Programming a Provider-hosted App using the CSOM

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\09_CSOM\Lab

Lab Overview: In this lab you will program create a provider-hosted app that programs against the host web using CSOM. Through the various exercises in this lab you will learn how to use CSOM to query site properties, to query the set of existing lists and to create a new list.

Exercise 1: Make Sure the S2S Configuration on Your VM is Setup Correctly

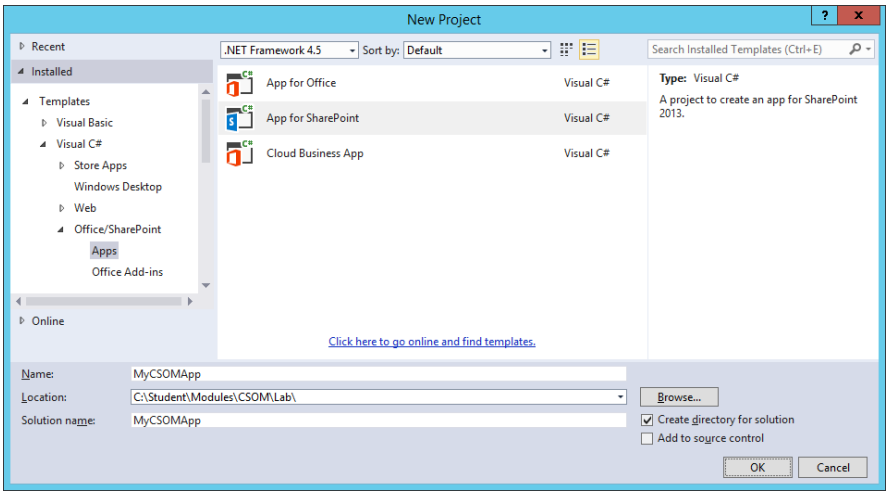
This lab exercise assumes that you have completed the previous lab titled **Configuring App Permissions and S2S Authentication**. More specifically, you must have completed Exercise 3 titled **Create a Test Certificate for Working with S2S Trusts Authentication** where you created a self-signed SSL certificate and then you created a Trusted Security Token Issuer based on that certificate to support S2S authentication.

1. Determine whether you have already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** in the earlier lab titled **Lab06: Configuring App Permissions and S2S Authentication**.
   a) If you are not sure whether you have completed this prerequisite exercise, you can examine the **C:\Certs** folder on your student VM to see if there are two certificate files named **WingtipAppCetificate01.cer** and **WingtipAppCetificate01.pfx**. If these files do not exist in the **C:\Certs** folder, you have not yet completed Exercise 3.
   b) If you have already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** from lab 6, you can now move ahead to Exercise 2 in this lab.
   c) If you have *NOT* already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** from lab 6 titled **Configuring App Permissions and S2S Authentication**, you must go back and complete that exercise now. After that you can then proceed with Exercise 2 in this lab.

Exercise 2: Create a Provider-hosted App that Uses CSOM

In this exercise you will create a new SharePoint Provider- Hosted app that will leverage the certificate and trusted security token issuer to implement a high -trust app that utilizes S2S authentication.

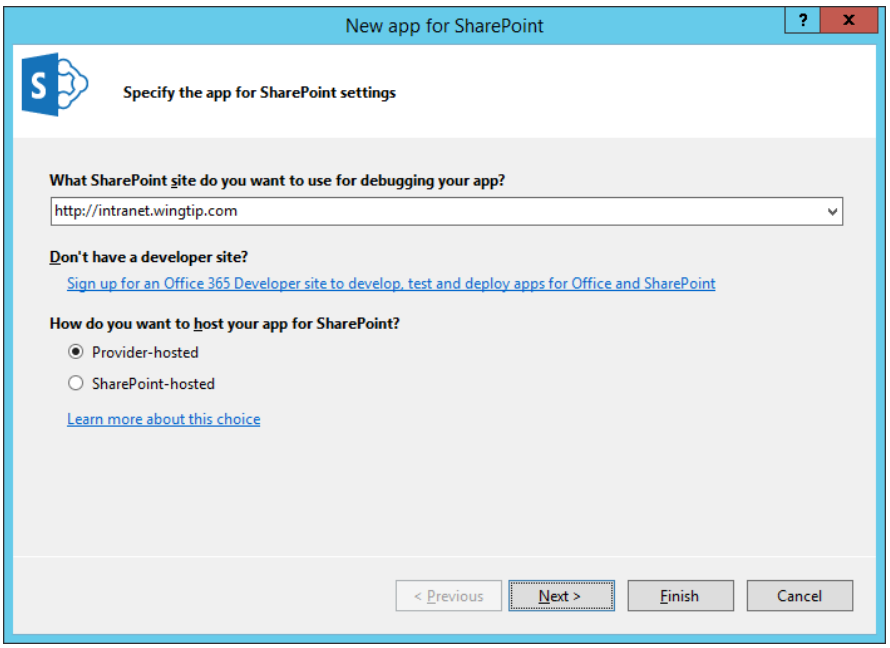
1. Launch Visual Studio 2015 as administrator (if it is not already open).
   a) Press the **Windows** key to display the Windows Start page.
   b) Right click on the **Visual Studio 2015** tile and select **Run as administrator**.
2. Create a new project in Visual Studio 2015 for hosting your app:
   a) Select **File > New > Project**.
   b) In the **New Project** dialog, find the **SharePointAdd-in** template under the **Templates >Visual C#>Office/SharePoint** section.
      i) Name: **MyCSOMApp**.
      ii) Location: **C:\Student\Modules\09_CSOM\Lab**
      iii) When the **New Project** dialog looks like the following below, click the **OK** button.



