## Programming with the Social APIs & Office Services

**Lab Time**: 60 minutes

**Lab Folder**: [[StudentFolder]]\OfficeServices

**Lab Overview**: In this lab you will use the new client-side API’s in the CSOM to read and write to a user’s social feed. After working with the social CSOM API you will then explore some of SharePoint Server features such as Excel Services, Word Automation Services, PowerPoint Automation Services and Translation Services.

### Exercise 1: Working with a User’s Social Feed

In this exercise you will create a client application that will write to the social feed of a user.

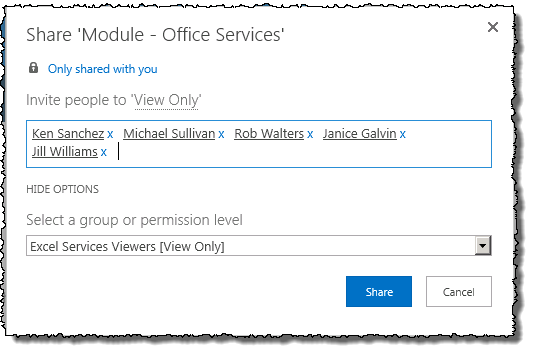
1. Setup a new site collection and create a few users for this lab:
   1. Ensure you are logged into the **WingtipServer** server as **WINGTIP\Administrator**.
   2. Run a PowerShell script, found in the root lab folder for this module:
      1. Right-click **SetupModule.ps1** and select **Run with PowerShell**. This file can be found in the files associated with this lab:

[..]\OfficeServices

* 1. When the script completes, it will launch a new browser and navigate to the lab site collection: <http://officeservices.wingtip.com>. The script also created five new users and add them to the local Remote Desktop Users group.
  2. Close the PowerShell console window.

#### Grant Users Access to the Lab Site Collection

1. Using Internet Explorer navigate to <http://officeservices.wingtip.com>.
2. Grant Ken Sanchez access to the site:
   1. Click the **Share** link in the top-right corner of the page.
   2. In the **Share** dialog, within the **Invite people to ‘View Only’** section type **Ken Sanchez** the box.
   3. A drop-down box will appear. Select **Ken Sanchez**.



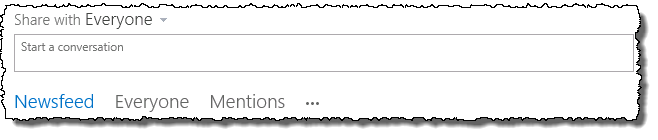
1. Repeat the above steps to add the following users access to the site:
   1. **Michael Sullivan** (WINGTIP\michael.sullivan)
   2. **Rob Walters** (WINGTIP\rob.walters)
   3. **Janice Galvin** (WINGTIP\janice.galvin)
   4. **Jill Williams** (WINGTIP\jill.williams)

#### Login as a Sample User, Create a My Site and Update their Social Feed

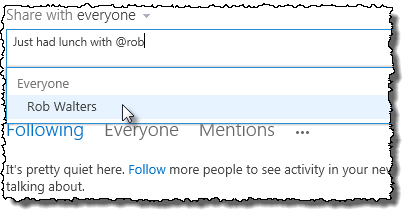
1. Launch a new instance of Internet Explorer, logged in as Ken Sanchez:
   1. Hold down **[SHIFT]** and right-click Internet Explorer (**Start 🡪 All Programs 🡪 Internet Explorer**) and select **Run as different user**.
   2. When prompted, login as **Ken Sanchez**: **WINGTIP\ken.sanchez** | **Password1**.
   3. You may get some prompts to login to various sites… cancel every request and navigate to <http://officeservices.wingtip.com>. You may be prompted to login to the site again.
2. Once the site loads, click the **Sites** link in the top-right of the page next **Ken Sanchez’s** name. This will start the process of creating the user’s My Site. You may need to login again.

Usually all these logins wouldn’t happen in a production environment. Your environments would typically be configured to trust any site in your domain and automatically pass along your username to automatically log you in.

