## Implementing and Consuming an ODATA Service

**Lab Time**: 60 minutes

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

**Lab Overview**: In this lab you will create an OData web service which allows clients to perform CRUD operations against customer data in a SQL Server database. After that, you will create an ASP.NET application that uses your new OData web service API to provide a user experience which makes it possible to view, add, edit and delete customers from the SQL Server database.

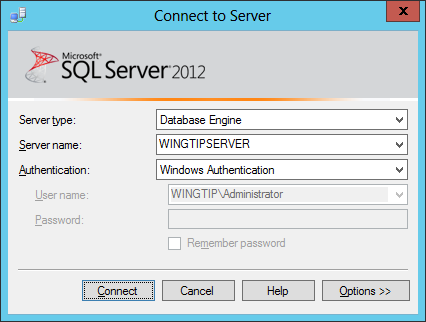
### Exercise 1: Create and Populate the Wingtip CRM Database in SQL Server

In this exercise you will open the solution for the lab and examine the contents.  
(Note: you will only need to do this if you have not already run this script for a previous exercise in your course)

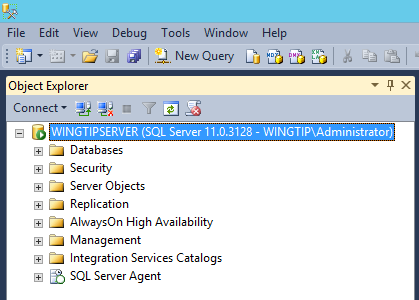
1. Launch SQL Server Management Studio and connect to the default SQL Server instance on **WingtipServer**.
   1. Press the Windows key to display the Window Start page.
   2. On the Windows Start page, locate and click the **SQL Server Management Studio** tile to launch this application.



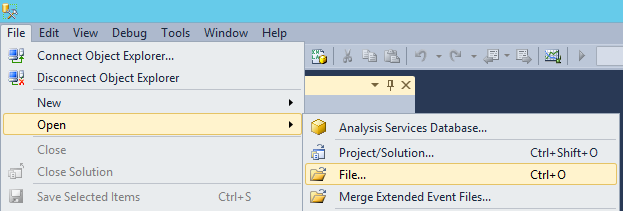
* 1. When you are prompted with the **Connect to Server** dialog, select **WingtipServer** as the server name and click **Connect**.



* 1. Once SQL Server Management Studio has started, you should be able to see a tree view which shows database objects in the Object Explorer.



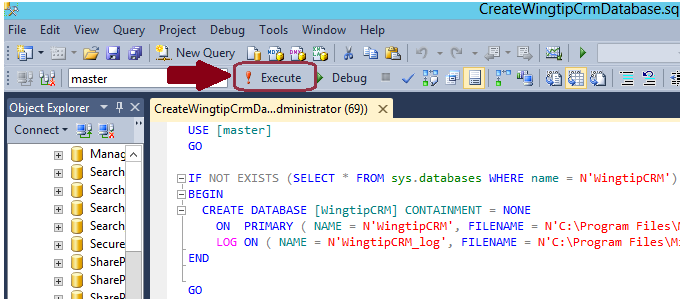
1. Execute the SQL script named. **CreateWingtipCrmDatabase.sql** to create the **WingtipCRM** database.
   1. From the **File** menu in SQL Server Management Studio, select the menu command **File  Open  File…**



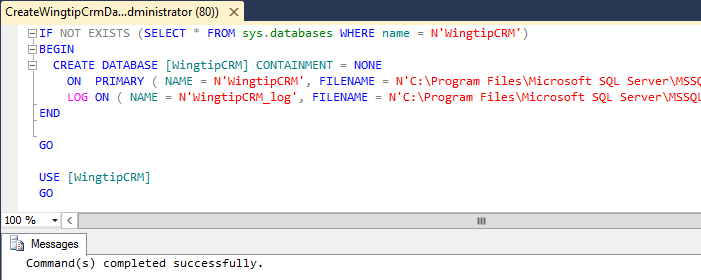
* 1. When prompted by the **Open File** dialog, select the script named **CreateWingtipCrmDatabase.sql** at the following path and then click Open.

C:\Student\Setup\CreateWingtipCrmDatabase.sql

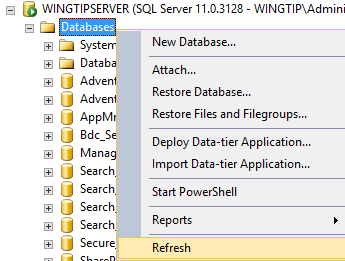
* 1. Once the script named **CreateWingtipCrmDatabase.sql** is open, take a moment to examine what’s inside. You should be able to see this script carries out the following tasks.
     1. Create a new SQL Server database named **WingtipCRM**.
     2. Create a new table named **Customers** in the **WingtipCRM** database.
     3. Add SQL Login for the Windows account **BUILTIN\IIS\_IUSRS** and configure permissions to the **WingtipCRM** database.
     4. Add SQL Login for domain account **WINGTIP\Domain Users** and configure permissions to the **WingtipCRM** database.
     5. Add sixteen samples records into the **Customers** table.
  2. Make sure that window displaying **CreateWingtipCrmDatabase.sql** is the active window. Then click the **Execute** button in the toolbar with the exclamation point to execute the script.



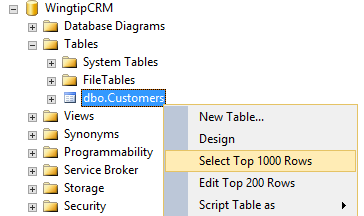
* 1. The script should execute without any errors and display **Command(s) completed successfully** in the Messages window.



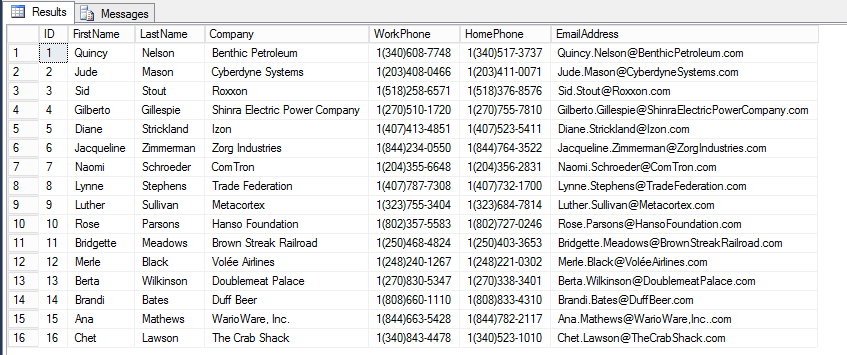
1. Verify that the database has been properly created.
   1. In the **Object Explorer** in SQL Server Management Studio, locate the **Database** node in the tree view control on the left-hand side of screen. Right-click the **Database** node and then click the **Refresh** menu command to display the database that has just been created.



* 1. After the collection of databases has been refreshed, look down and locate the new database named **WingtipCRM**.
  2. Expand the **WingtipCRM** node and then the **Tables** node and verify you can see a table named **dbo.Customers**.
  3. Right-click on the **dbo.Customers** node and ten click the menu command with the caption of **Select Top 1000 Rows**.



* 1. Verify that you can see a set of customer records that have been added to the **Customers** table.



You have now created the **WingtipCRM** database that you will use as a sample database for the remainder of this lab as well as many of the labs that come afterward. At this point you can close the SQL Server Management Studio or you can keep it open if you want to monitor changes to the data over the course of the following exercises.

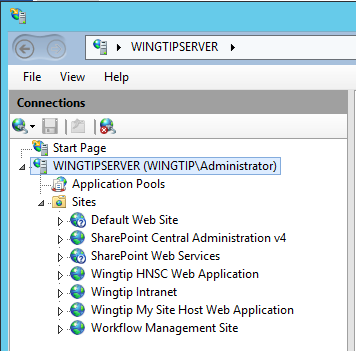
### Exercise 2: Create an IIS Website to Host a Web Service

In this exercise you will run a script to create a new IIS Website. You will use this IIS Website to host a web service you will build in the exercise after this one.

1. Launch the **Internet Information Services (IIS) Manager**.
   1. Press the Windows key to display the Window Start page.
   2. On the Windows Start page, locate and click the **Internet Information Services (IIS) Manager** tile to launch this application.

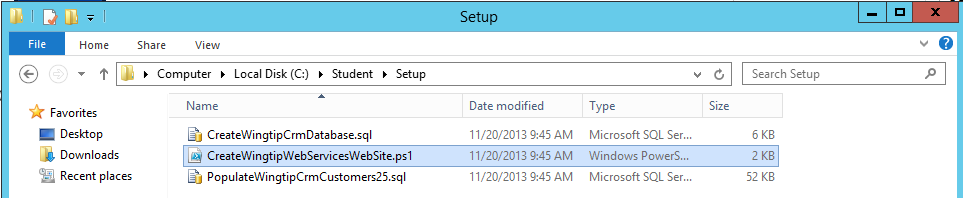


* 1. Once the **IIS Manager** has started, expand the **Sites** node in the tree view control to examine the existing set of IIS Websites that have been created on **WINGTIPSERVER**.

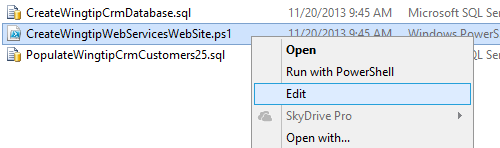


Most of the existing IIS websites have been created by SharePoint and should not be configured directly in the IIS Manager.

1. Run the PowerShell script named **CreateWingtipWebServicesWebSite.ps1** to create a new IIS Website.
   1. Switch from the IIS Manager over to the Windows Explorer.
   2. Use the Windows Explorer to locate the PowerShell script file named **CreateWingtipWebServicesWebSite.ps1** which is located in the folder at **C:\Student\Setup**.

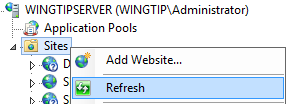


* 1. In the Windows Explorer, right-click on **CreateWingtipWebServicesWebSite.ps1** and select the **Edit** command to open the script in the Windows PowerShell ISE.

