

Hand­s-on lab

Lab: Connecting a UWP Client to Azure Mobile Apps

September 2015

Contents

[Overview 3](#_Toc430879729)

[Exercise 1: Getting Started with Azure App Service Mobile Apps 5](#_Toc430879730)

[Task 1 – Understand the Azure Portal support for Azure 5](#_Toc430879731)

[Task 2 – Run the server project on your PC 8](#_Toc430879732)

[Task 3 – Open and run the client app project 9](#_Toc430879733)

[Exercise 2: Enable authentication for your app and connect to the Cloud 10](#_Toc430879734)

[Task 1 – Update the client app to support offline features 10](#_Toc430879735)

[Exercise 3: Enable offline sync for your app 10](#_Toc430879736)

[Task 1 – Update the client app to support offline features 10](#_Toc430879737)

[Task 2 – Update the client app to support offline features 16](#_Toc430879738)

[Summary 16](#_Toc430879739)

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 Azure App Service Mobile Apps

* 1. Azure App Service is a fully managed Platform as a Service (PaaS) offering for professional developers that brings a rich set of capabilities to web, mobile and integration scenarios. Mobile Apps in Azure App Service offer a highly scalable, globally available mobile application development platform for Enterprise Developers and System Integrators that brings a rich set of capabilities to mobile developers.
  2. With Mobile Apps you can:
* Build native and cross platform apps - whether you're building native iOS, Android, and Windows apps or cross-platform Xamarin or Cordova (Phonegap) apps, you can take advantage of App Service using native SDKs.
* Connect to your enterprise systems - with Mobile Apps you can add corporate sign on in minutes, and connect to your enterprise on-premises or cloud resources.
* Connect to SaaS APIs easily - with more than 40 SaaS API connectors, you can easily integrate your app with SaaS APIs your enterprise uses today.
* Build offline-ready apps with sync - make your mobile workforce productive by building apps that work offline and use Mobile Apps to sync data in the background when connectivity is present with any of your enterprise data sources or SaaS APIs.
* Push Notifications to millions in seconds - engage your customers with instant push notifications on any device, personalized to their needs, sent when the time is right.

In this exercise, you will run a Windows 10 UWP client app which is similar to the ToDo app you can download from the Azure Portal when you create an Azure mobile app backend. You will connect the client app to Azure mobile app backend which you will run locally on your PC in the Azure development environment. You will connect to the real Azure backend running in the cloud in the following exercise.

Task 1 – Understand the Azure Portal support for Azure

We will begin by understanding how to create an Azure App Service Mobile App backend in the Azure Portal.

1. You do not need to create an Azure App Service mobile app backend for this hands-on lab as we have pre-created one for you.  
   If you want to learn how to create a new Azure mobile app backend, follow the instructions in the section **Create a new Azure mobile app backend** at <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-preview/> .  
     
   That tutorial walks you through how to use the wizard in Azure Portal to create a new Azure mobile app backend:

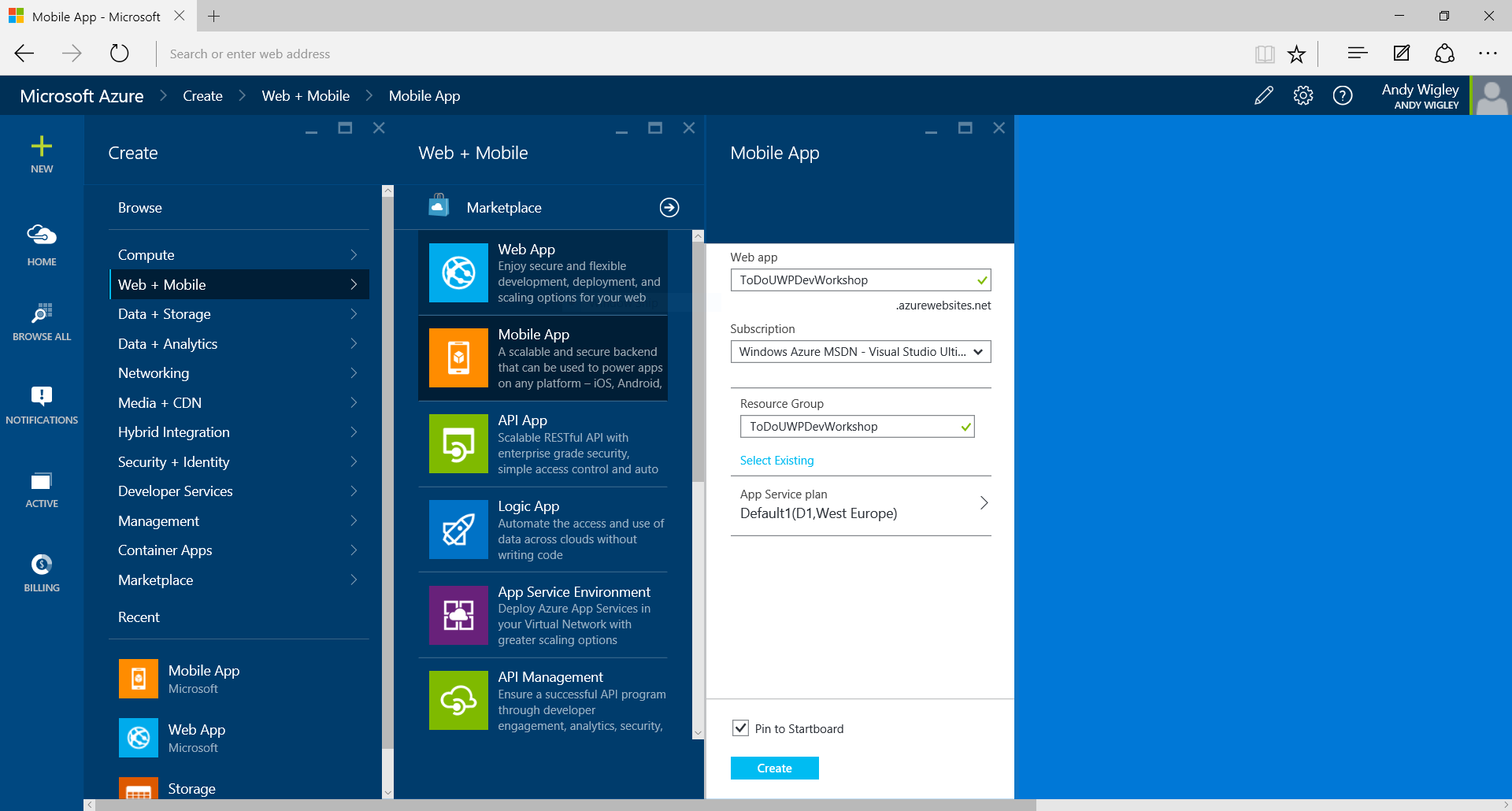


Figure 1

Starting to create a new Azure mobile app backend in Azure Portal.

1. If you had created your own Azure mobile apps backend, you would see a blade in the Azure Portal with information about your mobile app backend similar to this:

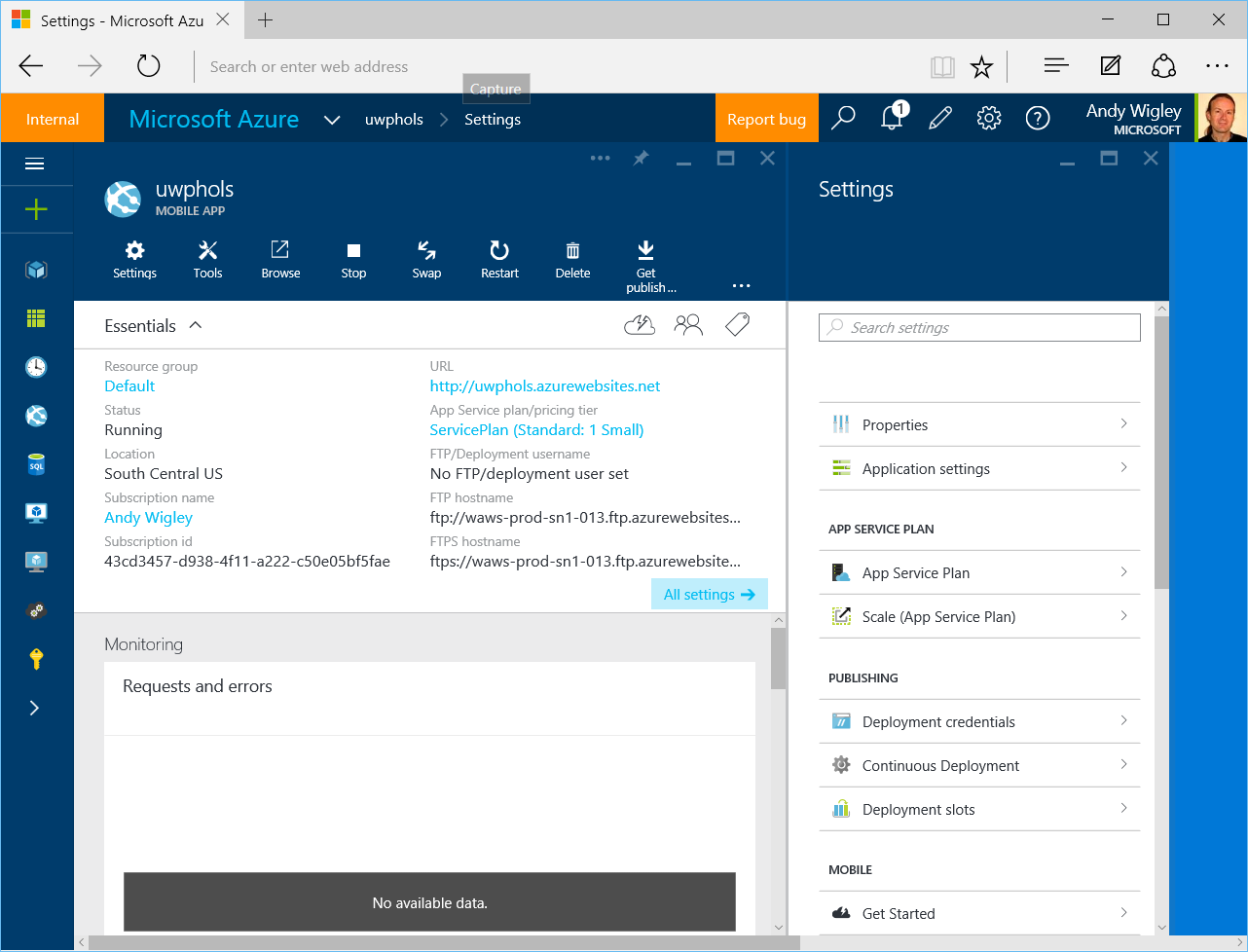


Figure 2

The blade for an Azure mobile app backend in Azure Portal.

1. On the **Settings** pane, you will find a section called Mobile containing a number of options including **Get Started**. If you were to click on this, you would open the **Quickstart** pane where you can get instructions on connecting an existing app to your Azure mobile apps backend, or you can download projects for an example server project and an example mobile app client. You can choose between a number of different clients, including Windows (C#), iOS (Objective-C), iOS (Swift), HTML/JavaScript, Xamarin.Android, Xamarin.iOS or Xamarin.Forms.

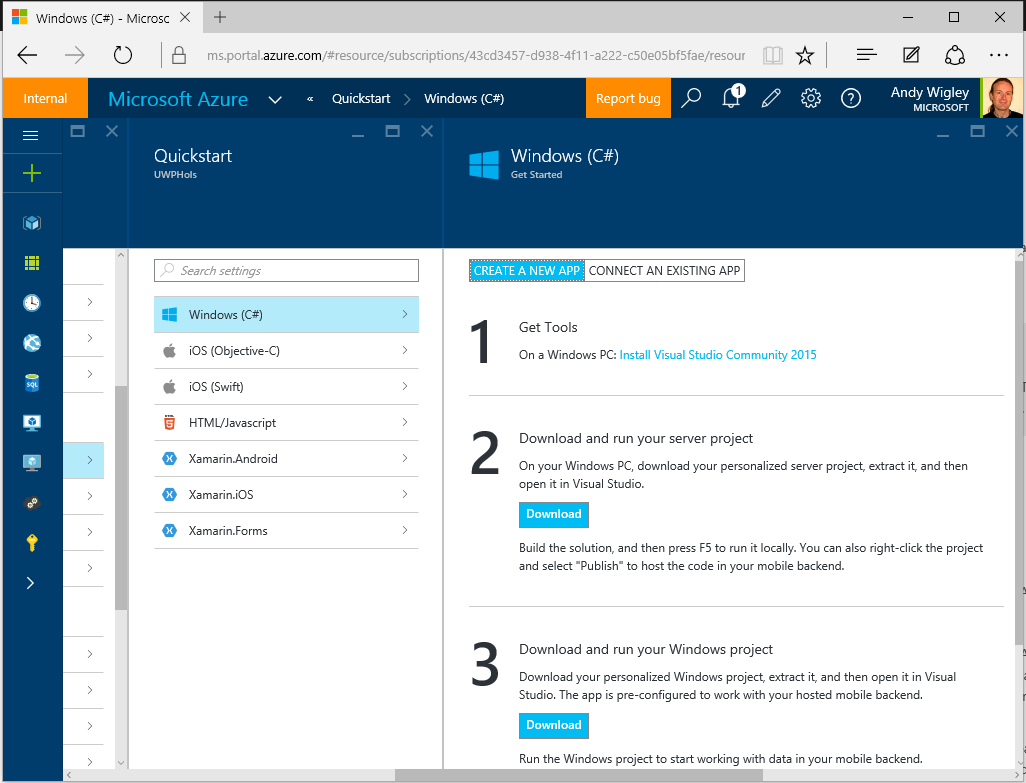


Figure 3

Azure mobile app Quickstart options in Azure Portal.

1. At the time of this writing (September 2015), the Quickstart download for Windows (C#) is a Windows 8.1 universal app project. For the purposes of this lab, we have created a UWP version of this client.

Task 2 – Run the server project on your PC

We will examine the server code which was downloaded when the Azure mobile apps backend that you will use later in this lab was created. You will then run it locally on your PC.

1. Open **Visual Studio 2015**. On the **File** menu, click **Open Project/Solution**. Navigate to the folder where you have saved the code for this lab, and open **Exercise 1\Begin-Cloud\UWPHolsService.sln**.
2. Build the project to restore the NuGet packages.

**Note:** This project contains the C# code for a .NET backend to your Azure mobile app backend. You can instead choose to have your backend logic implemented in NodeJS. See the Azure App Service mobile apps documentation for more information.