1. Click **OK** in the **Get the most out of SharePoint** prompt.
2. Every few seconds, click the click the **Newsfeed** link to reload the page until the **My Site** is created.
3. Once the page loads, enter a message into the **Share with everyone / Start a Conversation** box:



1. Mention someone in a message by using the “@” notation. Enter “**Just had lunch with @Rob**”. When Rob Walter’s name appears, select it and press **[ENTER]** to enter the message.

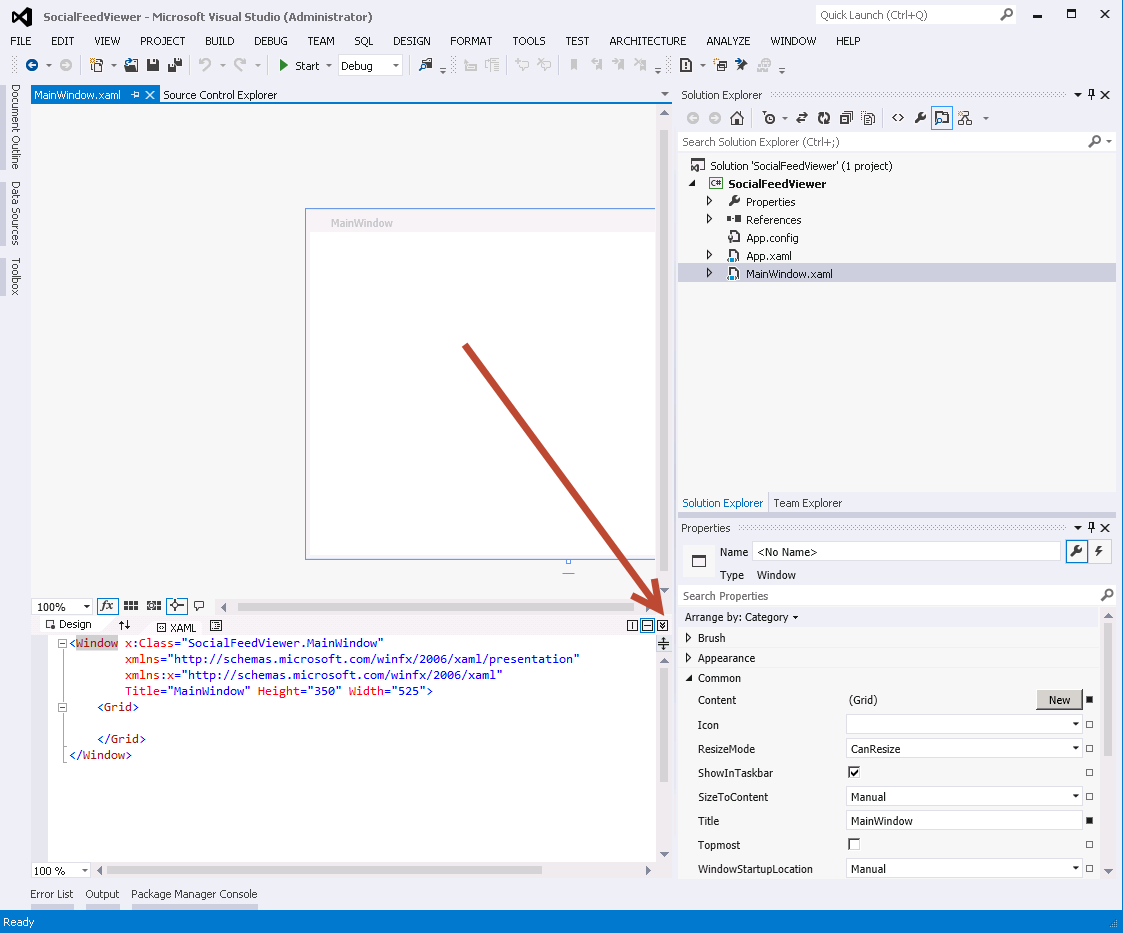


At this point you have loaded some sample users and added some messages to their social feed. Feel free to repeat the last section a few times using different users you previously created. If you login with a different user, you can click the **Everyone** link to view all the messages in the entire company’s social feed. From this view you can easily follow other employees.

