

Hand­s-on lab

Lab: Connecting a UWP Client to Azure Mobile Apps

September 2015

Contents

[Overview 3](#_Toc428893468)

[Exercise 1: Getting Started with UWP 5](#_Toc428893469)

[Task 1 – Create a blank Universal Windows app 5](#_Toc428893470)

[Task 2 - Explore the template 8](#_Toc428893471)

[Exercise 2: Hello World Across Devices 10](#_Toc428893472)

[Task 1 – Display a greeting 10](#_Toc428893473)

[Task 2 - Detect the device family 12](#_Toc428893474)

[Task 3 – Dynamically show the dimensions of the app window 14](#_Toc428893475)

[Task 4 - Deploy to the Mobile emulator 17](#_Toc428893476)

[Task 5 - Deploy to IoT devices (optional) 18](#_Toc428893477)

[Exercise 3: Hello World in Blend 20](#_Toc428893478)

[Task 1 – Create a new project in Blend 20](#_Toc428893479)

[Task 2 – Generate sample data 22](#_Toc428893480)

[Summary 27](#_Toc428893481)

Overview

Windows 10 introduces the Universal Windows Platform (UWP), which further evolves the Windows Runtime model and brings it into the Windows 10 unified core. As part of the core, the UWP now provides a common app platform available on every device that runs Windows 10. With this evolution, apps that target the UWP can call APIs specific to the device family in addition to the WinRT APIs that are common to all devices. The UWP provides a guaranteed core API layer across devices. With UWP, you can create a single app package that can be installed onto a wide range of devices.

In this lab, you will use the Universal Windows App Development Tools to build a Hello World app that runs on all Windows 10 devices. Your app will display information about the device it is running on, including the device family and current app window size. You will also create a Hello World app directly from Blend and leverage Blend’s ability to generate sample data.

# Objectives

* 1. This lab will show you how to:
  + Create a Universal Windows app from the Blank App template
  + Display a greeting in your app
  + Detect and display the device family
  + Dynamically display the app window size
  + Deploy to the Local Machine
  + Deploy to the Mobile emulator
  + Deploy to an IoT device
  + Generate sample data in Blend

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015
  + Windows 10 Mobile Emulator

# Optional add-ons

* 1. If you wish to complete the optional tasks in this lab, you will need:
  + An IoT device running Windows 10
  + A display that connects to the IoT device

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015. Choose a custom install and ensure that the Universal Windows App Development Tools are selected from the optional features list.
  4. Install the Windows 10 Mobile Emulator.
  5. Optional: Install Windows 10 on an IoT device.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Getting started with UWP
  3. Hello World Across Devices
  4. Hello World in Blend
  5. Estimated time to complete this lab:  **30 to 45 minutes**.

Exercise 1: Getting Started with UWP

* 1. The Universal Windows App Development Tools provides a template to help you get started building your own UWP apps. In this exercise, you will create a project from the Blank App template and explore what it has to offer.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose File > New> Project to open the New Project dialog. Navigate to Installed > Templates > Visual C# > Windows > Universal and select the Blank App (Universal Windows) template.