1. This project contains the C# code for the backend logic. In the **DataObjects** folder, open **ToDoItem.cs**. This class defines the data objects that the cloud service stores in a SQL Azure database:
   * 1. C#
   1. public class TodoItem : EntityData
   2. {
   3. public string Text { get; set; }
   4. public bool Complete { get; set; }
   5. }
2. In the **Controllers** folder, open **TodoItemController.cs**. This class contains the code that implements the CRUD operations offered by the cloud service REST API:
   * 1. C#
   1. public class TodoItemController : TableController<TodoItem>
   2. {
   3. protected override void Initialize(
   4. HttpControllerContext controllerContext)
   5. {
   6. base.Initialize(controllerContext);
   7. UWPHolsContext context = new UWPHolsContext();
   8. DomainManager =   
       new EntityDomainManager<TodoItem>(context, Request);
   9. }
   10. // GET tables/TodoItem
   11. public IQueryable<TodoItem> GetAllTodoItems()
   12. {
   13. return Query();
   14. }
   15. // GET tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   16. public SingleResult<TodoItem> GetTodoItem(string id)
   17. {
   18. return Lookup(id);
   19. }
   20. // PATCH tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   21. public Task<TodoItem> PatchTodoItem(string id, Delta<TodoItem> patch)
   22. {
   23. return UpdateAsync(id, patch);
   24. }
   25. // POST tables/TodoItem
   26. public async Task<IHttpActionResult> PostTodoItem(TodoItem item)
   27. {
   28. TodoItem current = await InsertAsync(item);
   29. return CreatedAtRoute("Tables", new { id = current.Id }, current);
   30. }
   31. // DELETE tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   32. public Task DeleteTodoItem(string id)
   33. {
   34. return DeleteAsync(id);
   35. }
   36. }
3. In the **App\_Start** folder, open file **Startup.MobileApp.cs**. This file contains configuration code for the service, including the **Seed** method in the **UWPHolsInitializer** class. This method runs the first time a request is made to the REST service, and inserts the two data items created in this method into the database (just for the purposes of this Quickstart):

**C#**

protected override void Seed(UWPHolsContext context)

{

List<TodoItem> todoItems = new List<TodoItem>

{

new TodoItem { Id = Guid.NewGuid().ToString(),

Text = "First item", Complete = false },

new TodoItem { Id = Guid.NewGuid().ToString(),

Text = "Second item", Complete = false },

};

foreach (TodoItem todoItem in todoItems)

{

context.Set<TodoItem>().Add(todoItem);

}

base.Seed(context);

}

1. Set your Solution Configuration to Debug and your Solution Platform to Any CPU. Select Microsoft Edge from the Debug Target dropdown next to the Start Debugging Button.

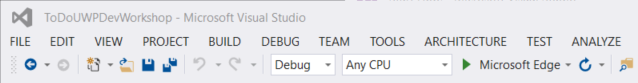


Figure 4

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

1. Build and run your app. You will see a web browser window showing the default ‘running’ information screen for the Azure mobile app backend.