### Exercise 2: Programming the Social Feed

In this exercise you will create a client application that will write to the social feed of a user.

1. Launch Visual Studio:
   1. **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**.
2. Create a new project:
   1. Select **File 🡪 New 🡪 Project**.
   2. In the left-hand section of the **New Project** dialog, select **Templates 🡪 Visual C# 🡪 Windows 🡪 WPF Application**.
   3. In the top part of the dialog, select **.NET Framework 4.5**.
   4. Give the project a name of **SocialFeedViewer**.
   5. Click **OK** to create the project.
3. Update the UI for the application:
   1. Right-click the **MainWindow.xaml** file in the **Solution Explorer** tool window and select **View Designer**.
   2. In the lower right portion of the designer, click the down arrow (shown in the following figure) to maximize the view:



* 1. Next, at the bottom of the Visual Studio 2012 window, select the **XAML** tab to view the markup for the page.
  2. Add the following markup to the **<Grid>** element:

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*" />

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal">

<TextBox Name="AccountNameTextBox" Width="400" />

<Button Name="GetActivitiesButton" Content="Get Activties" Width="100" Width="507" Height="263" Margin="5,10,5,0" />

</StackPanel>

<ListBox Name="FeedThreadsListBox" Grid.Row="1" />

<StackPanel Grid.Row="2" Orientation="Horizontal">

<TextBox Name="ThreadCountTextBox" Width="100" />

<TextBox Name="ThreadResponseTextBox" Width="300" />

<Button Name="PostReplyButton" Content="Post Reply" Width="100" />

</StackPanel>

1. At this point the application should look similar to the following figure (when viewed in the **Design** view):



1. Save the project by clicking **File 🡪 Save All**.

#### Code the Application to Query the Activity Feed

1. Add the necessary CSOM social APIs to the project:
   1. Within the **Solution Explorer** tool window, right-click **References** in the project and select **Add Reference…**.
   2. Click the **Browse** button at the bottom of the dialog and add the two assemblies found in the following folder

c:\Program Files\Common Files\Web Server Extentions\15\ISAPI

* + 1. **Microsoft.SharePoint.Client.dll**
    2. **Microsoft.SharePoint.Client.Runtime.dll**
    3. **Microsoft.SharePoint.Client.UserProfiles.dll**
  1. Click **OK** at the bottom of the dialog.
  2. Right-click **MainWindow.xaml** in the **Solution Explorer** tool window and select **View Code**.

1. Add the following statements to the top of the code file:

using Microsoft.SharePoint.Client;

using Microsoft.SharePoint.Client.Microfeed;

using Microsoft.SharePoint.Client.UserProfiles;

1. Add the following member variables to the **MainWindow** class:

ClientContext clientContext;

MicrofeedManager microfeedMgr;

Dictionary<int, string> idDictionary;

1. Add the following code to the **MainWindow()** constructor, immediately after the **InitializeComponent()** method call:

// establish a CSOM context

clientContext = new ClientContext(“http://officeservices.wingtip.com”);

1. Add the following method to the **MainWindow** class that will load all the feeds for the specified user:

private void LoadThreads()

{

try

{

string targetUser = AccountNameTextBox.Text;

// Get the MicrofeedManager object.

microfeedMgr = new MicrofeedManager(clientContext);

// Get the properties for the target user

PersonProperties personProps = new PeopleManager(clientContext).GetPropertiesFor(targetUser);

// Get Display Name and Account Name for the target user

clientContext.Load(personProps, o => o.DisplayName, o => o.AccountName);

clientContext.Load(microfeedMgr);

clientContext.ExecuteQuery();

// Specify the feed content that you want to retrieve.