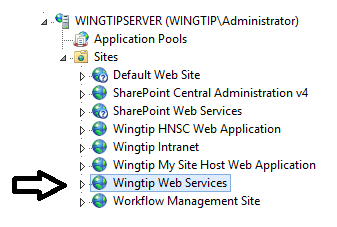


* 1. Once **CreateWingtipWebServicesWebSite.ps1** opens in the PowerShell ISE, take a moment and inspect what’s inside. Here’s an overview of the tasks performed by this script.
     1. It checks to see if an IIS website named **Wingtip Web Services** already exists.
     2. If the **Wingtip Web Services** IIS website does not yet exist, the script creates it.
     3. It adds an entry into the system HOST file so to redirect incoming HTTP request targeting the **webservices.wingtip.com** domain to the local IP address of **127.0.0.1**.
  2. Execute the script by pressing the {F5} key. You should not see any errors when running the script.

1. Verify that the IIS Website has been created.
   1. Switch back over to the IIS Manager.
   2. Right-click on the **Sites** node and click **Refresh** to repopulate the collection of IIS websites.



* 1. Verify that a new IIS website named **Wingtip Web Services** has been created.

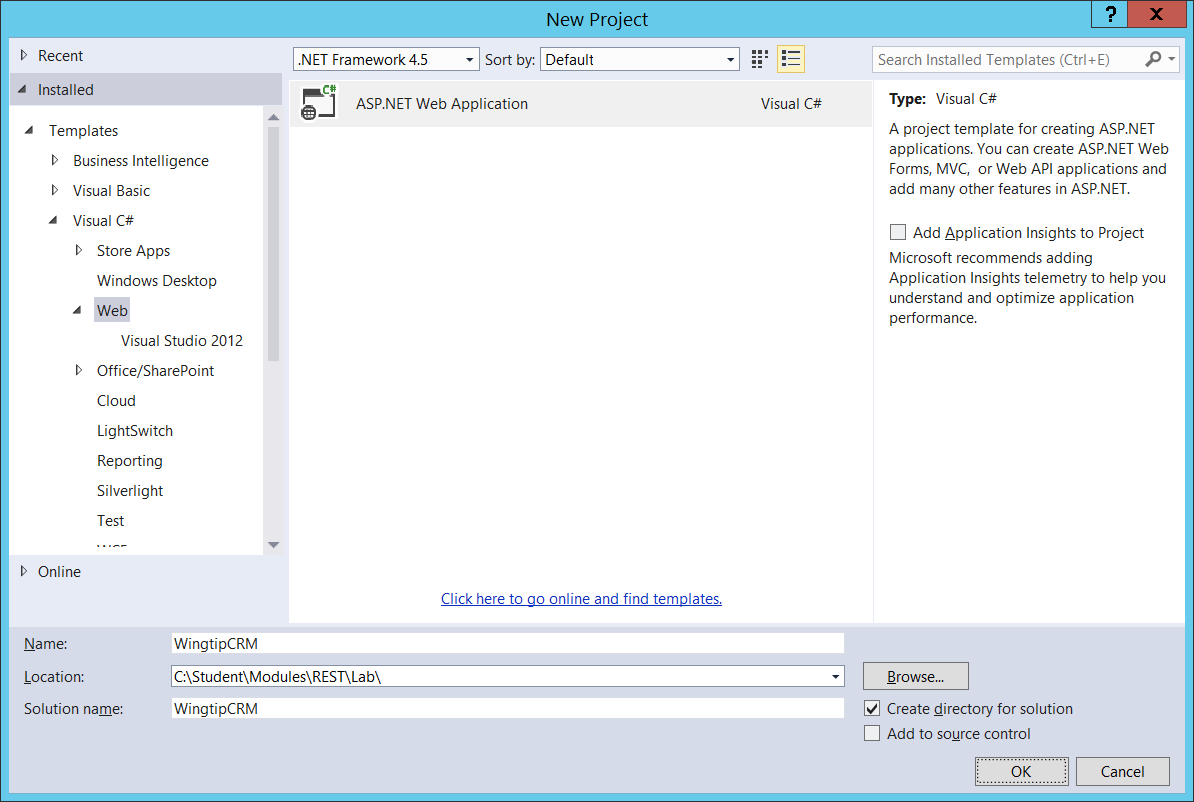


1. If they are still open, close the IIS Manager and the Windows PowerShell ISE.

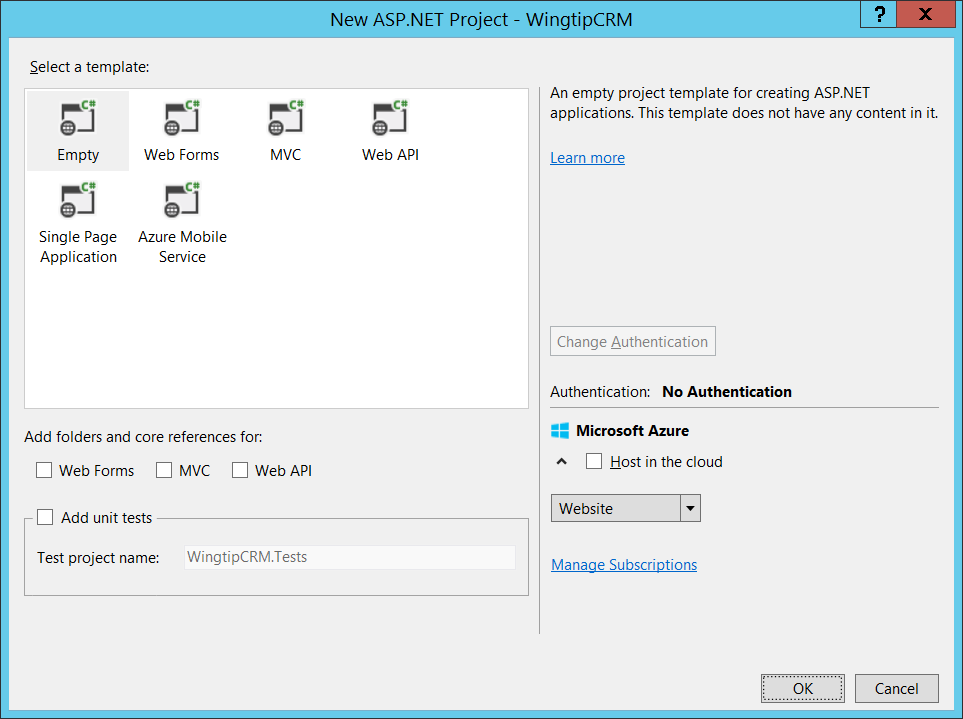
### Exercise 3: Create an OData Service using the Entity Framework and WCF

In this exercise you will open the solution for the lab and examine the contents.

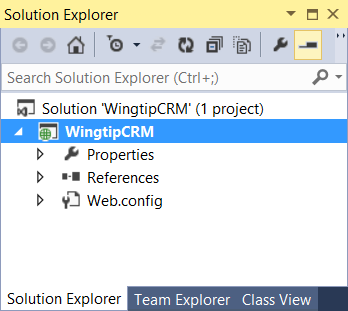
1. Launch Visual Studio 2013 As Administrator.
   1. Windows Keyboard Key 🡪 Right click on the Visual Studio 2013 tile and select Run as administrator.
2. Create a new ASP.Net Web Application project in Visual Studio 2013 to implement the web service.
   1. From the **File** menu, select the menu command **File 🡪 New 🡪 Project…**.
   2. On the left side of the **New Project** dialog, select **Templates 🡪 Visual C# 🡪 Web**.
   3. Select the project template named **ASP.NET Web Application**.
   4. Enter a **Name** of **WingtipCRM**.
   5. Enter a location of **C:\Student\Modules\REST\Lab**.
   6. Once the New Project dialog looks like the following screenshot, click the **OK** button.



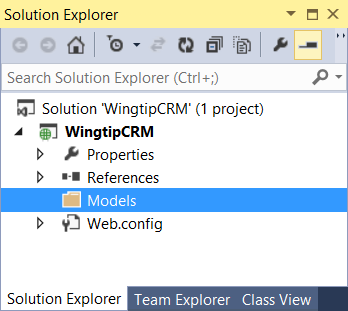
* 1. Next, Visual Studio will prompt you with the **New ASP.NET Project** dialog. Select the Empty template as shown in the following screenshot. Also make sure the **Authentication** setting is set to **No Authentication** and the **Host in the cloud** checkbox in the Microsoft Azure section is unchecked. When the the **New ASP.NET Project** dialog on your screen matches the following screenshot, click **OK** to create the new project.



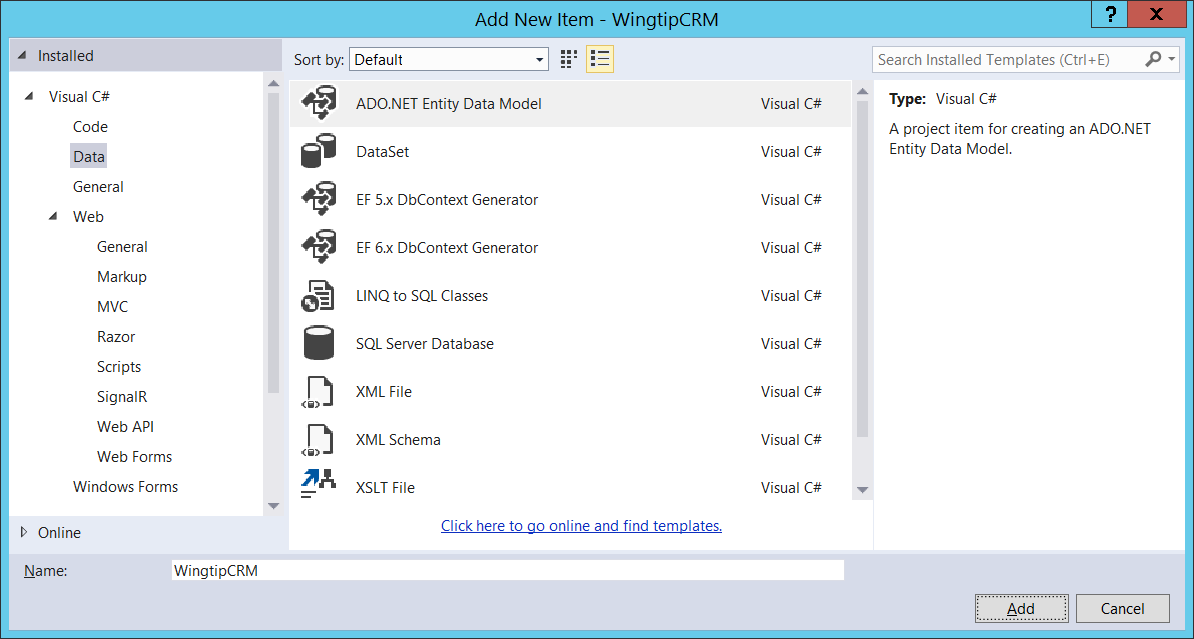
* 1. After the project has been created, you can see it initially only contains a single file which is the **web.config** file.