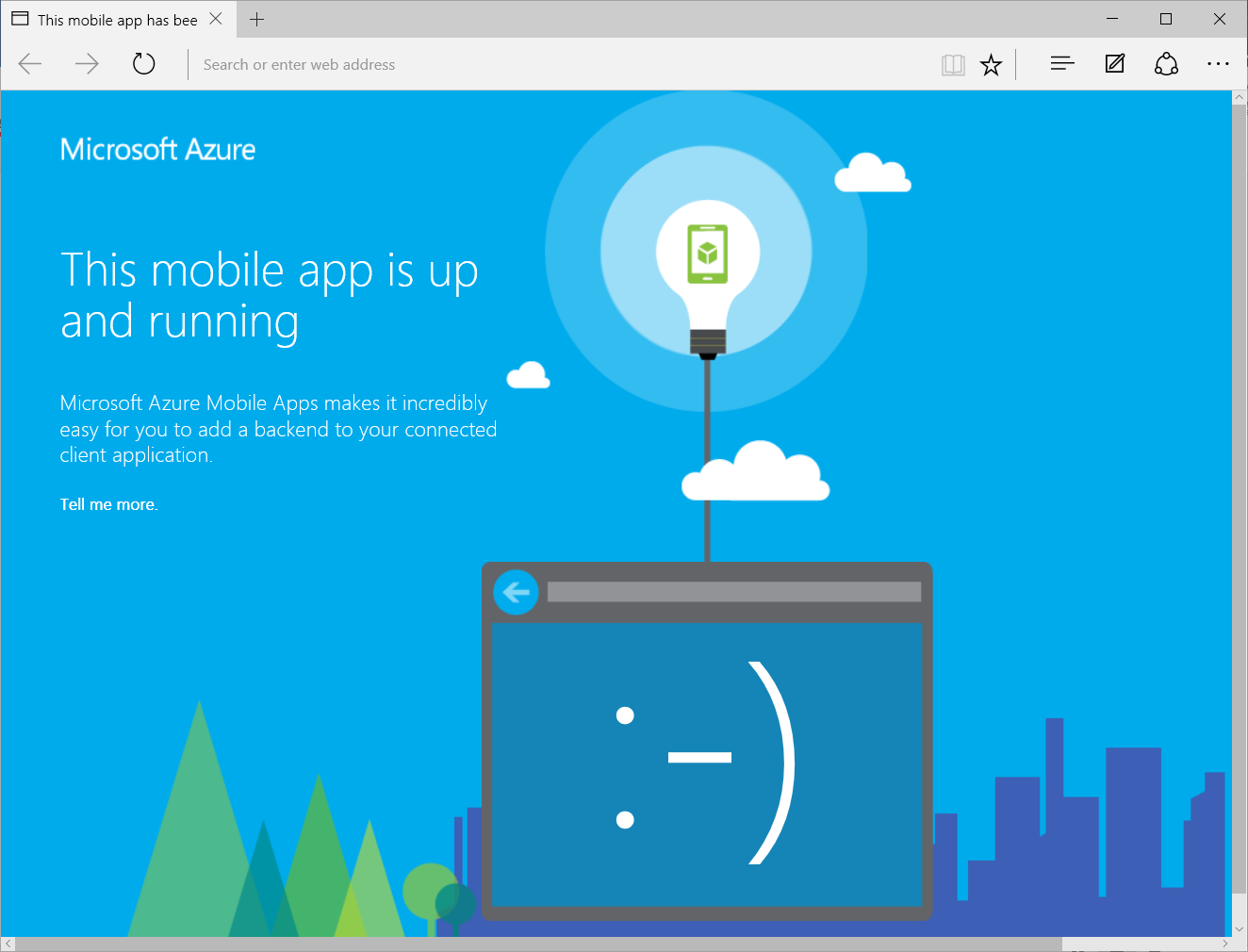


Figure 4

The Azure mobile app backend running locally on your PC.

* 1. **Note:** Running your backend service locally gives you one way ofdebugging problems with your application.

Task 3 – Open and run the client app project

We will examine the Windows UWP app code that connects to an Azure mobile app backend and will run it configured to connect to the backend service that is running locally on your PC.

1. Open a second instance of **Visual Studio 2015**. On the **File** menu, click **Open Project/Solution**. Navigate to the folder where you have saved the code for this lab, and open **Exercise 1\Begin-Client\ToDoUWPDevWorkshop.sln**.
2. 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.

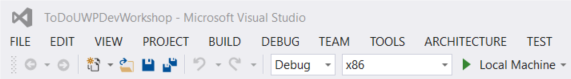


Figure 4

* + 1. Configure your client app to run on the Local Machine.

1. Build the project to restore the NuGet packages.
2. Open **App.xaml.cs**. Notice how at the top of this class, there is code to instantiate a **MobileServiceClient** object. The first declaration is the form you use to connect to the cloud – leave this commented out. The second declaration that is NOT commented out is the form that connects to the local version of the cloud service that you just started running in the previous steps.

**C#**

sealed partial class App : Application

{

// Uncomment this code for configuring the MobileServiceClient to

// communicate with the Azure Mobile Service and

// Azure Gateway using the application key. You're all set to start

// working with your Mobile Service!

//public static MobileServiceClient MobileService =

// new MobileServiceClient(

// "https://uwphols.azurewebsites.net",

// "",

// ""

//);

// Use this code for configuring the MobileServiceClient to

// communicate with your local

// test project for debugging purposes.

public static MobileServiceClient MobileService =

new MobileServiceClient(

"http://localhost:50781"

);

...

1. Open **MainPage.xaml.cs**. In this class you can see the way that your application code communicates with the cloud backend service to download objects from the service and to insert, update and delete objects.  
   At the top of the class, the field **todoTable** of type **IMobileServiceTable<TodoItem>** is initialized by calling **App.MobileService.GetTable<TodoItem>()**. This object is used throughout the class to perform strongly typed data operations for that table.  
   For example, to insert a new data item, you use the **InsertAsync** method of that **todoTable** object:

**C#**

private IMobileServiceTable<TodoItem> todoTable =

App.MobileService.GetTable<TodoItem>();

...

private async Task InsertTodoItem(TodoItem todoItem)

{

// This code inserts a new TodoItem into the database. When the

// operation completes and Mobile Services has assigned an Id,

// the item is added to the CollectionView

await todoTable.InsertAsync(todoItem);

items.Add(todoItem);

}

...

1. Build and run your app. When your client app runs, code in the OnNavigatedTo method of MainPage calls RefreshTodoitems which calls your backend service to retrieve any ToDo items stored in the backend database.
   1. **Note**: If your app fails with an HttpRequestException, it is probably because you stopped debugging your UWPHolsService at the end of the previous task. Run the backend service project again, leave it running, and then run the client app in a separate copy of Visual Studio.
2. As this is the first time the service has been accessed, it will automatically configure the database and run the Seed method that you looked at previously, which inserts two items. After a few moments, you should see the two items displayed in the client app UI.