MicrofeedRetrievalOptions retrievalOptions = new MicrofeedRetrievalOptions();

retrievalOptions.IncludedTypes = MicroBlogType.RootPost;

retrievalOptions.ThreadCount = 5;

// Get all of the target owner's posts and activities

var threads = microfeedMgr.GetPublishedFeed(

personProps.AccountName,

retrievalOptions,

MicrofeedPublishedFeedType.Full);

clientContext.ExecuteQuery();

//Create a dictionary to store the thread identifiers

idDictionary = new Dictionary<int, string>();

// clear out of the existing feeds

FeedPosts = new List<string>();

FeedThreadsListBox.Items.Clear();

for (int i = 0; i < threads.Value.Count; i++)

{

MicrofeedThread thread = threads.Value[i];

// Keep only user-sourced threads (not events).

if (thread.DefinitionName == "Microsoft.SharePoint.Microfeed.UserPost")

{

//Save thread identifier

idDictionary.Add(i, thread.Identifier);

// get the posts

FeedThreadsListBox.Items.Add(string.Format("{0}: {1}. {2}",

personProps.DisplayName,

(i + 1),

thread.RootPost.Content));

}

}

}

catch (Exception x)

{

MessageBox.Show(x.Message);

}

}

1. Update the application user interface:
   1. Right-click the **MainWindow.xaml** file in the **Solution Explorer** tool window and select **View Designer**.
   2. Select the button **Get Activities**.
   3. In the **Properties** tool window, click the event button to see all the events:



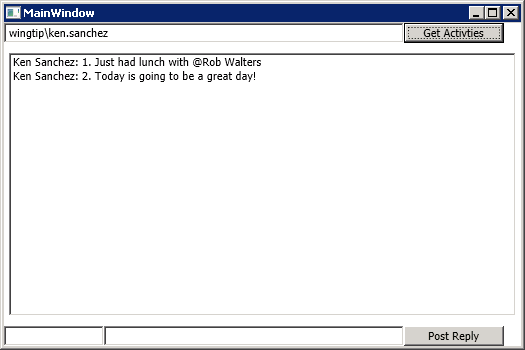
* 1. Find the **Click** event and double-click the box to the right of it. This will create a new event handler for clicking the button.
  2. When the code view loads, add the following code to the **GetActivitiesButton\_Click()** event handler:

LoadThreads();

1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**. This will compile and execute the app.
2. When the application loads, enter a username into the top most **TextBox** and click **Get Activities**. You will see the contents of this user’s activity feed in the **ListBox**:



1. Close the application to stop debugging.

#### Code the Application to Write to the Activity Feed

1. Right-click **MainWindow.xaml** in the **Solution Explorer** tool window and select **View Code**.
2. Add the following method to write to the activity feed:

private void PostReploy()

{

try

{

string threadId = string.Empty;

// Get the thread identifier

idDictionary.TryGetValue((Convert.ToInt32(ThreadCountTextBox.Text) - 1), out threadId);

// Define properties for the reply.

MicrofeedPostOptions postOptions = new MicrofeedPostOptions();

postOptions.Content = ThreadResponseTextBox.Text;

// Register the reply.

microfeedMgr.PostReply(threadId, postOptions);

// Make the changes on the server.

clientContext.ExecuteQuery();

MessageBox.Show("Reply Posted!");

}

catch (Exception x)

{

MessageBox.Show(x.Message);

}

}

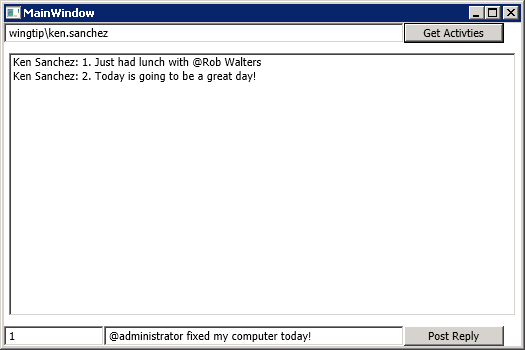
1. Wire up the method you just added to the Post Reply button’s Click event:
   1. Right-click **MainWindow.xaml** in the **Solution Explorer** tool window and select **View Designer**.
   2. Select the button **Post Reploy**.
   3. In the **Properties** tool window, click the event button to see all the events:
   4. Find the **Click** event and double-click the box to the right of it. This will create a new event handler for clicking the button.
   5. When the code view loads, add the following code to the **PostReployButton\_Click()** event handler:

PostReploy();