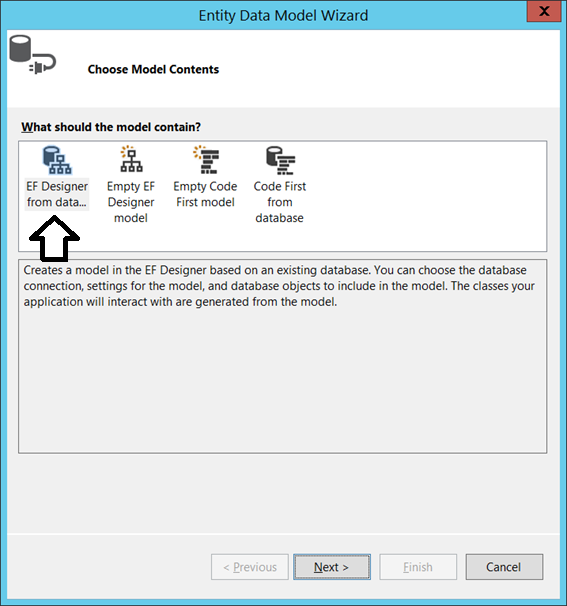
1. In the Solution Explorer, right-click on the **WingtipCRM** project and click the **Add > New Folder** command to create a new folder named **Models**.



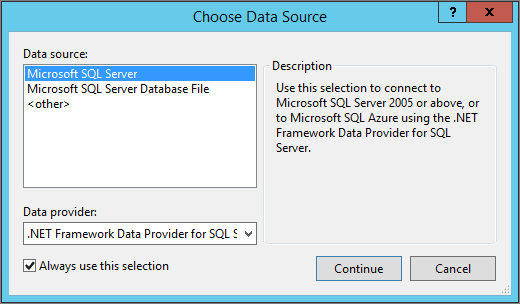
1. Add an ADO.NET Entity Model for the **WingtipCRM** Database.
   1. Right-click on the **Models** folder and click **Add > New Item**.
   2. In the left side of the **Add New Item** dialog, select **Installed 🡪** **Visual C# 🡪 Data**.
   3. Select the project item template named **ADO.NET Entity Data Model**.
   4. Enter a **Name** of **WingtipCRM.edmx**.
   5. Click the **Add** button on the bottom right to begin the process of creating the new project item.



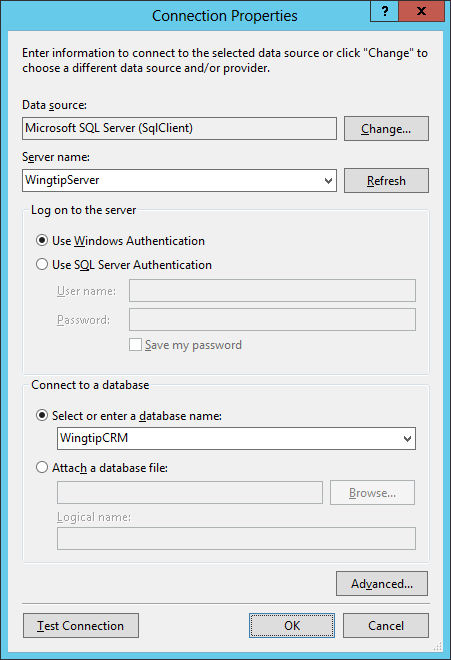
* 1. When you are prompted by **Choose Model Contents** page of the **Entity Data Model Wizard** dialog, select the first option on the left which is **EF Designer from database** as shown in the following screenshot and then click **Next**.



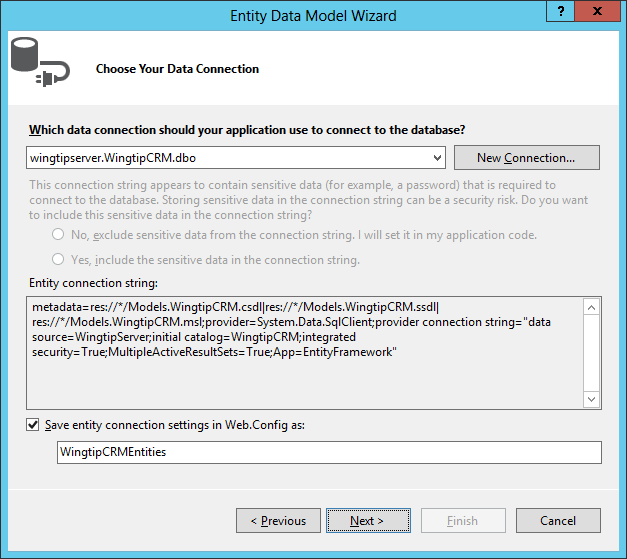
* 1. On the **Choose Your Data Connection** page of the **Entity Data Model Wizard**, you are prompted to select or create a new connection. Click the **New Connection…** button to display the **Connection Properties** dialog.
  2. If you are prompted by the **Choose Data Source** dialog, select **Microsoft SQL Server** and then click **Continue**.



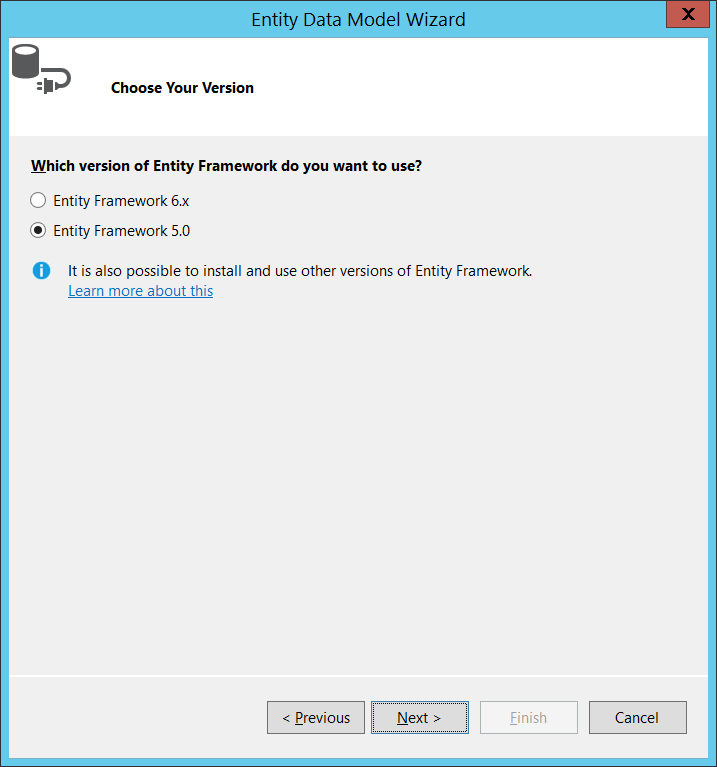
* 1. Next, you should see the **Connection Properties** dialog. Follow these steps to fill in the required information.
     1. Enter a **Server name** of **WingtipServer**.
     2. Enter a database name of **WingtipCRM**.
     3. Once the **Connection Properties** dialog looks like the following screenshot, click the **OK** button to save the connection information and return to the **Choose Your Data Connection** page of the **Entity Data Model Wizard**.



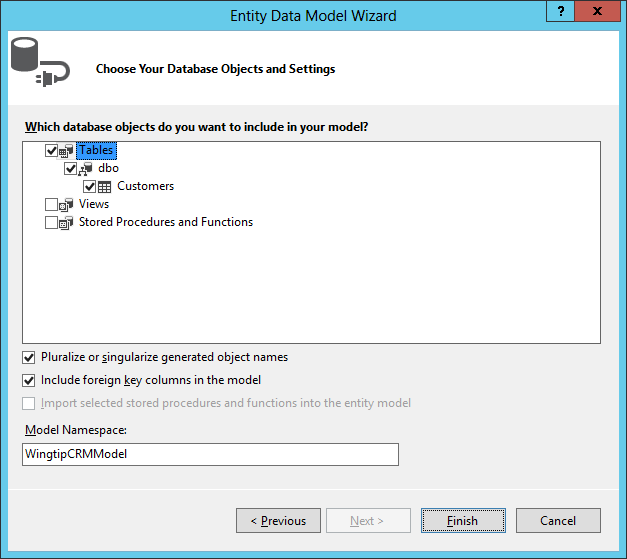
* 1. On the **Choose Your Data Connection** page of the **Entity Data Model Wizard,** leave the bottom of the page with the default settings as shown in the following screenshot so that the connection information is stored in **web.config** file under the name of **WingtipCRMEntities**. Click **Next** to continue to the **Choose Your Database Objects and Settings** page.



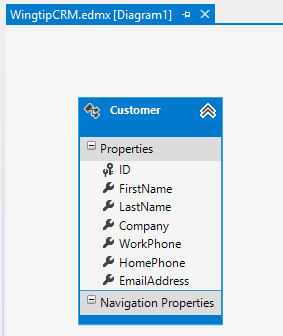
* 1. On the **Choose Your Version** for the Entity Framework page select **Entity Framework 5.0** and click **Next**.



* 1. On the **Choose Your Database Objects and Settings** page, expand the **Tables** node and select the **Customers** table. Leave the other settings on the page with their default settings as shown in the following screenshot. Click the **Finish** button to complete the task of create the entity data model.

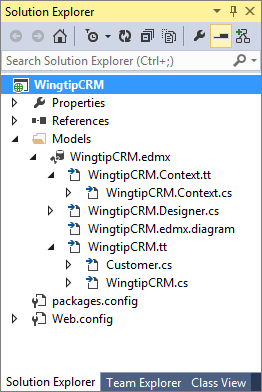


* 1. If Visual Studio prompts you regarding a Security Warning or whether it is OK to update the **web.config** file, select **OK** or **Yes** as appropriate.
  2. Once the Entity Data Model Wizard has finished its work, it will then display a Visual Studio designer that provides a visual representation of the new entity data model which contains a single entity named **Customer**.

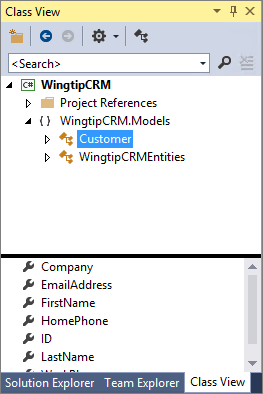


* 1. After you have examined the **Customer** entity and its properties in the visual designer, close **WingtipCRM.edmx**.