c) In the **New app for SharePoint** wizard, use the following values to complete the wizard and click **Next**.

i) What site do you want to use for debugging? https://intranet.wingtip.com

ii) How do you want to host your app for SharePoint? **Provider- hosted**.



d) On the **Specify the web project type** screen select the **ASP.NET Web Forms Application** choice and click the **Next** button to move to the next screen. Do not click the **Finish** button until you have gone through all the pages of the wizard.

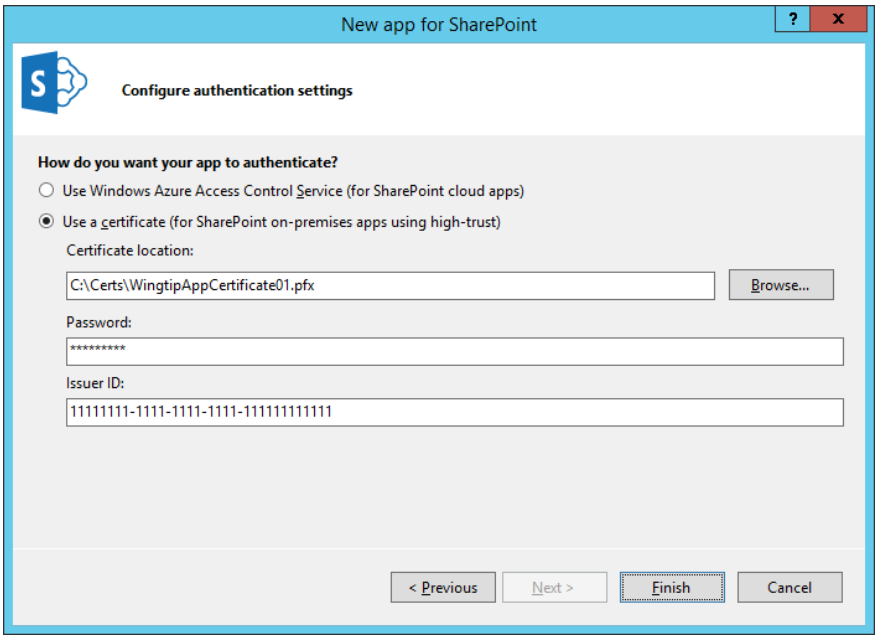
e) You should now be at the **Configure authentication settings** page. Fill this page out as follows.

i) Select **Use a certificate (for SharePoint on -premise apps using high -trust)**

ii) Certificate location : **C:\Certs\WingtipAppCertificate01.pfx**

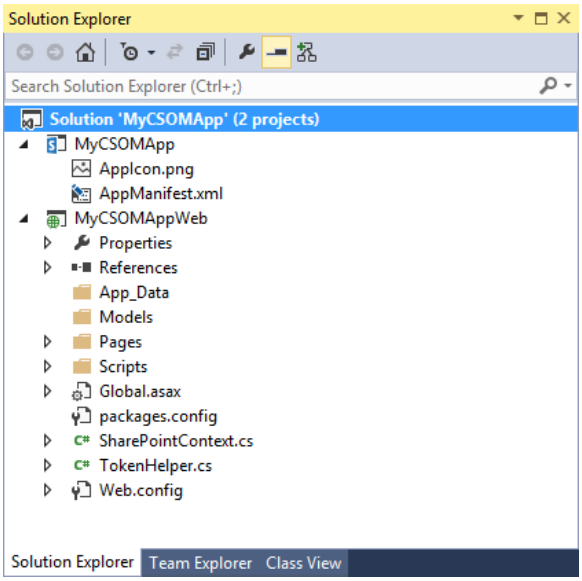
iii) Password : **Password1**

iv) Issuer ID : **11111111 - 1111 - 1111 - 1111 – 111111111111**



f) Click **Finish** to complete the wizard and create the new provider-hosted app project.

g) Once the Visual Studio solution has been created, you should see it contains two projects named **MyCsomApp** and **MyCsomAppWeb**.