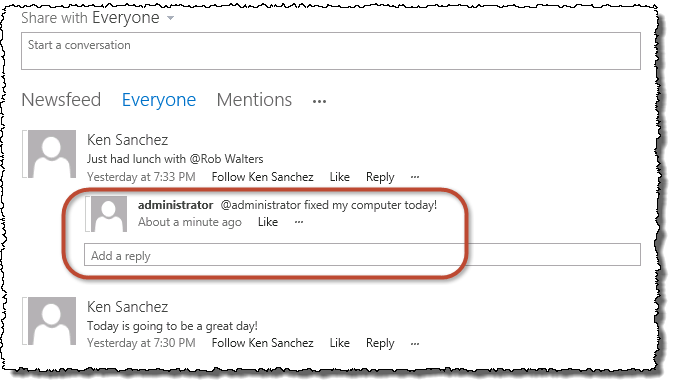
1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**. This will compile and execute the app.
2. When the application loads, enter a username into the top most **TextBox** and click **Get Activities**. You will see all the same entries from the last test.
3. Post a reply:
   1. In the **ThreadCountTextBox**, the TextBox in the lower left corner, enter **1**.
   2. In the **ThreadResponseTextBox**, enter “**@administrator fixed my computer today!**”.
   3. Click the **Post Reply** button.



1. If you go back to any of the user’s My Sites and view the everyone feed, you’ll see the response:



1. Close the application to stop debugging.

In this exercise you created an application that used the new CSOM social API to read and write to a user’s social feed.

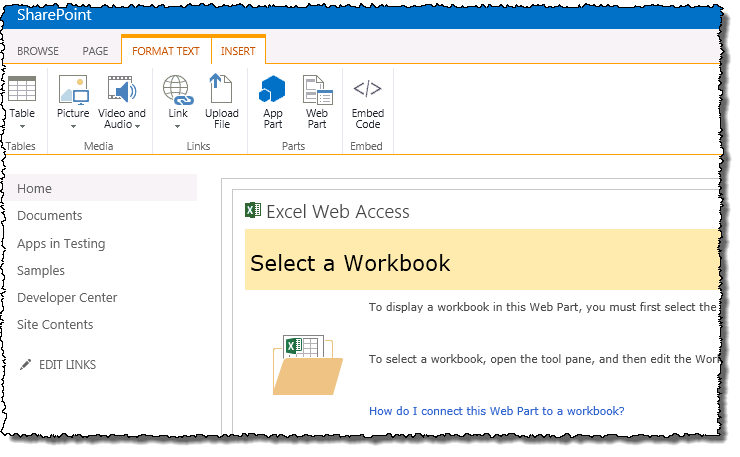
### Exercise 3: Excel Services – JavaScript Object Model

In this lab, you will use JavaScript to display data from a spreadsheet presented in the Excel Web Access Web Part.

1. Using Internet Explorer navigate to <http://officeservices.wingtip.com>.
2. Upload a sample spreadsheet:
   1. Using the **Quick Launch** navigation to the left, select the **Documents** library.
   2. Upload the **MiniCRM.xmlx** workbook, found in the files associated with this exercise:

[..]\OfficeServices\Exercises\Ex3

1. Now, go back to the homepage of the site by clicking **Home** in the **Quick Launch**.
2. Add the Excel Web Access Web Part to the page:
   1. Using the ribbon, click the **Page** tab and select the **Edit** button.
   2. Click **REMOVE THIS** to the right of **Get Started with Office Apps** to clean up the page a bit.
   3. Place the cursor at the top of the Web Part Zone and, using the ribbon, select the **Insert** tab and then the **Web Part** button in the **Parts** group.
   4. Select the **Excel Web Access** Web Part from the **Business Data** category and click **Add**:



1. Next, configure the **Excel Web Access Web Part** to display our **MiniCRM.xlsx** workbook:
   1. In the **Excel Web Access Web Part**, click **Click Here to Open the Tool Pane**.
   2. In the tool pane, click the **Workbook …** (ellipse) button.
   3. In the **Select an Asset** dialog, select the **MiniCRM.xlsx** file from the Documents library and click **Insert**.
   4. In the tool pane, click **OK**.
2. At this point you should see the workbook rendered in the Excel Web Access Web Part. Next, you will add some JavaScript to the page to read the data from the workbook in the Web Part:
   1. If the page is not in **Edit** mode, get into edit mode following the steps you previously performed.
   2. Place the cursor at the top of the Web Part Zone and, using the ribbon, select the **Insert** tab and then the **Web Part** button in the **Parts** group.
   3. Select the **Script Editor** Web Part from the **Media and Content** category and click **Add**:
3. Add JavaScript to the Script Editor Web Part:
   1. While in **Edit** mode, select the **Script Editor** Web Part.
   2. Using the ribbon, select the **Web Part** tab and then click the **Web Part Properties** button.
   3. In the **Script Editor** Web Part, click **EDIT SNIPPET** which is right-aligned in the rendered Web Part.
   4. Add the following JavaScript code to the provided box and click Insert.

<script type="text/javascript">

var ewa;

function ewaStart(){

Ewa.EwaControl.add\_applicationReady(ewaConnect);

}

function ewaConnect() {

ewa = Ewa.EwaControl.getInstances().getItem(0);

if (ewa) {ewa.add\_activeCellChanged(ewaCellChanged);}

}

function ewaCellChanged(rangeArgs) {

var sheetName = rangeArgs.getRange().getSheet().getName();

var col = rangeArgs.getRange().getColumn();

var row = rangeArgs.getRange().getRow();

var value = rangeArgs.getFormattedValues();

alert('Active Cell is now at Row' + (row + 1) + ', Column' + (col + 1) + ', with Value of ' + value);

}

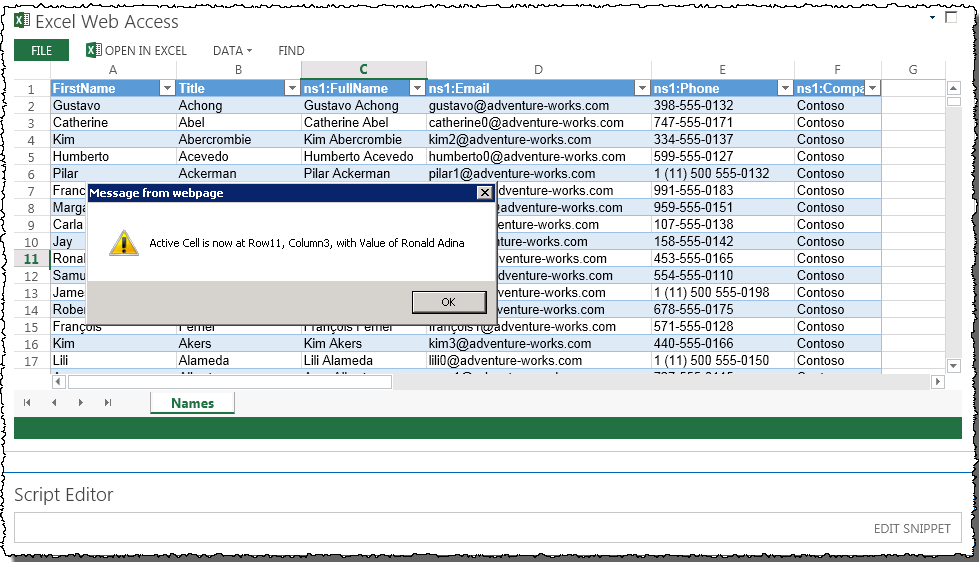
ewaStart();

</script>

This code can also be found in the **excelJSON.js** file found in the following location:

**[..]\OfficeServices\Exercises\Ex3**

1. In the ribbon, under the **Page** tab, click the **Save** button.
2. Click on the spreadsheet and verify that a message is displayed showing the current active cell.



In this exercise you used JavaScript and the Excel Services JavaScript Object Model to interact with the Excel Web Access Web Part.

### Exercise 4: Excel Services – Retrieve Excel Data as OData

In this lab, you will create a web part that leverages the OData interface to Excel Services.