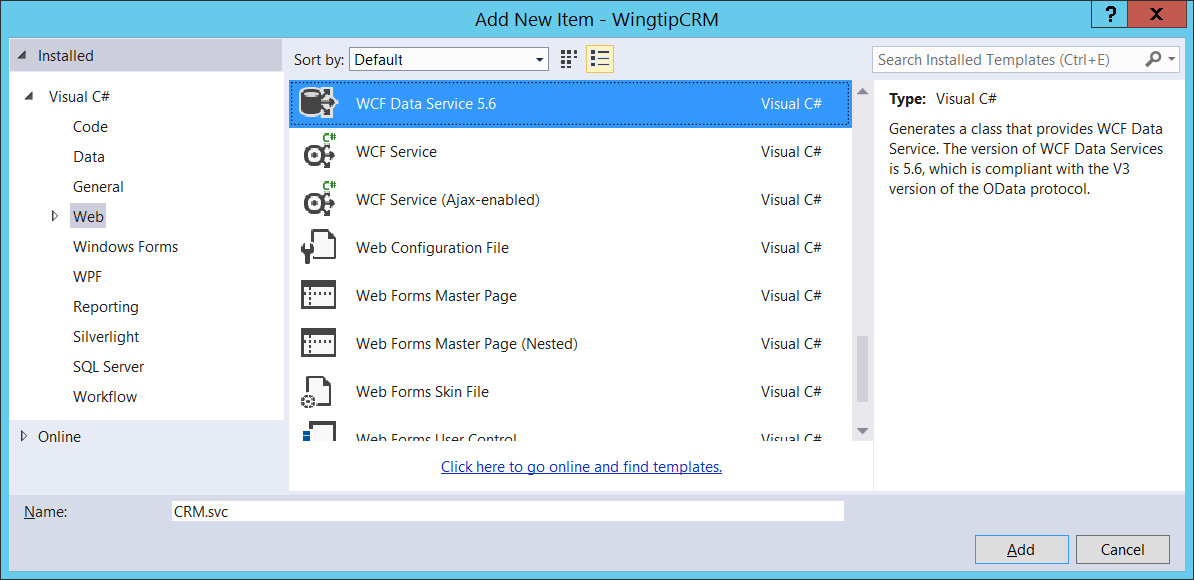
1. Once you have added a new **Customer** entity to your project, it is a good idea to run the project once in the Visual Studio debugger to ensure all the Entity Framework code is properly generated. Do this by pressing the **{F5}** key to run the project in the Visual Studio debugger. Note that the browser window will show an error because there is no page to display but that doesn’t matter. Close the browser windows and return to Visual Studio.
2. Use the Solution Explorer to examine the new files that the Entity Data Model Wizard added to the **WingtipCRM** project.
   1. You should be able to see several files have been added the project to support the new entity data model.



1. Use Class View to see what C# classes have been created.
   1. Switch from the Solution Explorer over to Class View.
   2. Expand the node with the **WingtipCRMModels** namespace.
   3. Verify that there is an entity model class named **Customer** which is associated with the **Customers** table in SQL Server.
   4. Verify that there is second class named **WingtipCRMEntities**.



1. Create a new WCF Data Service to expose the **WingtipCRM** database as an OData web service.
   1. Return to the Solution Explorer and locate the top-level node for the **WingtipCRM** project.
   2. Right-click on the **WingtipCRM** project node and then click **Add 🡪 New Item**.
   3. In the left side of the **Add New Item** dialog, select **Installed 🡪** **Visual C# 🡪 Web**.
   4. Select the project item template named **WCF Data Service 5.6**.
   5. Enter a Name of **CRM.svc**.
   6. Click the **Add** button to add the new item to your project.



* 1. When you add the WFC Data Service project item, Visual Studio adds a WCF web service file into your project named **CRM.svc** and an associated code-behind file named **CRM.svc.cs**. It also opens the **CRM.svc.cs** file so that you can examine and modify the C# code inside.
  2. Examine the C# code that was added to **CRM.svc.cs**. You can see that there are comments which are TODO tasks. For example, it instructs you to update the CRM class definition with a data source class.

public class CRM : DataService< /\* TODO: put your data source class name here \*/ >

* 1. Update the code inside the **CRM** class so it looks like the code shown in the following listing.  
     (Note: this code tells the DataService to use the WingtipCRMEntities we created earlier (in the Models folder) and inside of this entity access the Customers table with all CRUD operations enabled)

namespace WingtipCRM {

public class CRM : DataService<WingtipCRM.Models.WingtipCRMEntities> {

public static void InitializeService(DataServiceConfiguration config) {

config.SetEntitySetAccessRule("Customers", EntitySetRights.All);

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V3;

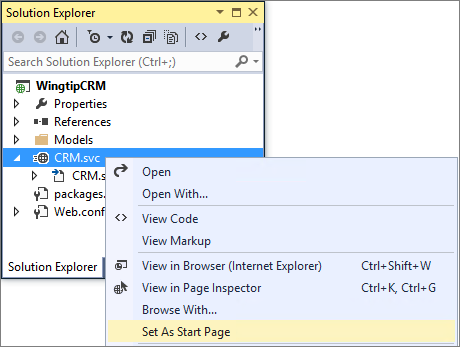
}

}

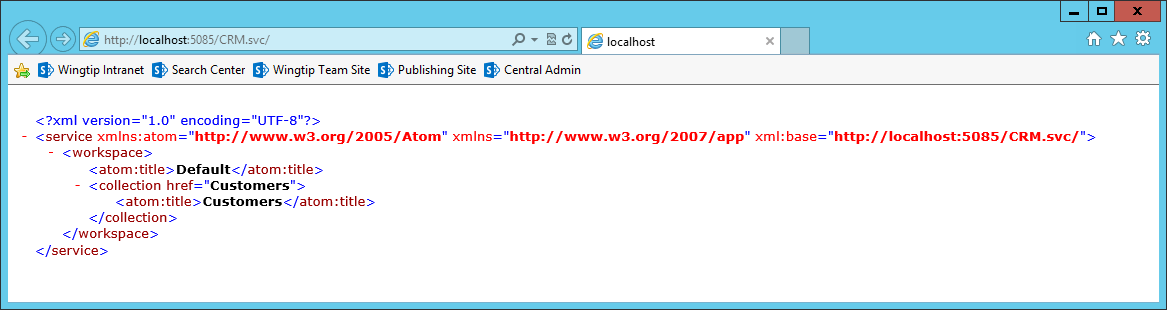
}

* 1. Save and close **CRM.svc.cs**.

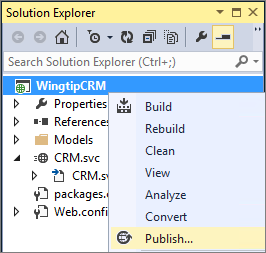
1. Now it’s time to test the web service.
   1. In the Solution Explorer, right-click on **CRM.svc** and then click **Set As Start Page**.



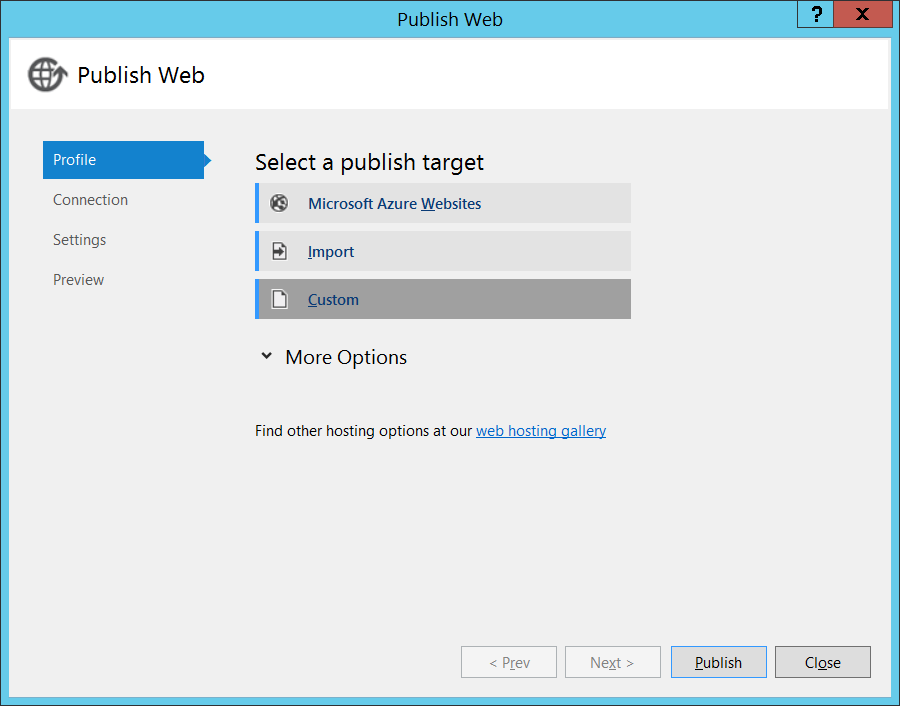
* 1. Begin a Visual Studio debugging session by pressing the **{F5}** key. This should launch the browser and perform an HTTP GET request on the web service entry point **CRM.svc**. This web service request should return an XML result which is displayed in the browser as shown in the following screenshot.



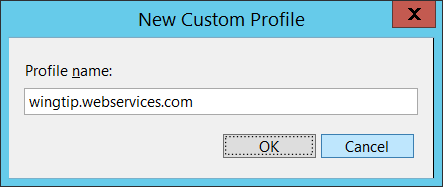
1. Close the browser.
2. Return to Visual Studio and make sure that the debugging session has stopped.
3. Deploy the new **CRM.svc** web service to the IIS website named **Wingtip Web Services**.
   1. In Solution Explorer, right-click on the **WingtipCRM** project and select the **Publish…** command.