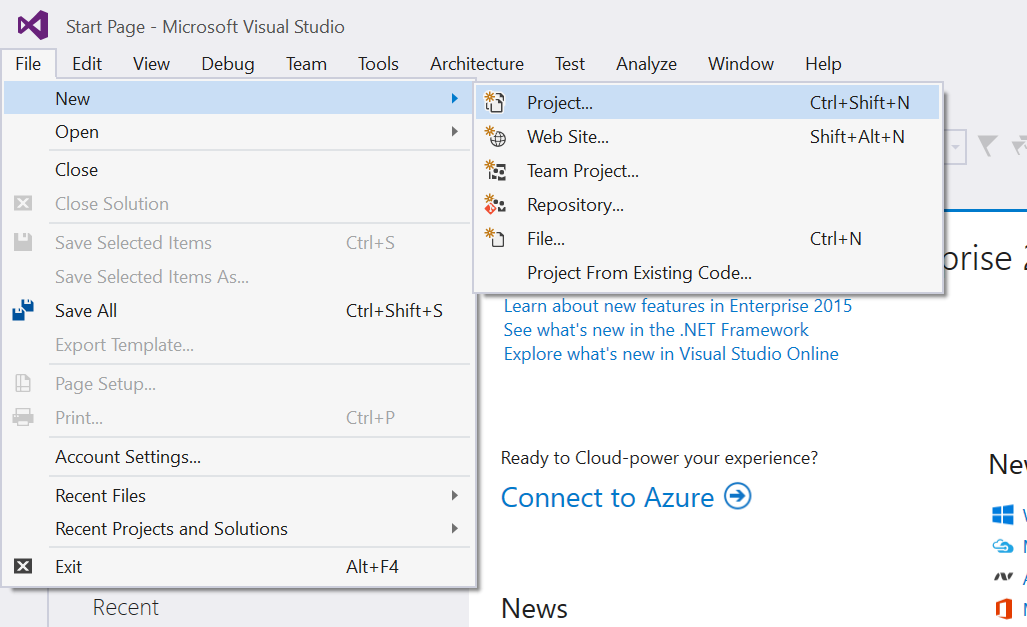


Figure 1

Open the New Project dialog in Visual Studio 2015.

1. Name your project **HelloUWP** and select the filesystem location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.
2. Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect **Add to source control** if you don't wish to version your work. Click **OK** to create the project.