Make some modification to the App project named MyCsomApp

3. Replace the AppIcon.png file with a custom image.

a) In the Solution Explorer, look inside the **MyCsomApp** project and locate the image file **AppIcon.png**.

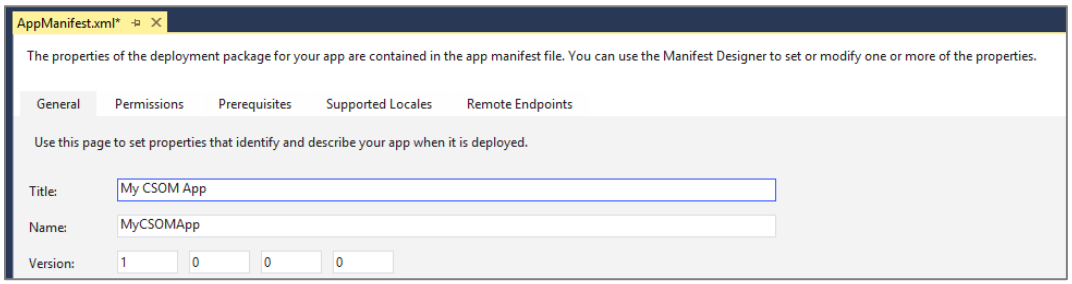
b) Using Windows Explorer, look inside the **StarterFiles** folder for this lab at the following path.

```
C:\Student\Modules\CSOM\Lab\StarterFiles
```

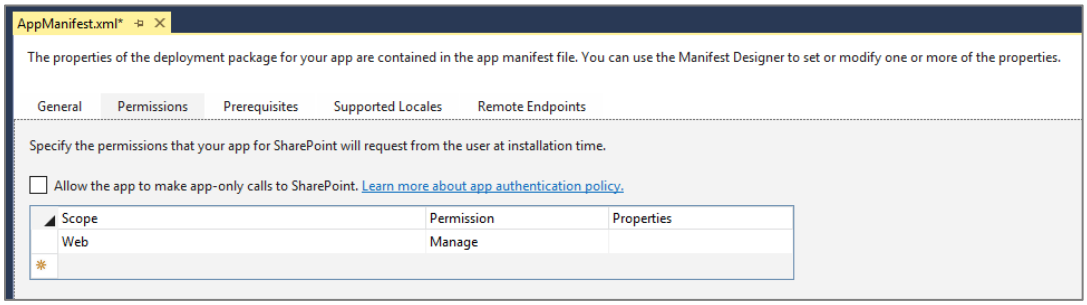
c) Locate the custom image file named **AppIcon.png**.

d) Use the **AppIcon.png** file in the **StarterFiles** folder to replace the **AppIcon.png** file in the **MyCsomApp** project.

4. Modify the **AppManifest.xml** file.
   a) In Solution Explorer, double-click on **AppManifest.xml** open it in Visual Studio's App Manifest Designer.
   b) Update the **Title** property from **MyCsomApp** to **My CSOM App**.

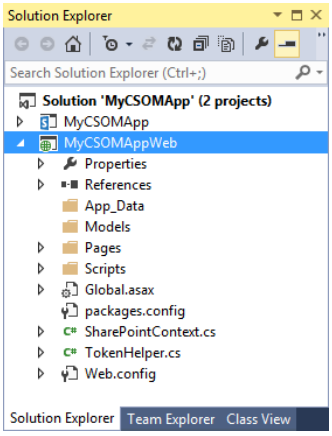


   c) Click on the **Permissions** tab and then add a permission with a **Scope** of **Web** and a **Permission** level of **Manage**. This permissions is required so that your app will be able to read site properties as well as query and create lists.

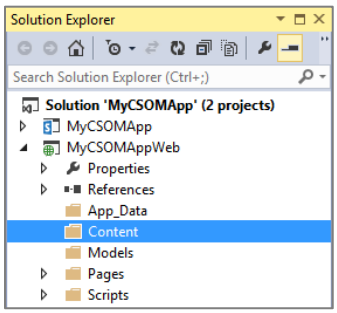


   d) Save your changes and close **AppManifest.xml**.