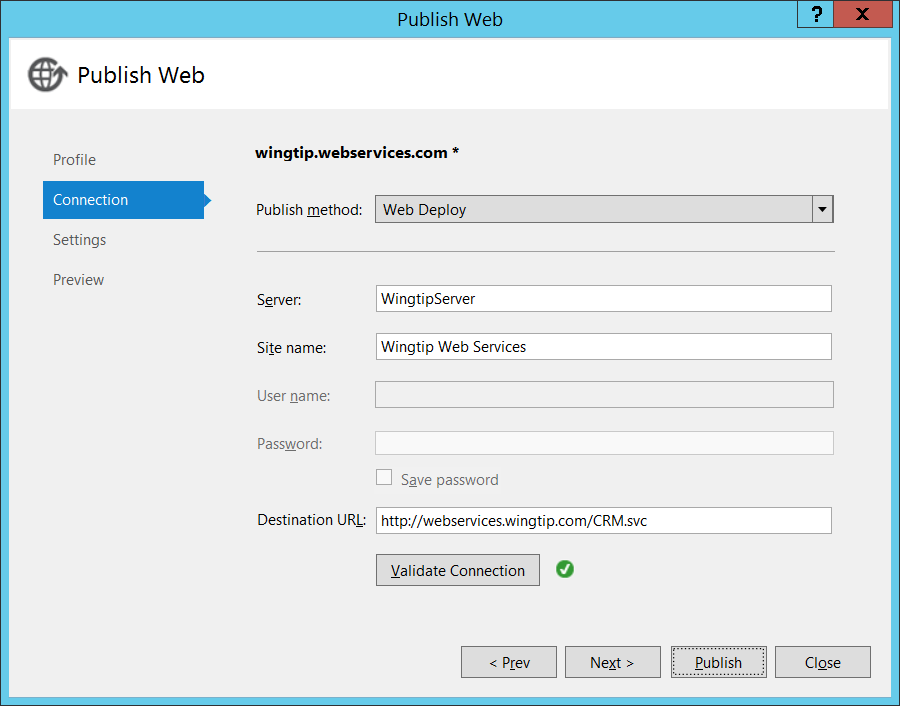
* 1. When the **Publish Web** dialog appears, there are three choices to pick from in the **Select a publish target** list. Click on Custom to create a new publish profile.



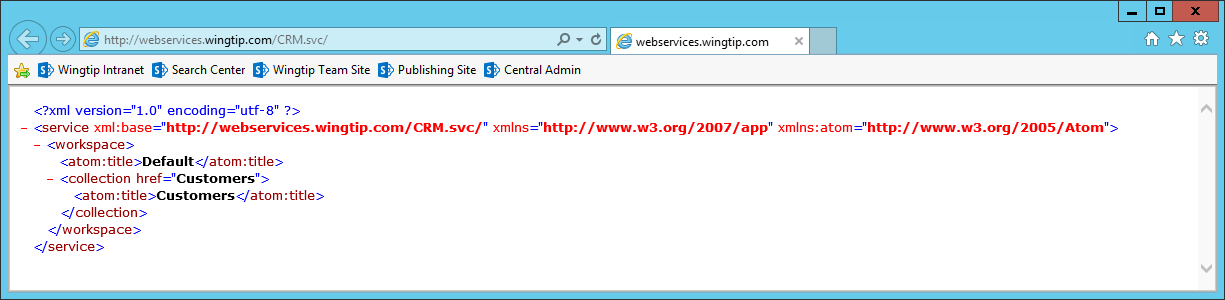
* 1. In the **New Custom Profile** dialog, enter a **Profile name** of **webservices.wingtip.com** and click **OK**.



* 1. Next, you should be prompted to enter connection information in the **Publish Web** dialog.
     1. Ensure the **Publish method** is set to **Web Deploy**.
     2. Enter a **Server** value of **WingtipServer**.
     3. Enter a **Site name** of **Wingtip Web Services**.
     4. Enter a **User name** of **WINGTIP\Administrator**.
     5. Enter a **Destination URL** or <http://webservices.wingtip.com/CRM.svc>.
     6. Click the **Validate Connection** button to ensure your information is entered correctly.



* + 1. Click **Publish** in the Publish Web dialog to deploy your **WingtipCRM** project files to the IIS website at <http://webservices.wingtip.com>. After Visual Studio has deployed the files from the project, it completes the publishing process by opening the browser and navigating to the Destination URL so you can verify that the project has been properly deployed.



In this exercise you created a custom web service to expose the data in a SQL Server database as an OData service. You also deployed the web service to run in the IIS website at <http://webservices.wingtip.com>. In the next exercise you will use this web service to get up to speed on creating the REST URLs required to execute queries against an OData data source.

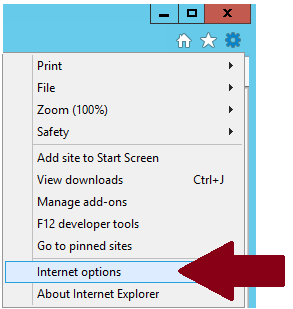
### Exercise 4: Execute OData Queries using the Browser

In this exercise you will get some practice structuring the REST URLs required to execute queries against an OData data source.

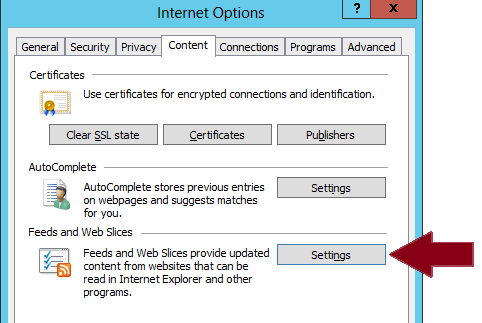
1. Configure the Internet Explorer to disable Feed Reading View.
   1. Navigate to the Internet Explorer. Launch Internet Explorer if it is not already running.
   2. Locate the Settings menu with the gear icon in the top right corner of the Internet Explorer window.



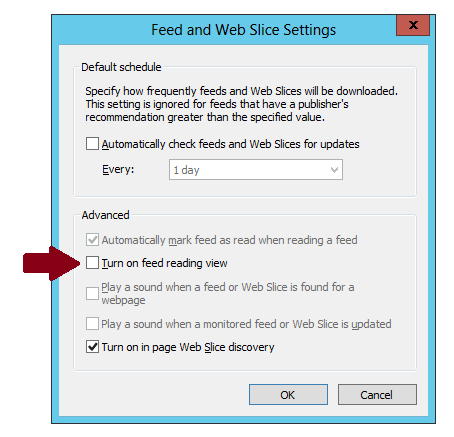
* 1. Drop down the Settings menu and select the menu command **Internet Options**.



* 1. In the **Internet Options** dialog, navigate to the **Content** tab and click the **Settings** button in the **Feeds and Web Slices** section.



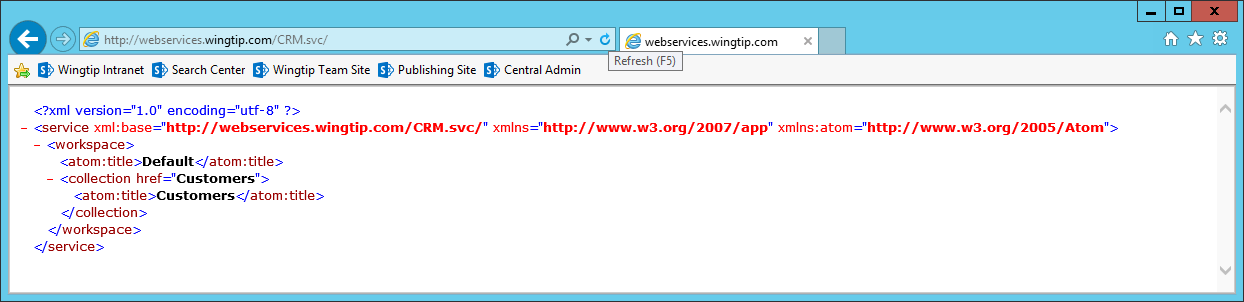
* 1. In the **Feeds and Web Slices** dialog, uncheck the **Turn on feed reading view** check box and then click **OK**.



* 1. Click **OK** to close the **Internet Options** dialog.
  2. Close Internet Explorer.
  3. Close any other running sessions of Internet Explorer.
  4. Restart Internet Explorer.

Why did you need to adjust this setting in the Internet Explorer? It has to do with viewing XML-based results. By default, Internet Explorer attempts to display certain types of XML results in a more human-readable format. However, for the purposes of this lab it’s best to disable this feature so you can see the raw XML-based results in the browser window.

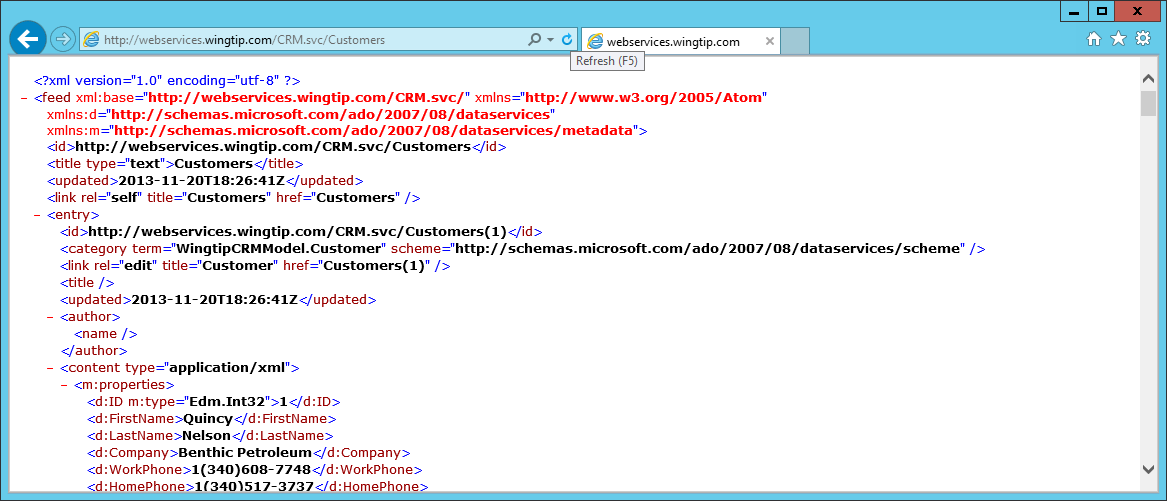
1. In Internet Explorer, navigate to the URL at <http://webservices.wingtip.com/CRM.svc>. You should see that an XML-based result has been returned from the web service which shows the collections of data it provides. You should be able to see that the XML result shows that there is a single collection named **Customers**.



1. Execute an OData query to retrieve the **Customers** table.
   1. In the address bar of the Internet Explorer to update the URL to **http://webservices.wingtip.com/CRM.svc/Customers**.



* 1. Press the **ENTER** key so that Internet Explorer issues a new HTTP GET request for the new URL. You should a large XML result which contains information about the customer records in the **Customers** table.