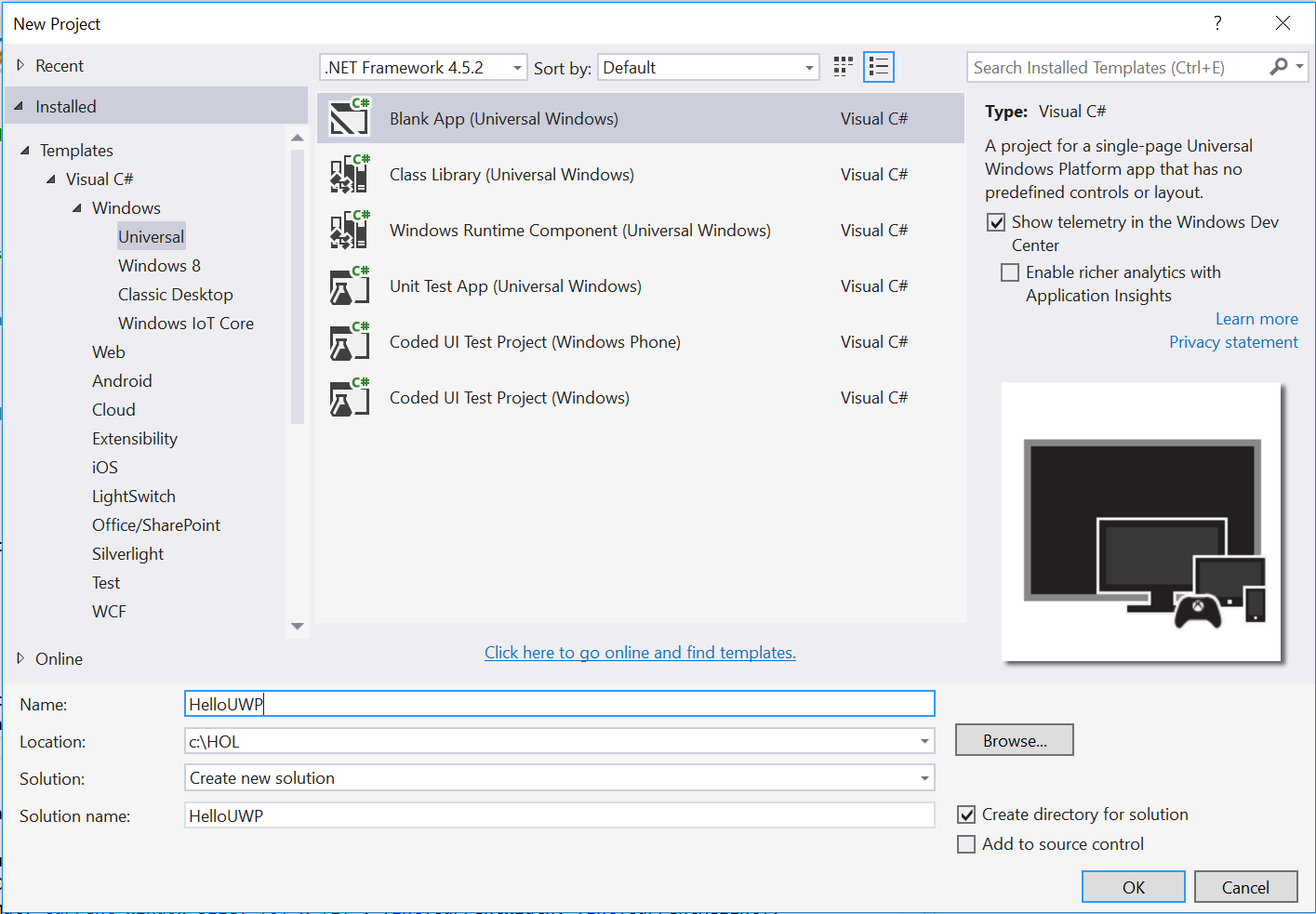


Figure 2

Create a new Blank App project in Visual Studio 2015.

1. Set your Solution Configuration to Debug and your Solution Platform to x86. Select Local Machine from the Debug Target dropdown next to the Start Debugging Button.



Figure 3

* + 1. Configure your app to run on the Local Machine.
  1. **Note:**  is the Start Debugging button.

1. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

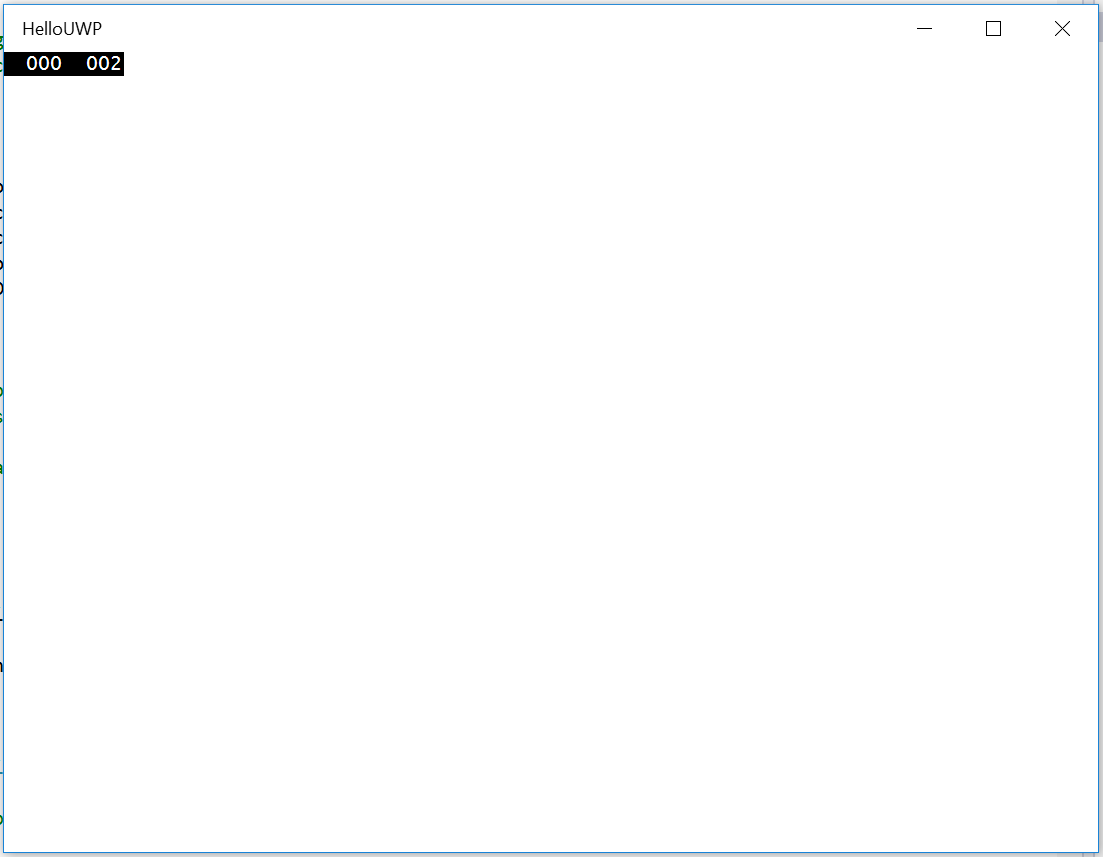


Figure 4

The Blank universal app running in Desktop mode.

* 1. **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
  2. In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, we will turn it off.

1. Return to Visual Studio and stop debugging.

Exercise 3: Enable offline sync for your app

This exercise shows you how to add offline support to a UWP app using an Azure Mobile App backend. Offline sync allows end-users to interact with a mobile app--viewing, adding, or modifying data--even when there is no network connection. Changes are stored in a local database; once the device is back online, these changes are synced with the remote backend.

In this exercise, you will update the Windows 8.1 app project from the tutorial Create a Windows app to support the offline features of Azure Mobile Apps.

Task 1 – Update the client app to support offline features

* 1. Azure Mobile App offline features allow you to interact with a local database when you are in an offline scenario. To use these features in your app, you initialize a **MobileServiceClient.SyncContext** to a local store. Then reference your table through the **IMobileServiceSyncTable** interface. In this tutorial we use SQLite for the local store.
  2. The first task is to open the ToDoUWPDevWorkshop solution you created in the previous exercise.

1. Navigate to the file location where you saved your **ToDoUWPDevWorkshop** app in Exercise 1. Open **ToDoUWPDevWorkshop.sln** in Visual Studio 2015.
2. Install the SQLite runtime for Universal Windows Platform.
   1. In Visual Studio, on the **Tools** menu, click **Extensions and Updates**
   2. In the left pane of the Extensions and Updates wizard, click **Online**
   3. In the search box at the top right of the window, enter **SQLite**
   4. When the Search Results display, scroll down until you see **SQLite for Universal App Platform**. If this SDK is not already installed on your system, select this item, and then click the **Download** button