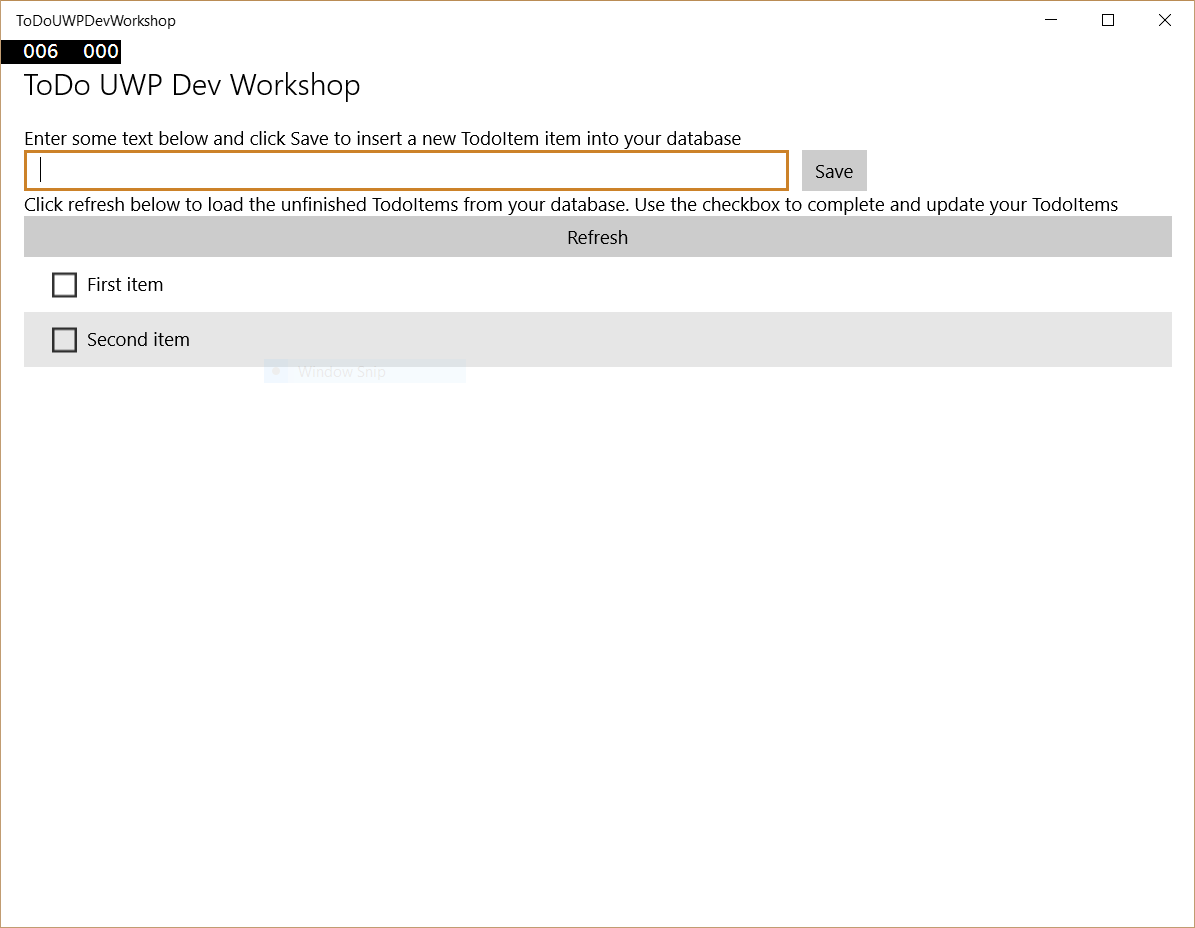


Figure 7

The client app retrieves the two ‘Seed’ data items from the backend service.