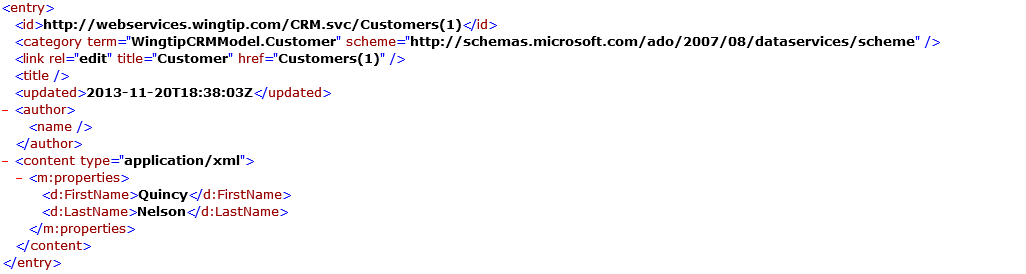


As you can see, it is pretty simple to execute an query against an OData data source using a browser. For the remainder of this exercise, you will create more and more complex queries to gain a better understanding of how to formulate the REST URLs required to execute queries with filtering and sorting.

1. Execute an OData query that selects a specific set of properties using the **$select** query string parameter
   1. Execute a query which to specify properties using the following URL.

http://webservices.wingtip.com/CRM.svc/Customers/?$select=FirstName,LastName

* 1. When you execute this query, you should see that it returns an XML result which includes only the requested properties.



1. Execute OData queries that filters results using the **$filter** query string parameter
   1. Execute a query for all customers with **FirstName** of **Luther**.

http://webservices.wingtip.com/CRM.svc/Customers/?$filter=FirstName eq 'Luther'

* 1. Verify that there is a single customer returned with a first name of Luther.
  2. Execute a query for all customers with **FirstName** starting with **B**.

http://webservices.wingtip.com/CRM.svc/Customers/?$filter=startswith(FirstName, 'B')

* 1. Verify that three customers are returned from the query which all have first names starting with B.

1. Execute an OData query that sorts results using the **$orderby** query string parameter.
   1. Execute a query which sorts by **LastName** and then **FirstName**.

http://webservices.wingtip.com/CRM.svc/Customers/?$orderby=LastName, FirstName

* 1. Verify that the query result is sorted by last name and then first name.

1. Execute an OData query that combines several query string parameter.
   1. Execute the following query which select properties and applies filtering and sorting.

../Customers/?$select=FirstName,LastName & $filter=startswith(FirstName, 'B')

* 1. Add a third query string parameter for sorting

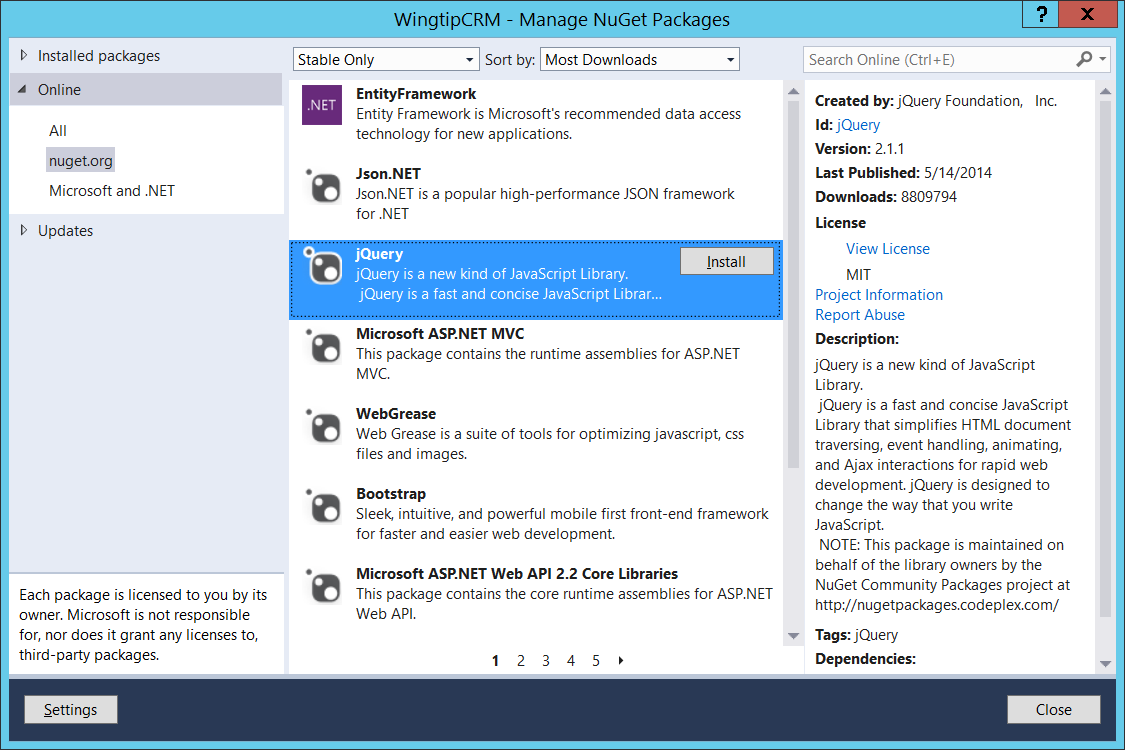
/?$select=FirstName,LastName & $filter=startswith(FirstName, 'B') & $orderby=LastName

* 1. Ensure that the query executed and returned the expected result.

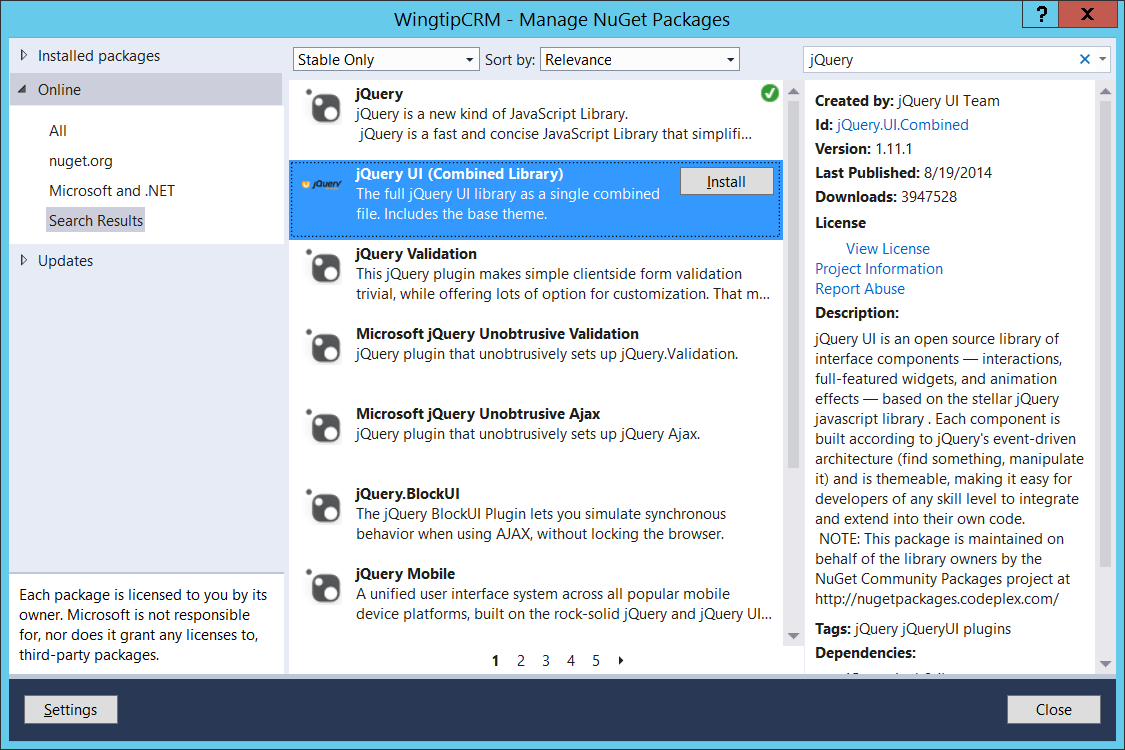
### Exercise 5: Write JavaScript Code to Consume an OData Data Source

In this exercise you will continue to work on the same ASP.NET application you created earlier in this lab. You will add a page and write client-side JavaScript code to create a user experience which allows the user to view, create, update and delete customers from the **WingtipCRM** database.

1. Return to Visual Studio 2013 and open the **WingtipCRM** project if it is not already open.
2. In Solution Explorer, right-click the **WingtipCRM** project node and then select the menu command **Manage NuGet Packages…**.  
   Select the nugget.org option from the Online category (on the left side of the screen)  
   Select the **jQuery** package and click **Install**.

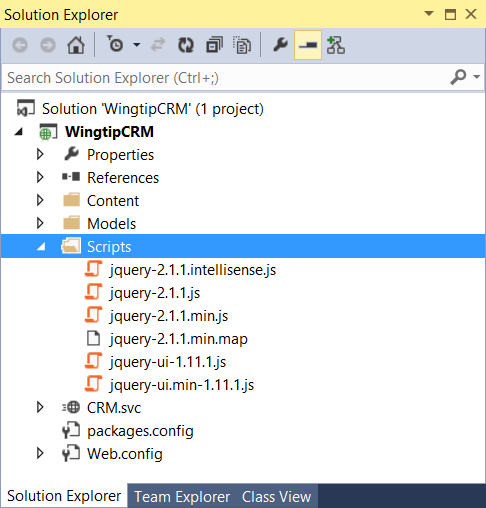


1. Select the **jQuery UI (Combined Library)** package and click **Install** and then click **Close.**



Note that the latest version number of the jQuery library and the jQuery UI library are constantly changing. Therefore, the version numbers you see in your project might be greater than the version numbers shown in these screenshots.

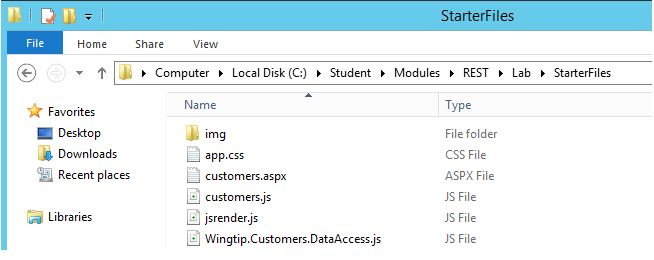
1. After you have added the NuGet packages for jQuery and jQuery UI, examine the changes to the project structure using the Solution Explorer. You should be able to see that the project now contains a **Scripts** folder with JavaScript source files for these libraries as well as a **Content** folder which includes CCS files and images for a jQuery UI theme.



