you have written the code to display the customers as an HTML list, you will modify the **onGetCustomersTable** function to display the same set of customers as an HTML table. Begin by copying-and-pasting the implementation of the **onGetCustomersList** into the **onGetCustomersTable**. Next, change the name of the call back function to **onTableDataReturned**. Also, add a new function named **onTableDataReturned**.

function onGetCustomersTable() {

// clear resultsArea and add spinning gears icon

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

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

// begin work to call across network

var requestUri = \_spPageContextInfo.webAbsoluteUrl +

"/\_api/Web/Lists/GetByTitle('Customers')/Items" +

"?$select=Title,FirstName,WorkPhone";

// execute AJAX request

$.ajax({

type: "GET",

url: requestUri,

contentType: "application/json",

headers: { Accept: "application/json;odata=verbose" },

success: onTableDataReturned,

error: onError

});

}

function onTableDataReturned(data) {

}

1. Implement the **onTableDataReturned** function to render the content of the customers tables as an HTML table.

function onTableDataReturned(data) {

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

var odataResults = data.d.results;

$("<h2>").html("Customers").appendTo("#results");

// set rendering template

var tableHeader = "<thead>" +

"<td>Last Name</td>" +

"<td>First Name</td>" +

"<td>Work Phone</td>" +

"</thead>";

var table = $("<table>").append($(tableHeader));

var renderingTemplate = "<tr>" +

"<td>{{=Title}}</td>" +

"<td>{{=FirstName}}</td>" +

"<td>{{=WorkPhone}}</td>" +

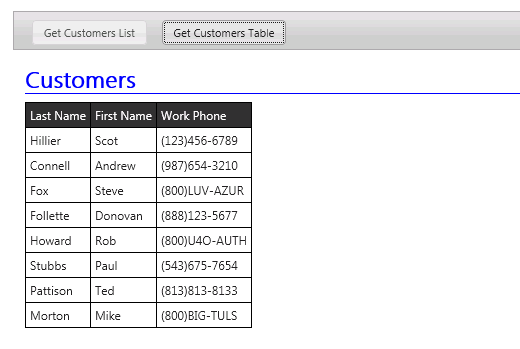
"</tr>";

$.template("tmplTable", renderingTemplate);

table.append($.render(odataResults, "tmplTable"));

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

}

1. Test your work. Press the **{F5}** key to start the Visual Studio debugger. When the debugger is finished initializing and the Internet Explorer becomes the active window, click the **Exercise 4** link to navigate to the page named **Exercise04.aspx**. Click the **Get Customers Tables** command button and ensure you can see that your code displays the customers in an HTML table.
   * 1. 
2. You have now finished all the steps of this lab.

## Summary

* 1. In this lab, you learned how to call into the SharePoint 2010 CSOM REST API using JavaScript and the jQuery. You learned to into this API using the getJSON function of the jQuery library and to process JSON-based results. You have also learned how to leverage the jsRender library to create HTML output using a template-based approach.