1. Launch **Visual Studio** **2012** by selecting **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**.
2. Create a new project:
   1. Select **File 🡪 New 🡪 Project**.
   2. Find the **SharePoint 2013 – Empty Project** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 SharePoint Solutions** section.
   3. Give the project a name of **ExcelOData**.
   4. Click **OK** to create the project.
   5. In the **SharePoint Customization Wizard**, use the following values to complete the wizard and click **Finish**.
      1. **What site do you want to use for debugging?** <http://officeservices.wingtip.com>
      2. **What is the trust level for this SharePoint solution?** Deploy as farm solution
3. Add the jQuery library to the project:
   1. Right-click the **ExcelOData** project in the **Solution Explorer** tool window and select **Add 🡪 SharePoint “Layouts” Mapped Folder**.
   2. Right-click the **Layouts \ ExcelOData** folder and select **Add 🡪 Exiting Item**.
   3. Find the JavaScript files provided with this exercise and add them to the folder. The files will be found in the following folder:

[..]\OfficeServices\Exercises\Ex4

1. Add a Visual Web Part to the project:
   1. Right-click the **ExcelOData** project in the **Solution Explorer** tool window and select **Add 🡪 New Item**.
   2. In the **Add New Item** dialog, select **Visual Web Part** from the **Visual C# Items 🡪 Office / SharePoint** category and click **Add**.
2. Code the Visual Web Part:
   1. In the **Solution Explorer**, open **VisualWebPart1.ascx** for editing.
   2. Add the following code to the file to make an OData call to the spreadsheet and display data:

You can find the code for this snippet in **excelREST.js** along with the other files associated with this exercise.

<script type="text/javascript" language="javascript" src="/\_layouts/15/ExcelOData/jquery-1.6.2.min.js"></script>

<div id="resultsDiv">Loading...</div>

<script type="text/javascript">

$(document).ready(function () {

var e = ExecuteOrDelayUntilScriptLoaded(showSheet, "sp.js");

});

function showSheet() {

Results = {

element: '',

url: '',

init: function (element) {

Results.element = element;

Results.url = \_spPageContextInfo.webAbsoluteUrl + "/\_vti\_bin/ExcelRest.aspx/Shared%20Documents/MiniCRM.xlsx/OData/Table1";

},

load: function () {

$.ajax(

{

url: Results.url,

method: "GET",

headers: {

"accept": "application/json",

},

success: Results.onSuccess,

error: Results.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].ns1FullName;

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

html += results[i].ns1Email;

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

}

html += "</table>";

Results.element.html(html);

},

onError: function (err) {

alert(JSON.stringify(err));

}

}

Results.init($('#resultsDiv'));

Results.load();

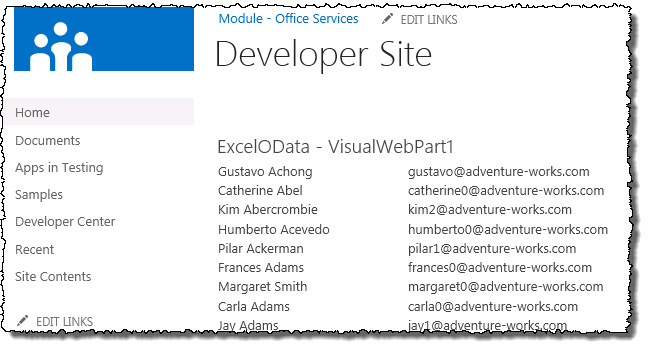
}

</script>

1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the <http://officeservices.wingtip.com> site.
3. Add the Web Part to the page:
   1. Using the ribbon, click the **Page** tab and select the **Edit** button.
   2. Place the cursor at the top of the Web Part Zone and, using the ribbon, select the **Insert** tab and then the **Web Part** button in the **Parts** group.
   3. Select the **ExcelOData – VisualWebPart1** from the **Custom** category and click **Add**:
4. Using the ribbon, click the **Page** tab and then click the **Save** button.
5. Notice the Web Part, after loading, shows data from the Excel workbook as shown in the following figure:



1. Close the application to stop debugging.

In this exercise you used the Excel OData service to retrieve and display data on the page from an Excel workbook.