5. In Solution Explorer, move down to the Web Project named **MyCsomAppWeb**.



6. Add a CSS file and an image file for the start page.
   a) Inside the **MyCsomAppWeb** project, add a new top-level folder named **Content**.



b) Using Windows Explorer, locate the CCS file in the **StarterFiles** folder at the following path.

```
C:\Student\Modules\09_CSOM\Lab\StarterFiles\Content\App.css
```

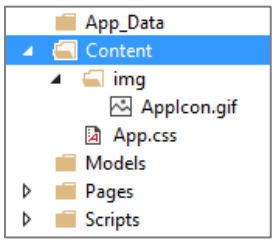
c) Add the App.css file from the **StarterFiles** folder into the new **Content** folder in the **MyCsomApp** project.

d) Create a child folder named **img** inside the **Content** folder.

e) Using Windows Explorer, locate the GIF file in the **StarterFiles** folder at the following path.

```
C: \Student\Modules\09_CSOM\Lab\StarterFiles\Content\img\AppIcon.gif
```

f) Add the AppIcon.gif file from the **StarterFiles** folder into the new **img** folder in the **MyCsomApp** project.



g) Open the CSS file named **App.css** and have a quick look at the set of CSS rules inside. Note there is no need for you to modify the CSS rules in this lab. You are just looking.

h) When you are done, close the App.css file without saving any changes.

7. Modify the app's start page named **Default.aspx**.

a) Using the Solution Explorer, look inside the **Pages** folder of the **MyCsomAppWeb** project and locate **Default.aspx**.

b) Double click on **Default.aspx** to open it in the Web Forms Editor of Visual Studio.

c) Delete all contents from **Default.aspx** except for the first line with the **<@Page>** directive.

d) Using Windows Explorer, locate the file named **Default.aspx.txt** in the **StarterFiles** folder at the following path.

```
C: \Student\Modules\09_CSOM\Lab\StarterFiles\Default.aspx.txt
```

e) Open the file named **Default.aspx.txt** in Notepad.exe and copy its entire contents to the Windows clipboard.

f) Return to Visual Studio.

g) Navigate to **Default.aspx** which should still be open in the Web Forms Editor.

h) Position you cursor in **Default.aspx** right below the first line with the **<@Page>** directive and paste the content of the Windows clipboard.

i) Save your changes to **Default.aspx**.

8. Examine the layout of the HTML you just pasted into **Default.aspx**.

a) Make sure you are looking at **Default.aspx** in Code View.

b) Note in the head section that that there is already a link to the CSS file named **App.css**.

c) The body section contains a div with an id of **page_width** which contains child elements that make up the user interface.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" ... >
<!DOCTYPE html>
<html>
<head runat="server">

<meta charset="utf - 8" />

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

<title>My CSOM App</title>

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

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

<div id="page_width">

<! -- more inside -- >

</div>

</form> </body> </html>
```

9. Examine the server-side controls inside the div with the id of **page_width**.
   a) First, you should see there is an ASP.NET **HyperLink** control with an ID of **HostWebLink**.

```
<div id="nav_bar">

<asp:HyperLink ID="HostWebLink" runat="server" /> </div>
```

   b) Next, you should see two ASP.NET Button controls with ID values of **cmdGetTitleCSOM** and **cmdGetTitleREST**.

```
<nav id="toolbar">

<asp:Button ID="cmdGetSiteProperties" runat="server" Text="Get Properties" />

<asp:Button ID="cmdGetLists" runat="server" Text="Get Lists" />

<asp:Button ID="cmdCreateCustomersList" runat="server" Text="Create List" /> </nav>
```

   c) Finally, there is an ASP.NET Literal control with an ID of **placeholderMainContent**.

```
<div id="content_box">

<asp:Literal ID="placeholderMainContent" runat="server" ></asp:Literal> </div>
```

10. Modify the server-side C# code behind the start page.
    a) In Solution Explorer, locate the code-behind file for **Default.aspx** named **Default.aspx.cs**.
    b) Open **Default.aspx.cs** and inspect the code inside that was generate by Visual Studio. You should be able to see that there is a code- behind class named **Default** for the page named **Default.aspx**. This code-behind class named **Default** contains two methods named **Page_PreInit** and **Page_Load**.

```
public partial class Default : System.Web.UI.Page {

  protected void Page_PreInit(object sender, EventArgs e) {
    // implementation details
  }

  protected void Page_Load(object sender, EventArgs e) {
    // implementation details
  }

}
```

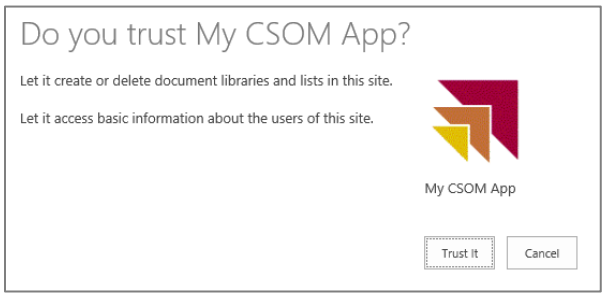
    c) Delete the entire method named **Page_PreInit**.
    d) Leave the **Page_Load** method but delete all to code inside it so that it is an empty method.