1. Insert some new items in addition to the two items that were added by the Seed method in the service.

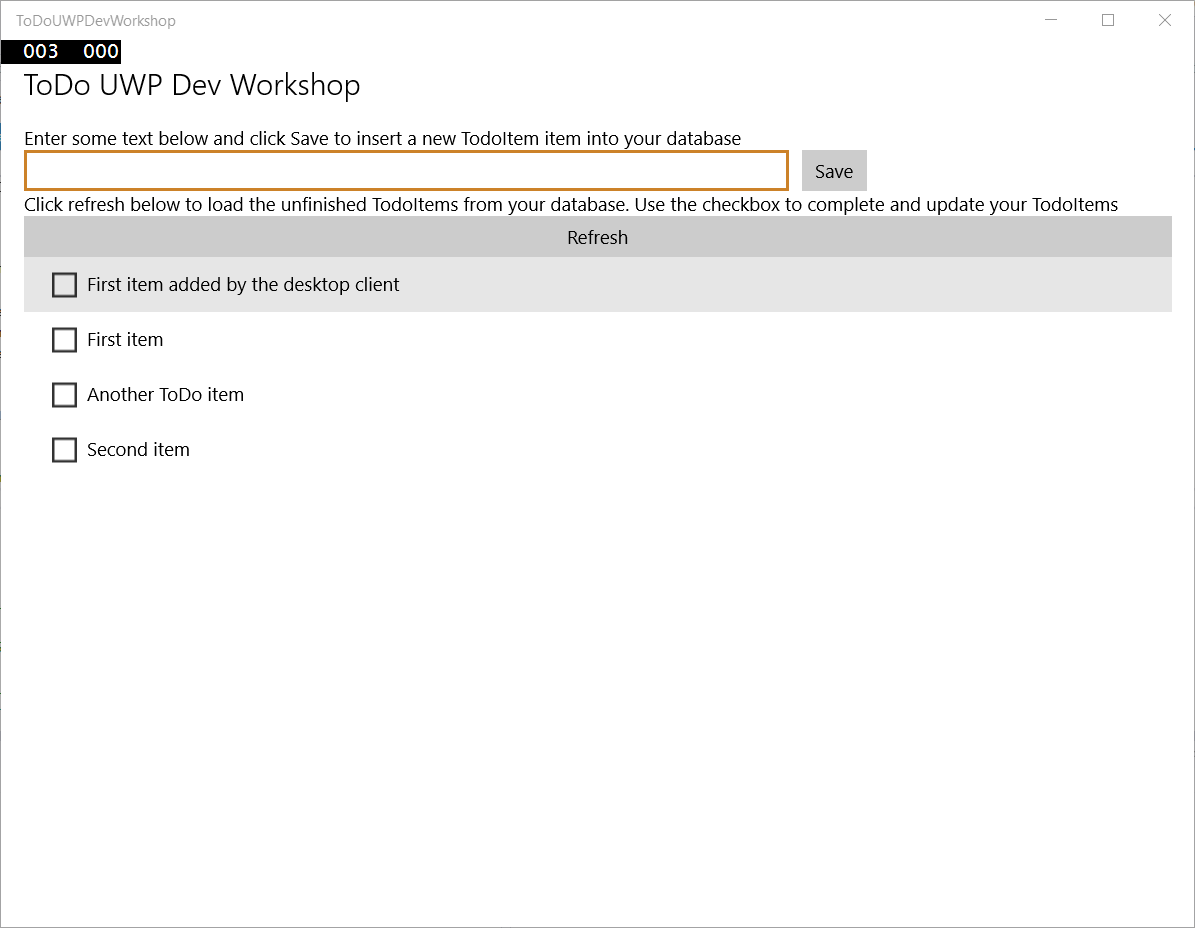


Figure 8

Adding new data items.

1. Stop debugging.
2. If your development PC has the Windows 10 Mobile emulators installed, select one of the **Mobile Emulator …** options from the Debug Target dropdown next to the Start Debugging Button. If you have a real phone device connected and enabled for Developer Mode, select **Device** from the Debug Target dropdown. If neither of these options are available, you will have to skip the rest of this exercise.

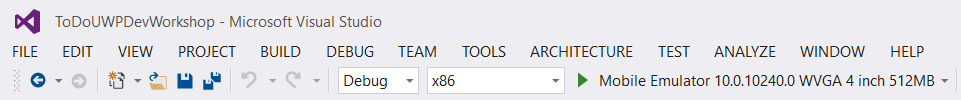


Figure 8

Configure your debug target to a Mobile emulator or a real device, if available.

1. Start debugging. The app will start on your Mobile device or emulator and show the same items you entered into the desktop client and which you stored in your mobile backend service database (which is running locally on your PC at present).

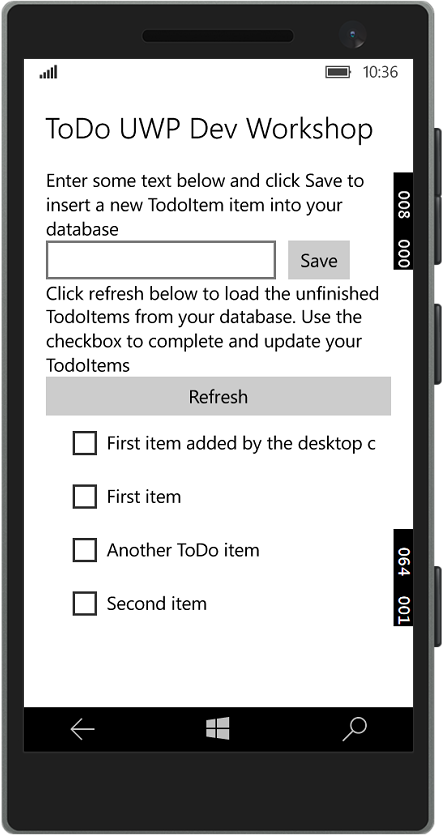


Figure 4

The client app running on a Windows 10 Mobile device uses the same backend service.

1. Stop debugging in both the client app project and also in the service project.
2. You have completed this exercise, and have run a mobile app over two different devices which shares data through a mobile backend service. You have created a *connected mobile experience* for the users of this app.