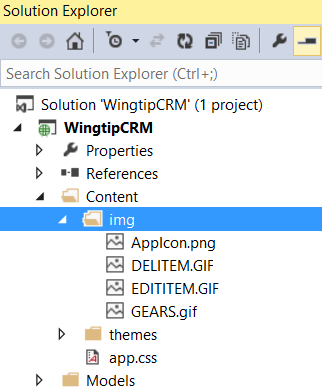
1. Now let’s add some starter files into the **WingtipCRM** project to build a page with a user interface to view and edit customers.
   1. Move over to the Windows Explorer and examine the files in the following folder.

C:\Student\Modules\REST\Lab\StarterFiles

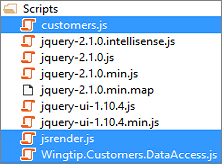
* 1. You should be able to see that there are several files in this folder along with a child folder named **img**.



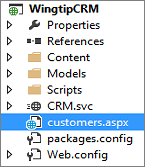
* 1. Add the **app.css** file from the **StarterFiles** folder into the **Contents** folder of the **WingtipCRM** project.  
     (Note: the easiest way to add files is to drag them from the source in Windows Explorer into the destination location in the Solution Explorer pane in Visual Studio)
  2. Add the **img** folder and the four image files into the **Contents** folder of the **WingtipCRM** project. After you have completed this step, your project should match the screenshot below.



* 1. There are three JavaScript files in the **StarterFiles** folder named **customer.js**, **jsrender.js** and **Wingtip.Customers.DataAccess.js**. Add all three of these files into the Scripts folder of the **WingtipCRM** project.



* 1. The last file to add is the ASP.NET page file name **customers.aspx**. Add this file to the root of the **WingtipCRM** project.



* 1. You have now added all the required start files.

1. Let’s configure the **customers.aspx** page as our start page and test this out.
   1. Within Visual Studio, double click customers.aspx to open the file in Code View.
   2. Examine the head section of the page. Verify that all the links which reference JavaScript files and CSS files are valid and are pointing to files that exist within your project. Fix any links that need to be adjusted. Pay particular attention to the links to jQuery and jQuery UI as you will likely need to update the version number in these.

<head runat="server">

<%@ Page %>

<!DOCTYPE html>

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

<head runat="server">

<meta charset="utf-8" />

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

<title>Wingtip CRM</title>

<!-- links to jQuery library and jQuery UI library -->

<!-- update these links if required -->

<script src="Scripts/jquery-2.1.1.js"></script>

<script src="Scripts/jquery-ui-1.11.1.js"></script>

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

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

<script src="Scripts/Wingtip.Customers.DataAccess.js"></script>

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

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

</head>

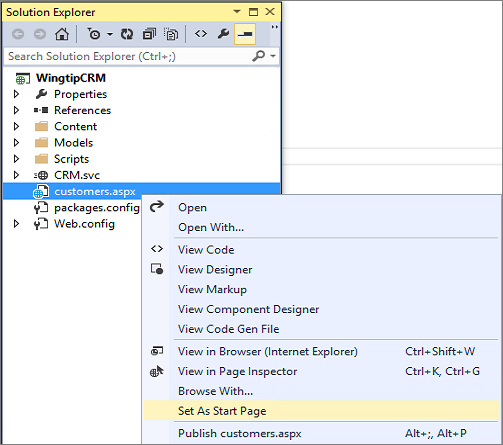
<body>

</body>

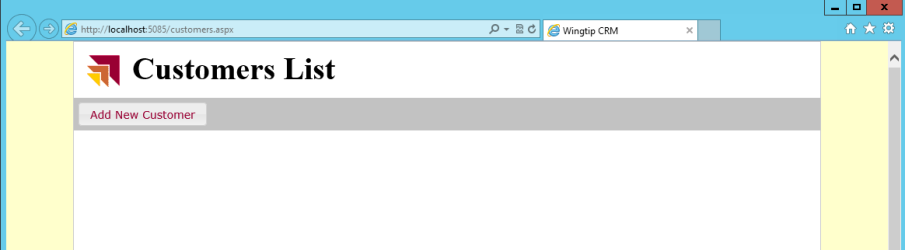
</html>

<body>

* 1. Right-click on customers.aspx and select the menu command **Set As Start Page**.



* 1. Press the **{F5}** key to begin a debugging session. Visual Studio should launch the ASP.NET application and display the **customers.aspx** page. At this point there should be a simple user interface as seen in the screenshot below. However, the application will require that you begin to write client-side JavaScript code to add the functionality to view and modify customers.



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

1. Open the file named **Wingtip.Customers.DataAccess.js**. You should be able to see that there is already some JavaScript code inside that defines a JavaScript module named **Wingtip.Customers.DataAccess**. However, the functions within this module do not contain any code yet. Your job is to add the required JavaScript code to this file and to **customers.js** to implement CRUD functionality for the application.

Over the next few steps you will be asked to write a good deal of JavaScript code to bring this application to life. Note that there are code snippets available if you would rather copy and paste all this code instead of typing it in. You can find these snippets in simple text files for each method in the folder at **C:\Student\Modules\REST\Lab\Snippets**.  
Note: the Snippet files associated with Wintip.Customers.DataAccess.js all start with **DataAccess** while the snippet files associated with the customers.js file all start with **View**.

1. In **Wingtip.Customers.DataAccess.js**, modify the **getCustomers** function with the following code and then save your work.

var getCustomers = function () {

// parse target URI for customer list in app web

var requestUri = "/crm.svc/Customers/" +

"?$select=ID,FirstName,LastName,Company,WorkPhone,HomePhone,EmailAddress" +

"&$orderby=LastName,FirstName";

// create object for request headers

var requestHeaders = {

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

}

// send call across network

var deferred = $.ajax({

url: requestUri,

headers: requestHeaders,

});

return deferred.promise();

};

1. In **Wingtip.Customers.DataAccess.js**, modify the **getCustomer** function with the following code and then save your work.

var getCustomer = function (Id) {

// begin work to call across network

var requestUri = "/crm.svc/Customers(" + Id + ")";

// execute AJAX request

var requestHeaders = {

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

}

var deferred = $.ajax({

url: requestUri,

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

headers: requestHeaders,

});

return deferred.promise();

}

1. In **customers.js**, modify the **getCustomers** function with the following code and then save your work.

function getCustomers() {

// clear results and add spinning gears icon

$("#content\_box").empty();

$("<img>", { "src": "Content/img/GEARS.gif" }).appendTo("#content\_box");

// call data access function which returns promise

var promise = Wingtip.Customers.DataAccess.getCustomers()

// use promise to implement what happens when OData result is ready

promise.then(onGetCustomersComplete, onError);

}

1. In **customers.js**, modify the **onGetCustomersComplete** function with the following code and then save your work. Remember that you can copy and paste this code from **C:\Student\Modules\REST\Lab\Snippets\View.onGetCustomersComplete.txt**.

function onGetCustomersComplete(data) {

$("#content\_box").empty();

var odataResults = data.d;

// set rendering template

var tableHeader = "<thead>" +

"<td>&nbsp;</td>" +

"<td>&nbsp;</td>" +

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

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

"<td>Company</td>" +

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

"<td>Home Phone</td>" +

"<td>Email</td>" +

"</thead>";

var table = $("<table>", { ID: "customersTable" }).append($(tableHeader));

// create rendering template for table row using jsRender syntax

var rowTemplateString = "<tr>" +

"<td><a href='javascript: onUpdateCustomerRequest({{>ID}});'>" +

"<img src='Content/img/EDITITEM.gif' alt='Edit' /></a></td>" +

"<td><a href='javascript: onDeleteCustomer({{>ID}});'>" +

"<img src='Content/img/DELITEM.gif' alt='Delete' /></a></td>" +

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

"<td>{{>LastName}}</td>" +

"<td>{{>Company}}</td>" +

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

"<td>{{>HomePhone}}</td>" +

"<td>{{>EmailAddress}}</td>" +

"</tr>";

// load row template and give it a name of "rowTemplate"

$.templates({ "rowTemplate": rowTemplateString });

var renderedTable = $.render.rowTemplate(odataResults);

// render table rows using row template and OData data result

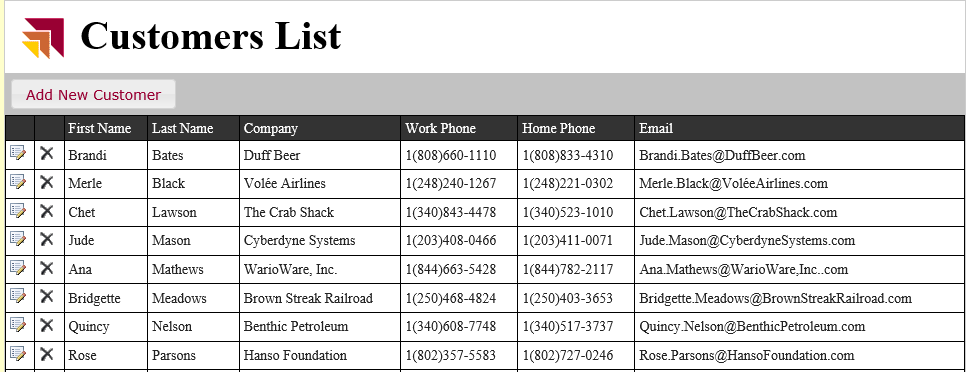
table.append(renderedTable);

// append table to div in DOM

$("#content\_box").append(table);

}

1. At this point, you should have implemented enough functionality to display a list of customers. Press the **{F5}** key to begin a debugging session. You should be able to see that the application now displays a table of customer information when the **customers.aspx** page has initialized.



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

Now, you have implemented the read functionality, it's time to move on and implement the functionality to add new customers.