```
public partial class Default : System.Web.UI .Page {

  protected void Page_Load(object sender, EventArgs e) {

  }
}
```

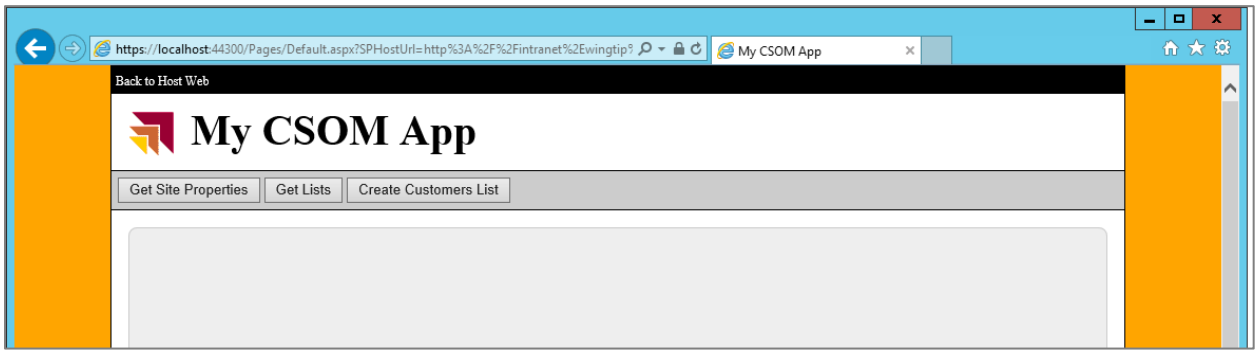
e) Implement the **Page_Load** method using the following code.

```
protected void Page_Load(object sender, EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);
  this.HostWebLink.NavigateUrl = spContext.SPHostUrl.AbsoluteUri;
  this.HostWebLink.Text = "Back to Host Web";

}
```

11. Test out your work by running the app in the Visual Studio Debugger.
    a) Press the **{F5}** key in Visual Studio to begin a debugging session for the app project.
    b) As Visual Studio installs the app in the test site, you will be prompted whether you trust the app. Click **Trust It**.



    c) The start page of the app should appear as the page in the following screenshot.



    d) Test it out the **Back to Host Web** link to make sure the it works.
    e) Close the browser and quit the debugging session.
12. Add event handlers behind the buttons
    a) Return to Visual Studio and the Web Project named **MyCsomAppWeb**.
    b) Open the page named **Default.aspx** in the Web Forms Editor.
    c) Switch **Default.aspx** into Design View.
    d) When in Design View, double-click on the **Get Site Properties** button to generate an event handler in **Default.aspx.cs**.
    e) Return to Design View in **Default.aspx** and double-click on the **Get Lists** button to generate another event handler.
    f) Return to Design View in **Default.aspx** and double-click on the **Create List** button to generate another event handler.
    g) When you're done, there should be three new event handlers named **cmdGetSiteProperties_Click**, **cmdGetLists_Click** and **cmdCreateCustomersList_Click**.

```
public partial class Default : System.Web.UI.Page {

  protected void Page_Load(object sender, EventArgs e) {
    // ...
  }

  protected void cmdGetSiteProperties_Click(object sender, EventArgs e) {

  }

  protected void cmdGetLists_Click(object sender, EventArgs e) {

  }

  protected void cmdCreateCustomersList_Click(object sender, EventArgs e) {

  }
}
```

13. Add the following code to **cmdGetSiteProperties_Click** to execute a CSOM call using S2S authentication.

```
protected void cmdGetSiteProperties_Click(object sender, EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

  using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
    clientContext.Load(clientContext.Web);
    clientContext.ExecuteQuery();
    placeholderMainContent.Text = "Host web title (CSOM): " + clientContext.Web.Title;
  }

