**Contoso Jobs**

**Lab 5. Add OneDrive Support**

In this Lab you will add support for OneDrive when loading and saving the list of Jobs in the Contoso Jobs app.

As the Surface Hub does not store any data files between sessions it is important that any files or data that is required between sessions is stored in a cloud store or networked location. For an internal enterprise application this could be a UNC path to a predefined internal network drive. For an app that stores data that could be used in external locations it would make more sense to use a cloud store, such as OneDrive.

This Lab continues from the previous Lab, or you can start from the start point in the folder named *Start Contoso Jobs OneDrive*

1. Open the existing **Contoso Jobs.sln** solution file in **Visual Studio 2015**
2. In order to support OneDrive in a UWP App add the OneDrive Authentication Nuget package to the project.
3. In Microsoft Visual Studio, in **Solution Explorer**, right click on the **Contoso Jobs (Universal Windows)** project. In the context menu select Manage Nuget Packages…
4. A Nuget Contoso Jobs child window will open in Visual Studio. At the top select the Browse heading and in the search box enter **Microsoft.OneDriveSdk.Authentication**
5. Select the latest version (at the time of writing 1.0.8) and install this Nuget Package

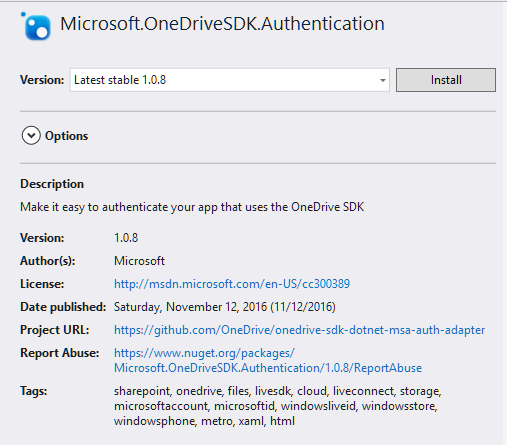


Figure Install OneDrive Nuget package

You will be asked to confirm the installation of a number of other packages upon which the OneDrive package is dependent. Select Ok. Then accept the license agreements presented.

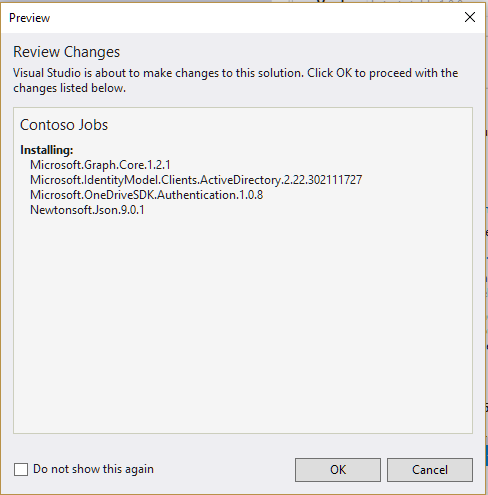


Figure Confirm installation of dependencies

1. Do the same for the **Microsoft.OneDriveSdk** Nuget package.
2. To use the OneDrive API you need to register your app and get a Client ID, that you will use in your application API calls. Currently OneDrive and OneDrive for Business use different app registration systems. You can find the instructions here on how to register your app for OneDrive <https://dev.onedrive.com/app-registration.htm>

As everyone reading this can get a personal OneDrive account, the rest of this lab will assume you have a personal OneDrive account and registered your app to obtain a Client ID.

1. In **Solution Explorer** right click on the **Common** folder and from the context menu select **Add | Class…** Name the class OneDrive.cs

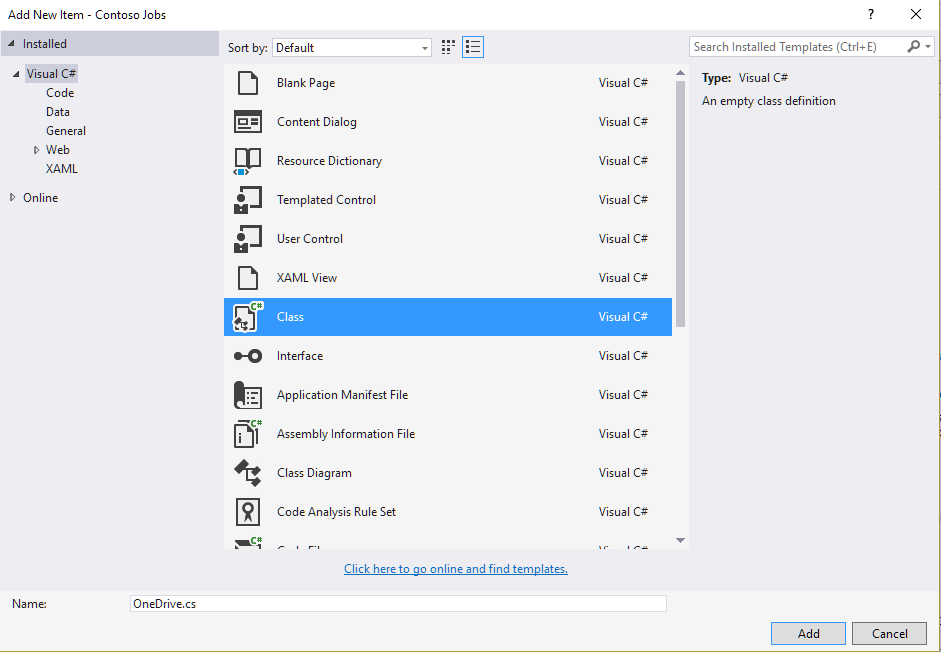


Figure Create a new OneDrive class

1. This new class will encapsulate all the OneDrive functionality to load and save files in your OneDrive folder.
2. Make the class public and add a member variable to the class for a OneDriveClient.