Exercise 2: Enable authentication for your app and connect to the Cloud

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

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 app project 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

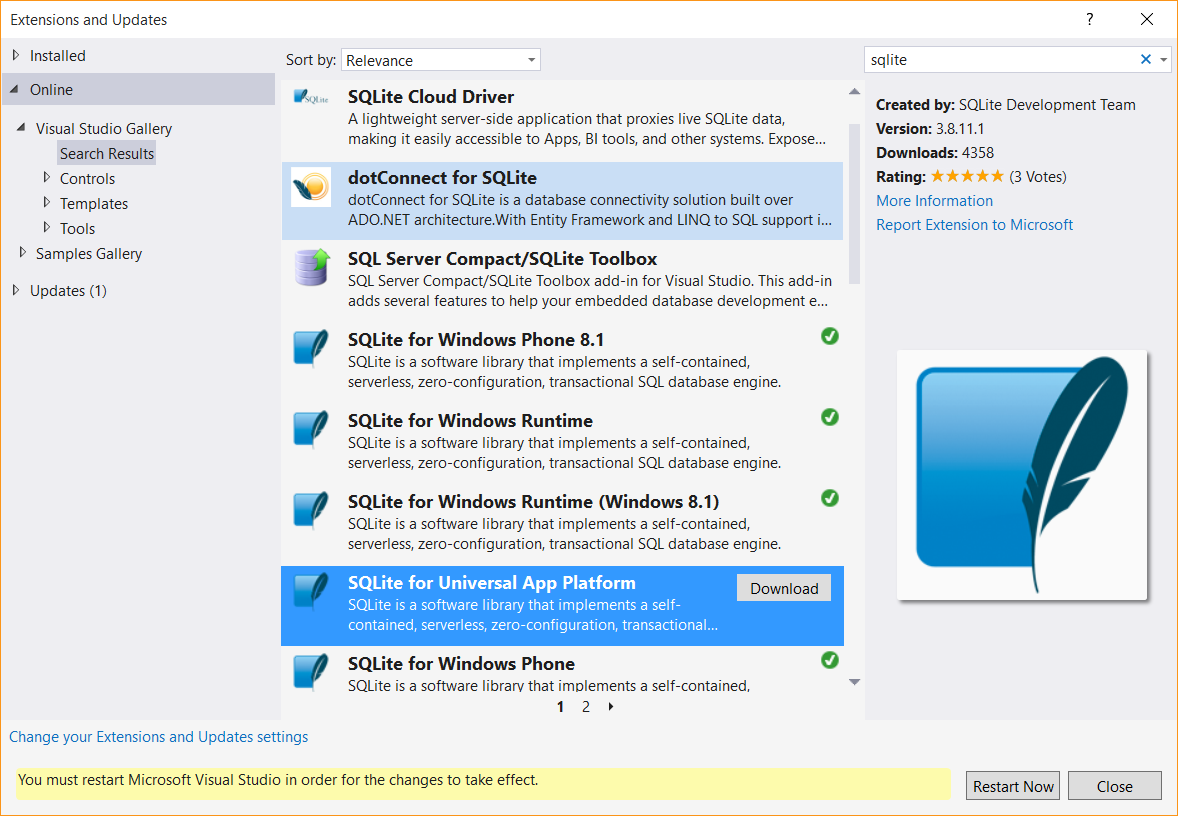


Figure 3

* + 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.

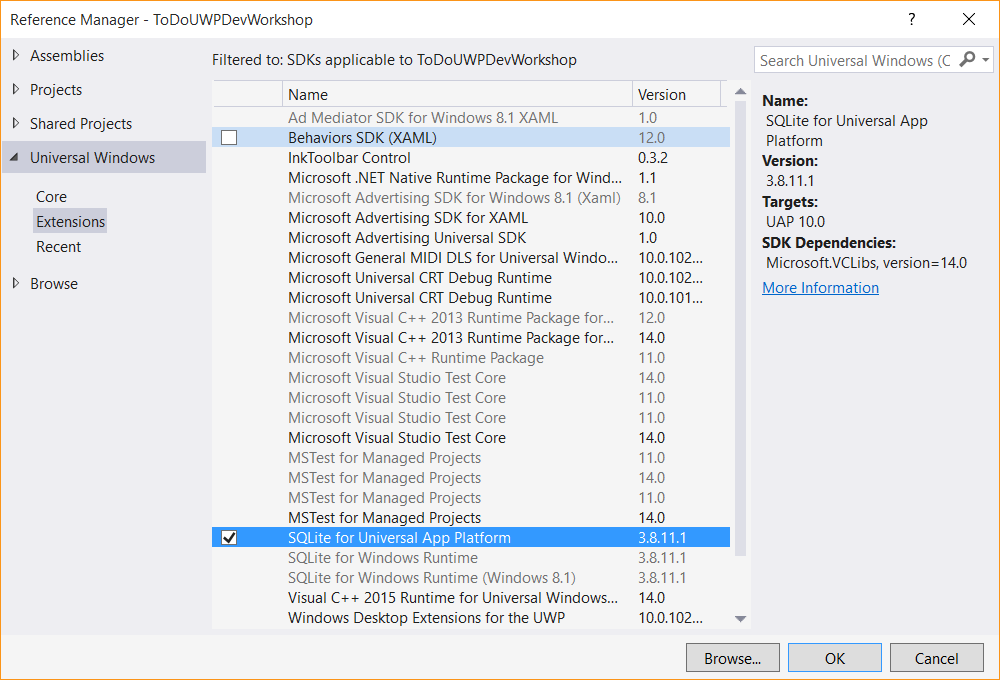


Figure 6

* + 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.