}
```

14. Test your work to ensure the CSOM call executes successful without any errors.
    a) Press the **{F5}** key to start another debugging session.
    b) If you are prompted to trust the app, click the **Trust It** button to complete the app installation.
    c) After the app has been installed, the Visual Studio debugger should redirect you to the app's start page in the remote web.
    d) Click the **Get Properties** button. When you click this button, the page should display a message with the title of the host web as shown in the following screenshot.

    e) Close the browser window to stop the debugging and return to Visual Studio.

15. Modify the code in **cmdGetSiteProperties_Click** to display the **Url** property and the **Id** property of the host web in addition to the **Title** as shown in the following code.

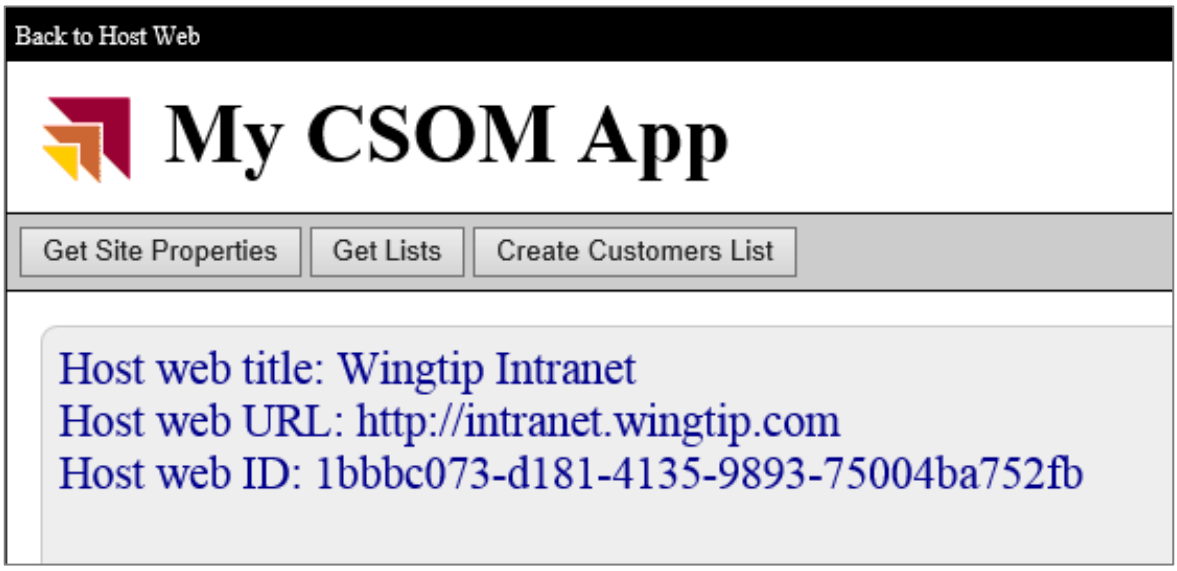
```
using (var clientContext = spContext.CreateUserClientContextForSPHost()) {

  clientContext.Load(clientContext.Web);
  clientContext.ExecuteQuery();
```

```
  placeholderMainContent.Text = "<div>Host web title: " + clientContext.Web.Title + "</div>" +
                                "<div>Host web URL:   " + clientContext.Web.Url + "</div>" +
                                "<div>Host web ID:    " + clientContext.Web.Id + "</div>";
}
```

16. Improve the code in **cmdGetSiteProperties_Click** to optimize the retrieval site properties by modifying the call to **ClientContent.Load** as shown in the following code.

```
using (var clientContext = spContext.CreateUserClientContextForSPHost()) {

  clientContext.Load(clientContext.Web, web => web.Title, web => web.Url, web => web.Id);
  clientContext.ExecuteQuery();

  placeholderMainContent.Text = "<div>Host web title: " + clientContext.Web.Title + "</div>" +
                                "<div>Host web URL:   " + clientContext.Web.Url + "</div>" +
                                "<div>Host web ID:    " + clientContext.Web.Id + "</div>";
}
```

17. Test the app in the debugger and make sure you can see all three site properties when clicking the **Get Site Properties** button.



18. When you have finished testing, close the browser window to stop the debugging and return to Visual Studio.

In this exercise you created a new Provider-Hosted App which can make CSOM calls to the host web using app authentication based on an S2S trust.

Exercise 3: Program CSOM to Query the Set of Lists in the Host Web

In this exercise you will write the CSOM code required to query the host web to determine when lists exist.

1. Open the source file **default.aspx.cs** if it is not already open.
2. Add the following **using** statement at the top of the file directly following the **using** statements that are already there.

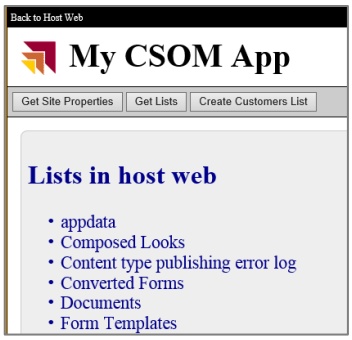
```
using Microsoft.SharePoint.Client;
```

3. Modify the code in **cmdGetLists_Click** to retrieve the set of lists on the host web and to display them on the app's starts page.