using Microsoft.OneDrive.Sdk;

...

public class OneDrive

{

private OneDriveClient oneDriveClient;

}

1. Add an Init method to the OneDrive class that will set up the OneDrive with a request to the customer to login. This uses the MsaAuthenticationProvider object and the client ID that you created for your app.

using Microsoft.OneDrive.Sdk.Authentication;

using System.Diagnostics;

...

public async Task Init()

{

try

{

string clientId = "<put your client id here>";

MsaAuthenticationProvider msaAuthProvider =

new MsaAuthenticationProvider(clientId,

"https://login.live.com/oauth20\_desktop.srf",

new string[] { "onedrive.readwrite", "wl.signin" },

new CredentialVault(clientId));

await msaAuthProvider.RestoreMostRecentFromCacheOrAuthenticateUserAsync();

oneDriveClient = new OneDriveClient("https://api.onedrive.com/v1.0",

msaAuthProvider);

}

catch (Exception ex)

{

Debug.WriteLine(ex.ToString());

}

}

1. Add a method to download files from OneDrive. This will call Init if the OneDriveClient is not already set up and then access a hardcoded path to an xml file in OneDrive that stores the jobs.

This file is copied into a local temporary file for use by the program.

using Windows.Storage;

using System.IO;

...

public async Task<bool> DownloadFileFromOneDrive()

{

bool downloaded = false;

try

{

if (oneDriveClient == null)

{

await Init();

}

Item item = await oneDriveClient

.Drive

.Root

.ItemWithPath("documents/ContosoJobs.xml")

.Request()

.GetAsync();

Stream contentStream = await oneDriveClient

.Drive

.Items[item.Id]

.Content

.Request()

.GetAsync();

//write the stream to our local storage for this session

StorageFolder storageFolder;

storageFolder = ApplicationData.Current.LocalFolder;

StorageFile jobFile = await storageFolder.CreateFileAsync(XmlService.JobsFile,

CreationCollisionOption.ReplaceExisting);

using (var writeStream = await jobFile.OpenStreamForWriteAsync())

{

int count = 0;

do

{

var buffer = new byte[1024];

count = contentStream.Read(buffer, 0, 1024);

await writeStream.WriteAsync(buffer, 0, count);

}

while (contentStream.CanRead && count > 0);

writeStream.Flush();

downloaded = true;

}

}

catch (Exception ex)

{

Debug.WriteLine(ex.ToString());

}

return downloaded;

}

1. Then add the code to upload a file to OneDrive. This will copy the local file to the OneDrive store

public async Task UploadFileToOneDrive()

{

try

{

if (oneDriveClient == null)

{

await Init();

}

StorageFolder storageFolder = ApplicationData.Current.LocalFolder;

if (await storageFolder.TryGetItemAsync(XmlService.JobsFile) != null)

{

var jobFile = await storageFolder.CreateFileAsync(XmlService.JobsFile,

CreationCollisionOption.OpenIfExists);

using (Stream readStream = await jobFile.OpenStreamForReadAsync())

{

var uploadedItem = await oneDriveClient

.Drive

.Root

.ItemWithPath("documents/ContosoJobs.xml")

.Content

.Request()

.PutAsync<Item>(readStream);

}

}

}

catch (Exception ex)

{

Debug.WriteLine(ex.ToString());

}

}

1. In the XmlService.cs code file change the JobsFile member to be internal as opposed to private

internal class XmlService

{

internal static string JobsFile = "jobs.xml";

To use this new OneDrive class you will make some modifications to the App class to download the xml file when the app starts and upload the file when the app is closed.

1. Open the App.xaml.cs code file and at the top of the App class add a member variable to store an instance of the new OneDrive class you just defined in the previous steps.

using Contoso\_Jobs.Common;

...

sealed partial class App : Application

{

private OneDrive oneDrive;

1. In the constructor for the App class instantiate the instance of the OneDrive class

public App()

{

this.InitializeComponent();

oneDrive = new OneDrive();

this.Suspending += OnSuspending;

}

1. In the OnLaunched method of the App class add a line of code to download the file from OneDrive

protected override async void OnLaunched(LaunchActivatedEventArgs e)

{

#if DEBUG

if (System.Diagnostics.Debugger.IsAttached)

{

this.DebugSettings.EnableFrameRateCounter = true;

}

#endif

await oneDrive.DownloadFileFromOneDrive();

…

1. Modify the OnSuspending method to save the jobs xml file to OneDrive

private async void OnSuspending(object sender, SuspendingEventArgs e)

{

var deferral = e.SuspendingOperation.GetDeferral();

await oneDrive.UploadFileToOneDrive();

deferral.Complete();

}

Build and run the application (F5). You are now downloading and uploading the storage file to OneDrive when the app launches and suspends. This provides a starting point for you to think about how your application will use a network or cloud store to save data between sessions.

By using OneDrive you have now created a new problem, it is not good practice on a Surface Hub, which is typically in a shared environment, to request someone enter their username and password. The OneDrive authentication does require this. In order to solve this you need a different way for the customer to authenticate themselves.

How might this authentication be achieved without requiring someone to enter their username and password on a shared device?