1. In **Wingtip.Customers.DataAccess.js**, modify the **addCustomer** function with the following code and then save your work.

var addCustomer = function (FirstName, LastName, Company, WorkPhone, HomePhone, Email) {

var requestUri = "/crm.svc/Customers/";

var requestHeaders = {

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

}

var customerData = {

LastName: LastName,

FirstName: FirstName,

Company: Company,

WorkPhone: WorkPhone,

HomePhone: HomePhone,

EmailAddress: Email

};

var requestBody = JSON.stringify(customerData);

var deferred = $.ajax({

url: requestUri,

type: "POST",

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

headers: requestHeaders,

data: requestBody,

});

return deferred.promise();

};

1. In **customers.js**, modify the **onAddCustomerRequest** function with the following code and then save your work.

function onAddCustomerRequest(event) {

$("#firstName").val("");

$("#lastName").val("");

$("#company").val("");

$("#workPhone").val("");

$("#homePhone").val("");

$("#email").val("");

var customer\_dialog = $("#customer\_dialog");

customer\_dialog.dialog({

autoOpen: true,

title: "Add Customer",

width: 640,

buttons: {

"Add": function () {

onAddCustomer();

$(this).dialog("close");

},

"Cancel": function () { $(this).dialog("close"); },

}

});

}

1. In c**ustomers.js**, modify the **onaddCustomer** function with the following code and then save your work.

function onAddCustomer() {

// get input data from add customer dialog

var LastName = $("#lastName").val();

var FirstName = $("#firstName").val();

var Company = $("#company").val();

var WorkPhone = $("#workPhone").val();

var HomePhone = $("#homePhone").val();

var Email = $("#email").val();

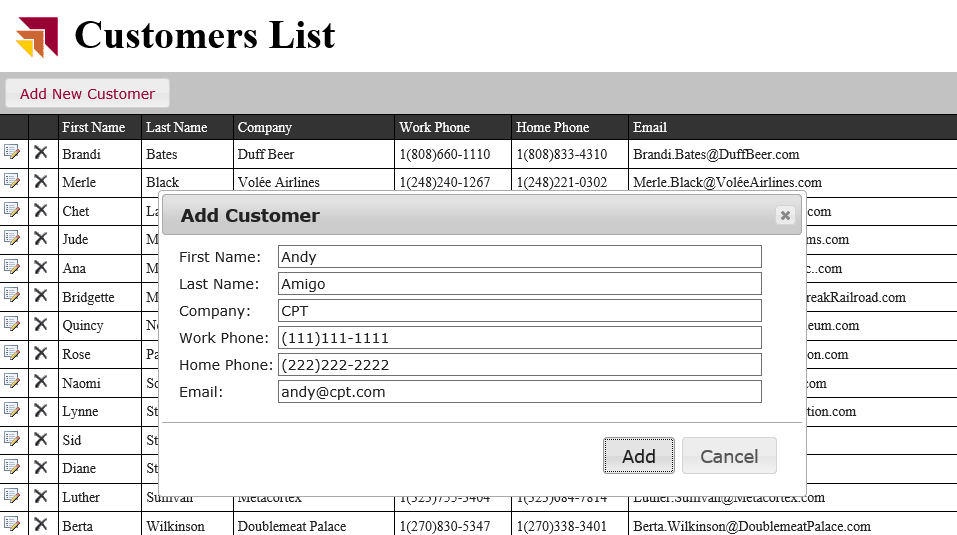
// add new customer

var promise = Wingtip.Customers.DataAccess.addCustomer(FirstName, LastName, Company, WorkPhone, HomePhone, Email);

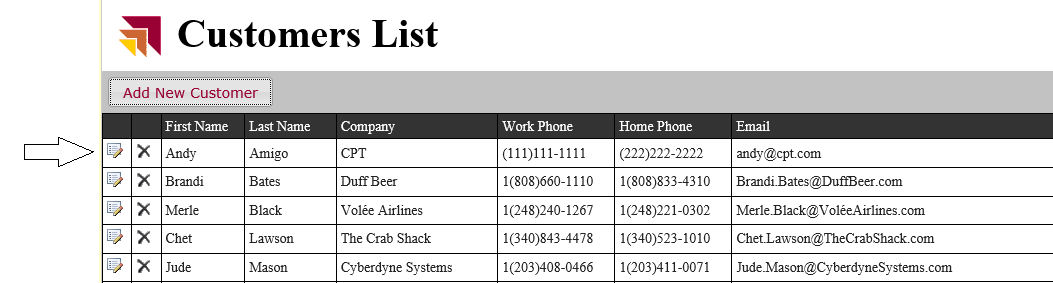
promise.then(onSuccess, onError);

}

1. At this point, you should have implemented enough functionality to add a new customer. Press the **{F5}** key to begin a debugging session. You should be able to see the table of customers on the **customers.aspx** page.
2. Click the **Add New Customer** button. When you do this you should be prompted with the **Add Customer** dialog. Fill out the input controls in the dialog with sample data as shown in the following screenshot and click to **Add** button to test your work.



1. You should be able to verify that you can successfully add new customers.



Now, it is time to implement delete functionality.

1. In **Wingtip.Customers.DataAccess.js**, modify the **deleteCustomer** function with the following code and then save your work.

var deleteCustomer = function (Id) {

var requestUri = "/crm.svc/Customers(" + Id + ")";

var requestHeaders = {

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

}

var deferred = $.ajax({

url: requestUri,

type: "DELETE",

headers: requestHeaders,

success: onSuccess,

error: onError

});

return deferred.promise();

};

1. In **customers.js**, modify the **onDeleteCustomer** function with the following code and then save your work.

function onDeleteCustomer(customerId) {

var promise = Wingtip.Customers.DataAccess.deleteCustomer(customerId);

promise.then(onSuccess, onError);

}

1. At this point, you should have implemented enough functionality to delete existing customers. Press the **{F5}** key to begin a debugging session. You should verify that you can delete customers from the **customers.aspx** page by clicking the delete icon with the X on it. (Try doing this on the record you created earlier (Andy Amigo)).

If you went “hog wild” with testing the delete function, you might want to refresh the customer data in the WingtipCRM database to return it to its original state. Note that there is a SQL script in the **C:\Student\Setup** folder named **RepopulateWingtipCrmCustomers.sql**. You can execute this script using SQL Server Management Studio to get back to the original set of customer records. You can follow the general patter laid out in Exercise 1 of this lab using the **RepopulateWingtipCRMCustomers.sql** to accomplish this.

1. In **Wingtip.Customers.DataAccess.js**, modify the **updateCustomer** function with the following code and save your work.

var updateCustomer = function (Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email) {

var requestUri = "/crm.svc/Customers(" + Id + ")";

var requestHeaders = {

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

"X-HTTP-Method": "MERGE",

}

var customerData = {

LastName: LastName,

FirstName: FirstName,

Company: Company,

WorkPhone: WorkPhone,

HomePhone: HomePhone,

EmailAddress: Email

};

var requestBody = JSON.stringify(customerData);

var deferred = $.ajax({

url: requestUri,

type: "POST",

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

headers: requestHeaders,

data: requestBody,

});

return deferred.promise();

};

1. In **customers.js**, modify the **onUpdateCustomerRequest** function with the following code and then save your work.

function onUpdateCustomerRequest(customerId) {

var promise = Wingtip.Customers.DataAccess.getCustomer(customerId);

promise.then(onUpdateCustomerDialog, onError);

}

1. In **customers.js**, modify the **onUpdateCustomerDialog** function with the following code and then save your work.

function onUpdateCustomerDialog(data) {

// update customer dialog with current customer data

$("#firstName").val(data.d.FirstName);

$("#lastName").val(data.d.LastName);

$("#company").val(data.d.Company);

$("#workPhone").val(data.d.WorkPhone);

$("#homePhone").val(data.d.HomePhone);

$("#email").val(data.d.EmailAddress);

// store value for item id

$("#customer\_id").val(data.d.ID);

var customer\_dialog = $("#customer\_dialog");

customer\_dialog.dialog({

autoOpen: true,

title: "Edit Customer",

width: 640,

buttons: {

"Update": function () {

onUpdateCustomer();

$(this).dialog("close");

},

"Cancel": function () {

$(this).dialog("close");

},

}

});

}

1. In **customers.js**, modify the **onUpdateCustomer** function with the following code and then save your work.

function onUpdateCustomer() {

// scrape input values from dialog

var Id = $("#customer\_id").val();

var FirstName = $("#firstName").val();

var LastName = $("#lastName").val();

var Company = $("#company").val();

var WorkPhone = $("#workPhone").val();

var HomePhone = $("#homePhone").val();

var Email = $("#email").val();

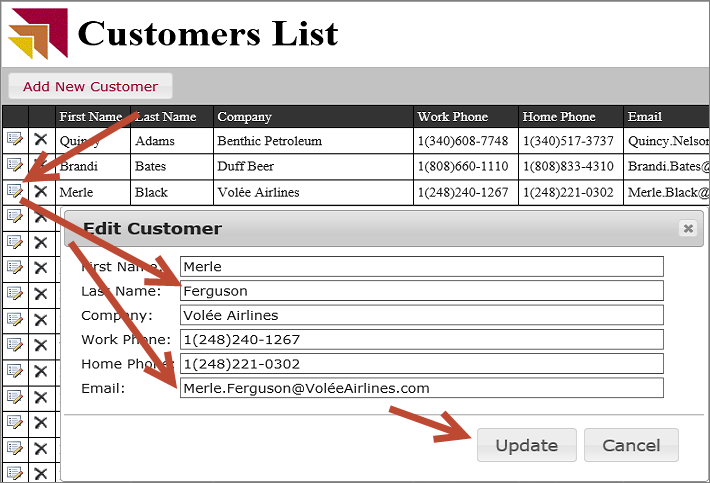
// update customer

var promise = Wingtip.Customers.DataAccess.updateCustomer(Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email);

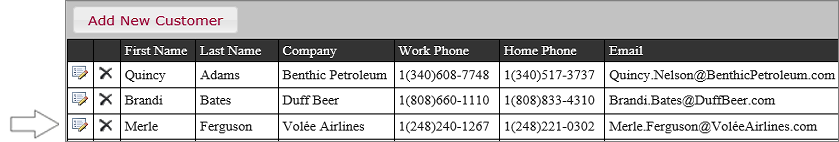
promise.then(onSuccess, onError);

}

1. At this point, you should have implemented enough functionality to update existing customers. Press the **{F5}** key to begin a debugging session. You should verify that you can update customers.
   1. Click on the edit icon in the row of an existing customer to display the **Edit Customer** dialog which should be populate with the current information for the customer. Modify the values of one of more input controls and click **Update** to save your work.



* 1. Verify that your changes were saved by examining that customer in the customer list.



At this point, you have now implemented the full range of CRUD functionality in a web application by writing client-side JavaScript code which consumes an OData data source.