```
protected void cmdGetLists_Click(object sender, EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

  using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
```

```
    clientContext.Load(clientContext.Web);
    clientContext.Load(clientContext.Web.Lists);
    clientContext.ExecuteQuery();
    string html = "<h2>Lists in host web</h2>";

    html += "<ul>";
    foreach (var list in clientContext.Web.Lists) {
      html += "<li>" + list.Title + "</li>";
    }
    html += "</ul>";

    placeholderMainContent.Text = html;
  }

}
```

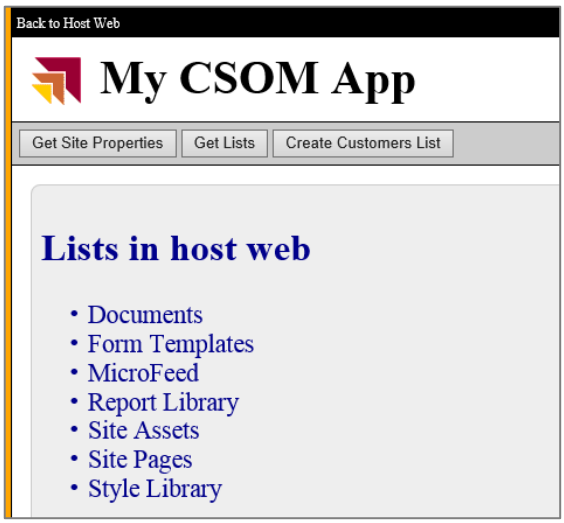
4. Test your code by pressing **{F5}** and starting a debugging session. You should be able to see that your query has returned many lists including several hidden lists such as **appdata**, **Composed Looks** and **Converted Forms**.



5. Close the browser window to stop the debugging and return to Visual Studio.
6. Improve the code in **cmdGetLists_Click** to optimize the retrieval site properties by modifying the call to **ClientContent.Load** as shown in the following code.

```
protected void cmdGetLists_Click(object sender , EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

  using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
    clientContext.Load(clientContext.Web);
    ListCollection Lists = clientContext.Web.Lists;
    clientContext.Load(Lists, lists => lists.Where(list => !list.Hidden)
                                            .Include(list => list.Title,

    list => list.DefaultViewUrl));
    clientContext.ExecuteQuery();

    string html = "<h2>Lists in host web</h2>";
    html += "<ul>";
    foreach (var list in Lists) {
      html += "<li>" + list.Title + "</li>";
    }
    html += "</ul>";

    placeholderMainContent.Text = html;
  }

}
```

7. Test your code by pressing {F5} and starting a debugging session. You should be able to see that your query has returned just the lists than are not hidden.



8. Close the browser window to stop the debugging and return to Visual Studio.

Exercise 4: Using CSOM to Create a List in the Host Web

In this exercise, you will continue your work with CSOM by creating a new list in the host web.

1. Open the source file **default.aspx.cs** if it is not already open.
2. Modify the code in **cmdCreateCustomersList_Click** to create a new **Customers** list using the **Contacts** list type.
   a) Begin by adding the following code to create a variable named **spContext** to reference the current **SharePointContext** instance and then to create a **using** block which is initialized by obtaining a CSOM **ClientContext** for the current user.

```
protected void cmdCreateCustomersList_Click(object sender, EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

  using (ClientContext clientContext = spContext.CreateUserClientContextForSPHost()) {
    // add code here to create new list
  }

}
```

   b) Inside the **using** block, add code to load the CSOM Web object for the current site. Also create a variable name **listTitle** and give it a value of **Customers**.

```
using (ClientContext clientContext = spContext.CreateUserClientContextForSPHost()) {

  clientContext.Load(clientContext.Web);
  clientContext.ExecuteQuery();

  string listTitle = "Customers";

}
```

   c) Add the following code to the end of the **using** block to delete any existing list with a conflicting title.

```
// delete list if it exists
ExceptionHandlingScope scope = new ExceptionHandlingScope(clientContext);
using (scope.StartScope()) {
  using (scope.StartTry()) {
    clientContext.Web.Lists.GetByTitle(listTitle).DeleteObject();
  }
  using (scope.StartCatch()) { }
}
```

d) Just below the code you just added, add the following code to create a new SharePoint **Contacts** list with a title of **Customers**.

```
// create and initialize ListCreationInformation object
ListCreationInformation listInformation = new ListCreationInformation();
listInformation.Title = listTitle;
listInformation.Url = "Lists/Customers";
listInformation.QuickLaunchOption = QuickLaunchOptions.On;
listInformation.TemplateType = (int)ListTemplateType.Contacts;

// Add ListCreationInformation to lists collection and return list object
List list = clientContext.Web.Lists.Add(listInformation);

// modify additional list properties and update
list.OnQuickLaunch = true;
list.EnableAttachments = false;
list.Update();

// send command to server to create list
clientContext.ExecuteQuery();
```

e) Next, add the following code to create two sample items in the new Customers list.