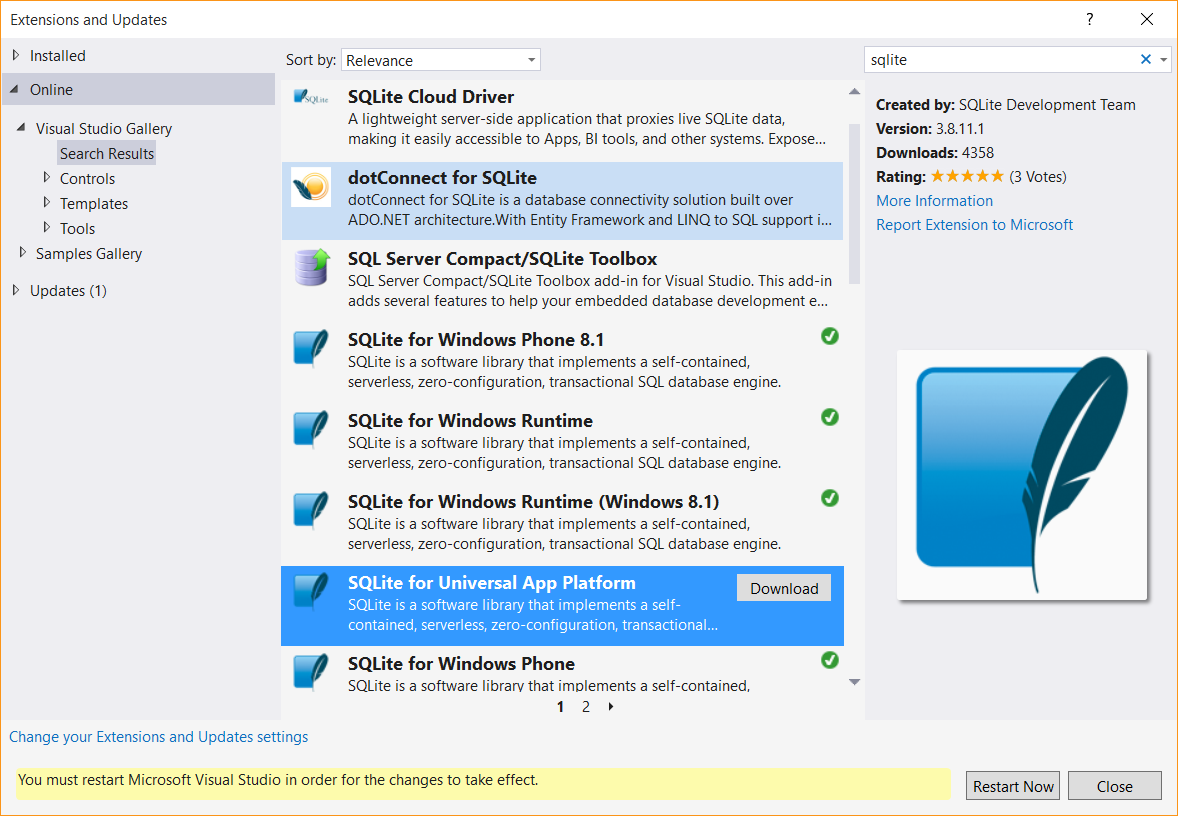


Figure 3

* + 1. Download and install the SQLite for Universal App Platform SDK.
  1. When the UAC prompt displays, click **OK.**
  2. In the VSIX Installer window, click **Install**. After the extension installs, click **Close**.
  3. Click the **Restart Now** button on the Extensions and Updates window and wait for Visual Studio 2015 to restart.

1. Add a reference to the SQLite runtime dll to your project.
   1. In Solution Explorer, right click the References node in the project tree and click **Add Reference** to run the Reference Manager.
   2. In the "Universal Windows" category, select the option "Extensions" in the navigation pane at the left.