```
// add an item to the list
ListItemCreationInformation lici1 = new ListItemCreationInformation();
var item1 = list.AddItem(lici1);
item1["Title"] = "Lennon";
item1["FirstName"] = "John";
item1.Update();

// add a second item
ListItemCreationInformation lici2 = new ListItemCreationInformation();
var item2 = list.AddItem(lici2);
item2["Title"] = "McCartney";
item2["FirstName"] = "Paul";
item2.Update();

// send add commands to server
clientContext.ExecuteQuery();
```

f) Finally, add the following code to display a message to the user that a new list has been created.

```
// add message to app's start page
placeholderMainContent.Text = "New list created";
```

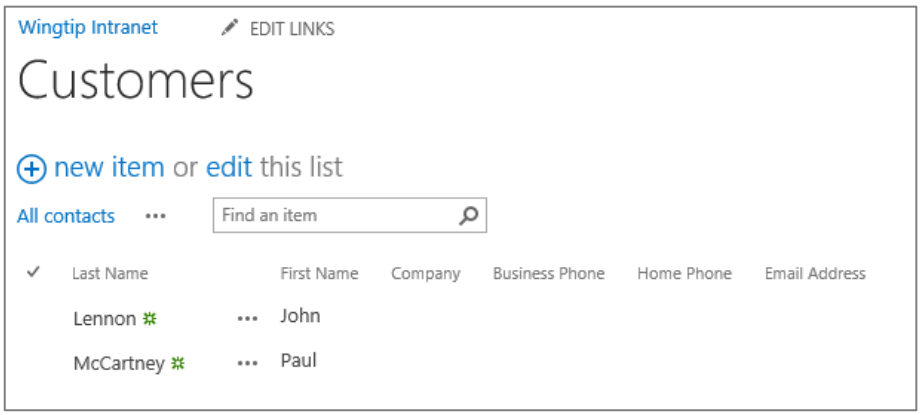
3. After you have completed the previous step, you should have a **cmdCreateCustomersList_Click** function that matches the following code listing. Make sure the structure of your code matches what you see here.

```
protected void cmdCreateCustomersList_Click(object sender, EventArgs e) {

  SharePointContext spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

  using (ClientContext clientContext = spContext.CreateUserClientContextForSPHost()) {

    clientContext.Load(clientContext.Web);
    clientContext.ExecuteQuery();
    string listTitle = "Customers";

    // delete list if it exists
    ExceptionHandlingScope scope = new ExceptionHandlingScope(clientContext);
    using (scope.StartScope()) {
      using (scope.StartTry()) {
        clientContext.Web.Lists.GetByTitle(listTitle).DeleteObject();
      }
      using (scope.StartCatch()) { }
    }

    // create and initialize ListCreationInformation object
    ListCreationInformation listInformation = new ListCreationInformation();
    listInformation.Title = listTitle;
```

```
    listInformation.Url = "Lists/Customers";
    listInformation.QuickLaunchOption = QuickLaunchOptions.On;
    listInformation.TemplateType = (int)ListTemplateType.Contacts;

    // Add ListCreationInformation to lists collection and return list object
    List list = clientContext.Web.Lists.Add(listInformation);

    // modify additional list properties and update
    list.OnQuickLaunch = true;
    list.EnableAttachments = false;
    list.Update();

    // send command to server to create list
    clientContext.ExecuteQuery();

    // add an item to the list
    ListItemCreationInformation lici1 = new ListItemCreationInformation();
    var item1 = list.AddItem(lici1);
    item1["Title"] = "Lennon";
    item1["FirstName"] = "John";
    item1.Update();

    // add a second item
    ListItemCreationInformation lici2 = new ListItemCreationInformation();
    var item2 = list.AddItem(lici2);
    item2["Title"] = "McCartney";
    item2["FirstName"] = "Paul";
    item2.Update();

    // send add commands to server
    clientContext.ExecuteQuery();

    // add message to app's start page
    placeholderMainContent.Text = "New list created";
  }

}
```

4. Test your code by pressing **{F5}** and starting a debugging session. Once the app start page has displayed, click the **Create List** button. After a second or two, you should see a message indicating that the list has been created.

5. Next, click the **Get Lists** button and verify that you can see the new **Customers** list in the list of lists.
6. Click the **Back to Host Web** link to navigate to the host web.
7. Click the **Customers** link in the Quick Launch bar of the Host Web and inspect the new **Customers** list. You should be able to see your sample items.



8. When you are done with your testing, close the browser to end the debugging sessions and close the project in Visual Studio.