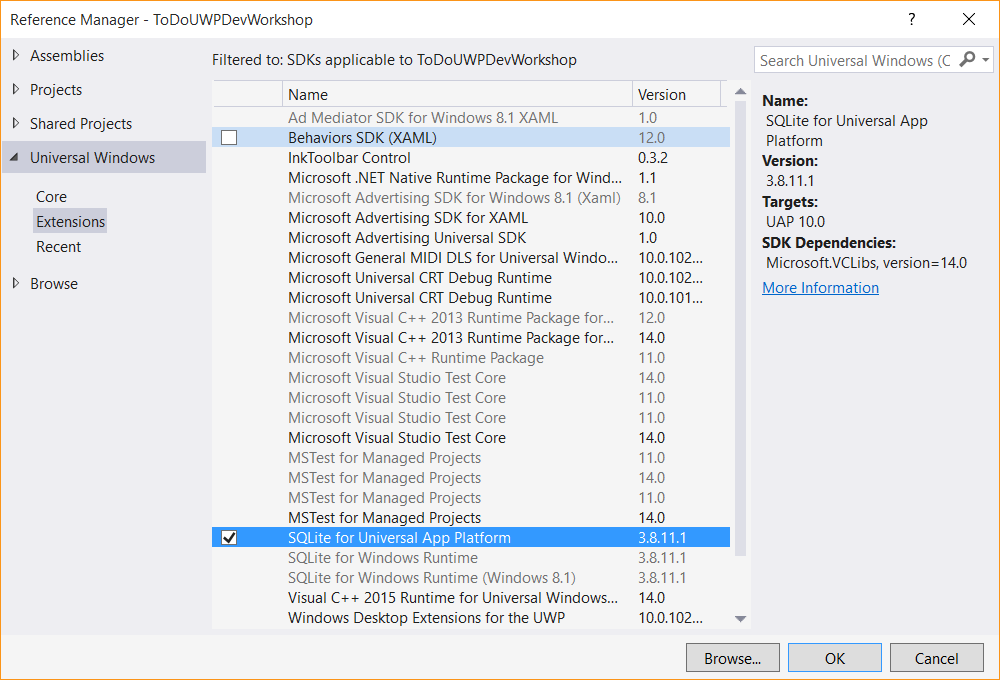


Figure 6

* + 1. Add a reference to the SQLite for Universal App Platform dll to your project.
  1. Select **SQLite for Universal App Platform**, and then click **OK**.

1. Install the WindowsAzure.MobileServices.SQLiteStore NuGet package.
   1. In Solution Explorer, right click the project and click **Manage Nuget Packages** to run NuGet Package Manager.
   2. In the "Online" tab, select the option "Include Prerelease" in the dropdown at the top. Search for **SQLiteStore** to locate the 2.0.0-beta of **WindowsAzure.MobileServices.SQLiteStore**.
   3. Then, click **Install** to add the NuGet reference to the project.
   4. Click **I Accept** on the License Acceptance window.
2. In Solution Explorer, open the **MainPage.cs** file. Uncomment the following using statements at the top of the file:
   * 1. C#
   1. using Microsoft.WindowsAzure.MobileServices.SQLiteStore; // offline sync
   2. using Microsoft.WindowsAzure.MobileServices.Sync; // offline sync
3. In MainPage.cs, comment the line of code that initializes todoTable as an IMobileServiceTable. Uncomment the line of code that initializes todoTable as an IMobileServiceSyncTable:

C#

* 1. //private IMobileServiceTable<TodoItem> todoTable = App.MobileService.GetTable<TodoItem>();
  2. private IMobileServiceSyncTable<TodoItem> todoTable =   
      App.MobileService.GetSyncTable<TodoItem>(); // offline sync

1. In **MainPage.cs**, in the region marked **Offline sync**, uncomment the methods **InitLocalStoreAsync** and **SyncAsync**. The method InitLocalStoreAsync initializes the client sync context with a SQLite store. In Visual Studio, you can select all commented lines and use the Ctrl+K+U keyboard shortcut to uncomment.

Notice in **SyncAsync** a push operation is executed off the MobileServiceClient.SyncContext instead of the IMobileServicesSyncTable. This is because the context tracks changes made by the client for all tables. This is to cover scenarios where there are relationships between tables. For more information on this behavior, see [Offline Data Sync in Azure Mobile Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/).

C#

* 1. private async Task InitLocalStoreAsync()
  2. {
  3. if (!App.MobileService.SyncContext.IsInitialized)
  4. {
  5. var store = new MobileServiceSQLiteStore("localstore.db");
  6. store.DefineTable<TodoItem>();
  7. await App.MobileService.SyncContext.InitializeAsync(store);
  8. }
  9. await SyncAsync();
  10. }
  11. private async Task SyncAsync()
  12. {
  13. await App.MobileService.SyncContext.PushAsync();
  14. await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
  15. }

1. **Note:** In this PullAsync example, we retrieve all records in the remote todoTable, but it is also possible to filter records by passing a query. The first parameter to PullAsync is a query ID that is used for incremental sync, which uses the UpdatedAt timestamp to get only records modified since the last sync. The query ID should be a descriptive string that is unique for each logical query in your client application. To opt-out of incremental sync, pass null as the query ID. This will retrieve all records on each pull operation, which is potentially inefficient.
2. In the OnNavigatedTo event handler, uncomment the call to InitLocalStoreAsync:

C#

* 1. protected override async void OnNavigatedTo(NavigationEventArgs e)
  2. {
  3. await InitLocalStoreAsync(); // offline sync
  4. // await RefreshTodoItems();
  5. }

1. Uncomment the 3 calls to SyncAsync in the methods InsertTodoItem, UpdateCheckedTodoItem, and ButtonRefresh\_Click:

C#

* 1. private async Task InsertTodoItem(TodoItem todoItem)
  2. {
  3. await todoTable.InsertAsync(todoItem);
  4. items.Add(todoItem);
  5. await SyncAsync(); // offline sync
  6. }

1. ...
   1. private async Task UpdateCheckedTodoItem(TodoItem item)
   2. {
   3. await todoTable.UpdateAsync(item);
   4. items.Remove(item);
   5. ListItems.Focus(Windows.UI.Xaml.FocusState.Unfocused);
   6. await SyncAsync(); // offline sync
   7. }
   8. private async void ButtonRefresh\_Click(object sender, RoutedEventArgs e)
   9. {
   10. ButtonRefresh.IsEnabled = false;
   11. await SyncAsync(); // offline sync
   12. await RefreshTodoItems();
   13. ButtonRefresh.IsEnabled = true;
   14. }
2. Modify the code in the SyncAsync method to add exception handlers. In an offline situation a MobileServicePushFailedException will be thrown with PushResult.Status == CancelledByNetworkError.

C#

* 1. private async Task SyncAsync()
  2. {
  3. String errorString = null;
  4. try
  5. {
  6. await App.MobileService.SyncContext.PushAsync();
  7. // first param is query ID, used for incremental sync  
      await todoTable.PullAsync("todoItems", todoTable.CreateQuery());   
      }
  8. catch (MobileServicePushFailedException ex)
  9. {
  10. errorString = "Push failed because of sync errors. " +
  11. "You may be offine.\nMessage: " +
  12. ex.Message + "\nPushResult.Status: " +
  13. ex.PushResult.Status.ToString();
  14. }
  15. catch (Exception ex)
  16. {
  17. errorString = "Pull failed: " + ex.Message +
  18. "\n\nIf you are still in an offline scenario, " +
  19. "you can try your Pull again when connected with " +
  20. "your Mobile Service.";
  21. }
  22. if (errorString != null)
  23. {
  24. MessageDialog d = new MessageDialog(errorString);
  25. await d.ShowAsync();
  26. }
  27. }

**Note:** The MobileServicePushFailedException can occur for both a push and a pull operation. It can occur for a pull because the pull operation internally executes a push to make sure all tables along with any relationships are consistent.

1. In Visual Studio, press the F5 key to rebuild and run the client app. The app will behave the same as it did before the offline sync changes, because it does a sync operation on the insert, update, and refresh operations. However, it will populate a local database which can be used in an offline scenario. We will test the offline scenario in the next section now that the local database is populated.

Task 2 – Update the client app to support offline features

* 1. Azure Mobile App offline features allow you to interact with a local database when you are in an offline scenario. To use these features in your app, you initialize a **MobileServiceClient.SyncContext** to a local store. Then reference your table through the **IMobileServiceSyncTable** interface. In this tutorial we use SQLite for the local store.
  2. The first task is to open the ToDoUWPDevWorkshop solution you created in the previous exercise.

1. Navigate to the file location where you saved your **ToDoUWPDevWorkshop** app in Exercise 1. Open **ToDoUWPDevWorkshop.sln** in Visual Studio 2015.

Summary

* 1. The Universal Windows Platform is a powerful collection of core APIs that allows you to target a wide range of devices with a single app. In this lab, you evolved a blank app created from a template into a Hello World app that displays device-specific information across all Windows 10 devices. You also learned how to leverage sample data in Blend to quickly start building and visualizing your app UI. In the next lab, you will learn how to navigate within a UWP app, handle back navigation with the shell-drawn back button, and implement custom back and forward navigation controls.