Using Windows Azure Mobile Services to Cloud-Enable Your Windows Store Apps in JavaScript

Windows Azure Developer Center

Step-by-Step

Microsoft

Using Windows Azure Mobile Services to Cloud-Enable your Windows Store Apps in JavaScript

Windows Azure Developer Center

Summary: This section shows you how to use Windows Azure Mobile Services and JavaScript to leverage data in a Windows Store app. In this tutorial, you will download an app that stores data in memory, create a new mobile service, integrate the mobile service with the app, and then login to the Windows Azure Management Portal to view changes to data made when running the app.

Category: Step-by-Step
Applies to: Windows Azure Mobile Services
Source: Windows Azure Developer Center (link to source content)
E-book publication date: January 2013

Microsoft

Contents

Introducing Windows Azure Mobile Services

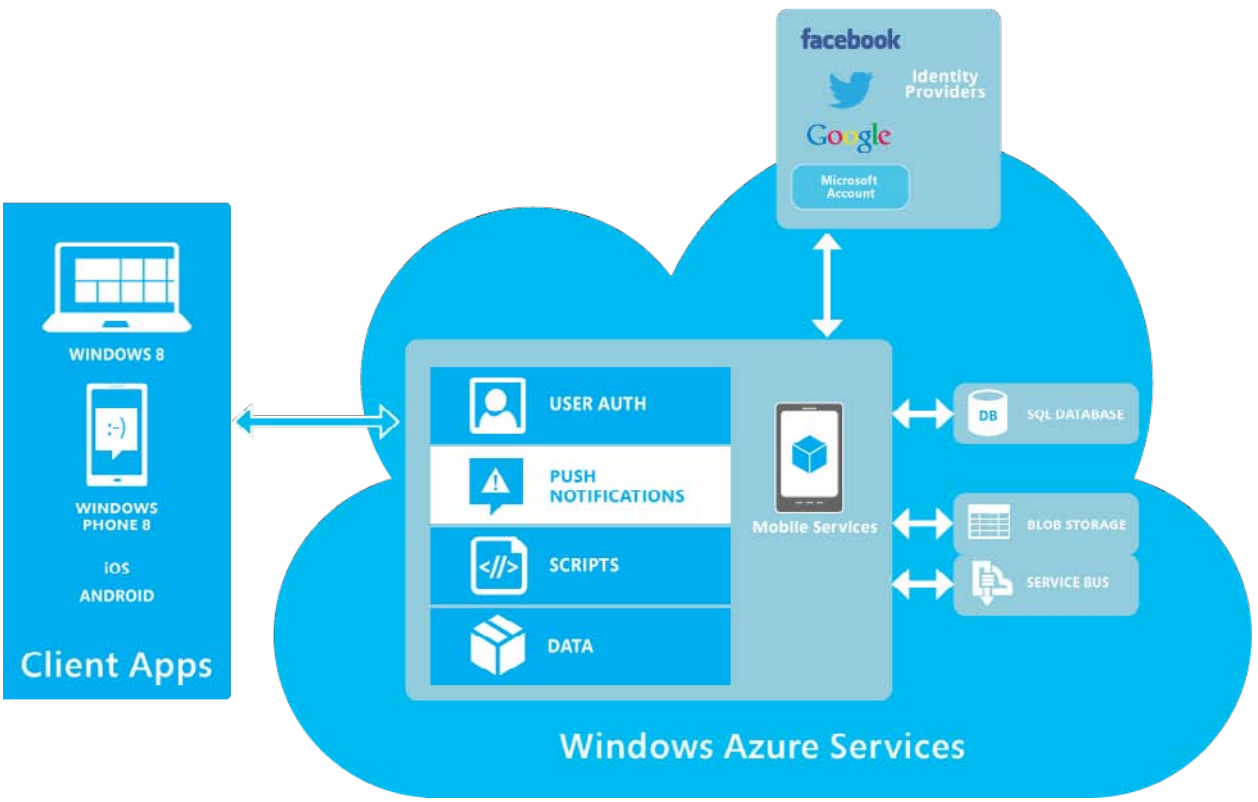
Windows Azure Mobile Services is a Windows Azure service offering designed to make it easy to create highly-functional mobile apps using Windows Azure. Mobile Services brings together a set of Windows Azure services that enable backend capabilities for your apps. Mobile Services provides the following backend capabilities in Windows Azure to support your apps:

- **Client libraries support mobile app development on various devices, including Windows 8, Windows Phone 8, iPhone, and iPad**:
  Like other Windows Azure service offerings, Mobile Services features a full set of REST APIs for data access and authentication so that you can leverage your mobile service from any HTTP compatible device. However, to make it easier for you to develop your apps, Mobile Services also provides client library support on most major device platforms so that you can interact with your mobile service by using a simplified client programming model that handles the HTTP messaging tasks for you.
- **Simple provisioning and management of tables for storing app data**:
  Mobile Services lets you store app data in SQL Database tables. By using the Windows Azure Management Portal, you easily create new tables as well as view and manage app data.
- **Integration with notification services to deliver push notifications to your app:**
  The ability to send real-time notifications to users has become a key functionality for device apps. Mobile Services integrates with platform-specific notification providers to enable you send notifications to your apps.
- **Integration with well-known identity providers for authentication:**
  Mobile Services makes it easy to add authentication to your apps. You can have your users log in with any of the major identity provider (Facebook, Twitter, Google, and Microsoft Account) and Mobile Services handles the authentication for you. Single sign-on is also supported by using Live Connect.
- **Granular control for authorizing access to tables:**
  Access to read, insert, update, and delete operations on tables can be restricted to various levels. This enables you to restrict table access to only authenticated users. Data can be further restricted based on the user ID of an authenticated user by using server scripts.
- **Supports scripts to inject business logic into data access operations:**
  The ability to execute your own business logic from the service-side is a key requirement of any backend solution. Mobile Services lets you register JavaScript code that is executed when specific insert, delete, update or read operations occur.
- **Integration with other cloud services:**
  Server scripts enable to integrate your mobile service with other backend services, such as Twilio,

SendMail, Twitter, Facebook, other Windows Azure services, and any other services accessible from HTTP requests.

- **Supports the ability to scale a mobile service instance:**
  When your app gets popular, Mobile Services lets you easily scale your backend solution by adding instances or increasing the size of the database.
- **Service monitoring and logging:**
  Mobile services provides a dashboard that gives you an at-a-glance assessment of your mobile services activity and it also lets you see logged errors and write to the logs from your own server scripts.

The following is a functional representation of the Mobile Services architecture:



The tutorials in this e-book show you how to perform most of the most important tasks in Mobile Services.

Get started with data in Mobile Services

This section shows you how to use Windows Azure Mobile Services to leverage data in a Windows Store app. In this tutorial, you will download an app that stores data in memory, create a new mobile service, integrate the mobile service with the app, and then login to the Windows Azure Management Portal to view changes to data made when running the app.

> **Note:** This tutorial is intended to help you better understand how Mobile Services enables you to use Windows Azure to store and retrieve data from a Windows Store app. As such, this topic walks you through many of the steps that are completed for you in the Mobile Services quickstart. If this is your first experience with Mobile Services, consider first completing the tutorial Get started with Mobile Services.

This tutorial walks you through these basic steps:

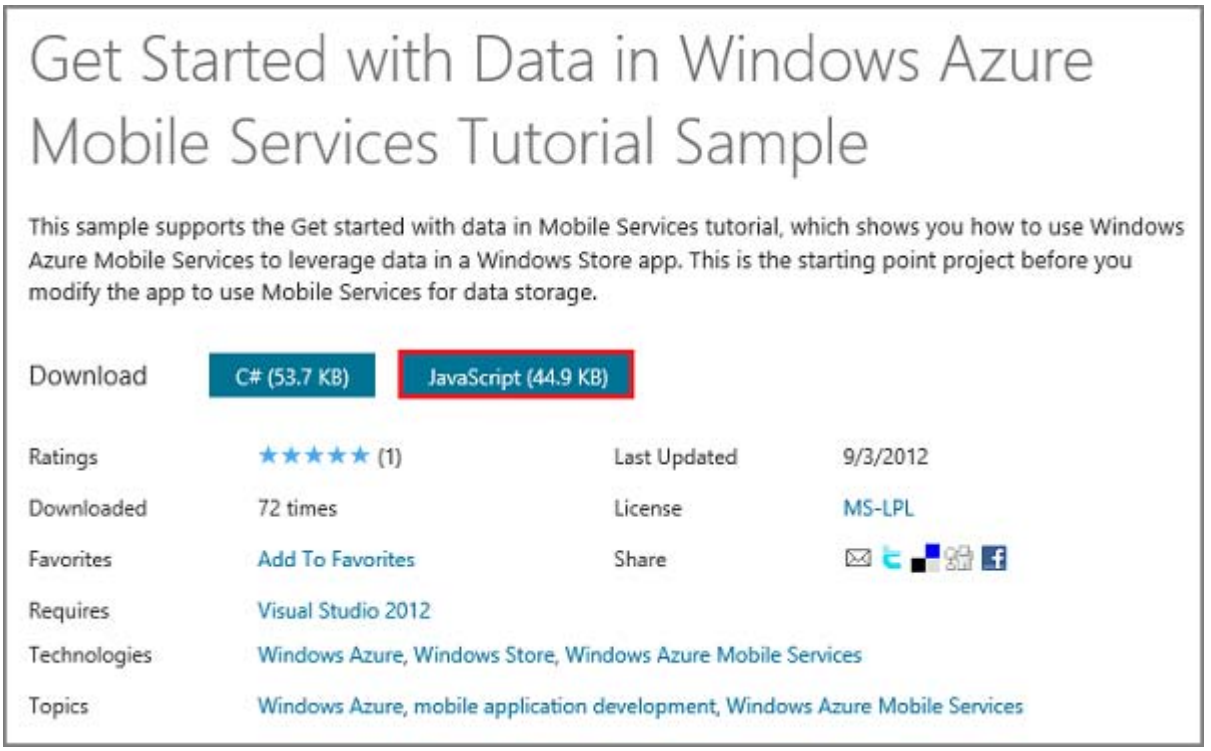
1. Download the Windows Store app project
2. Create the mobile service
3. Add a data table for storage
4. Update the app to use Mobile Services
5. Test the app against Mobile Services

This tutorial requires the Mobile Services SDK.

Download the GetStartedWithData project

This tutorial is built on the GetStartedWithData app, which is a Windows Store app. The UI for this app is identical to the app generated by the Mobile Services quickstart, except that added items are stored locally in memory.

1. Download the JavaScript version of the GetStartedWithData sample app from the Developer Code Samples site.

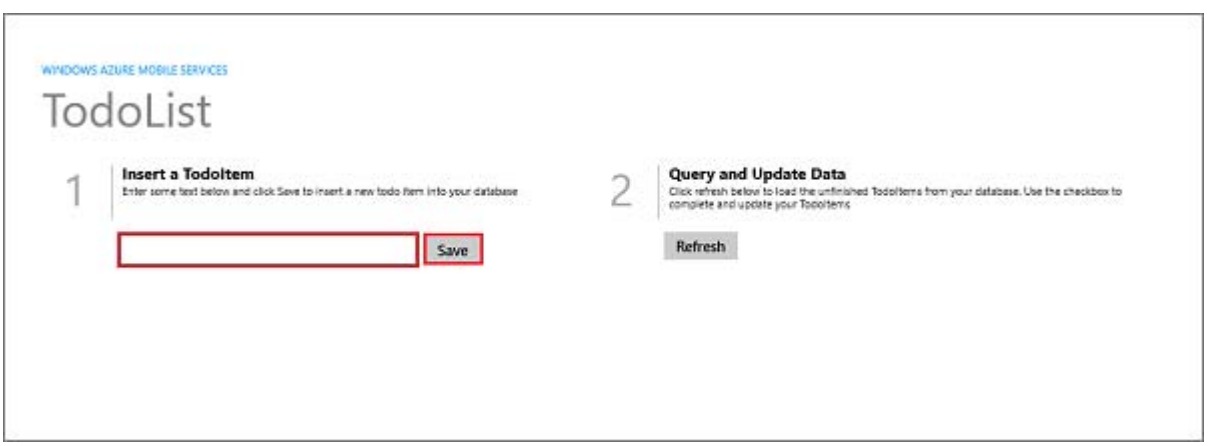


2. In Visual Studio 2012 Express for Windows 8, open the downloaded project, expand the **js** folder and examine the default.js file.

   Notice that added **TodoItem** objects are stored in an in-memory **List** object.

3. Press the **F5** key to rebuild the project and start the app.

4. In the app, type some text in **Insert a TodoItem**, then click **Save**.



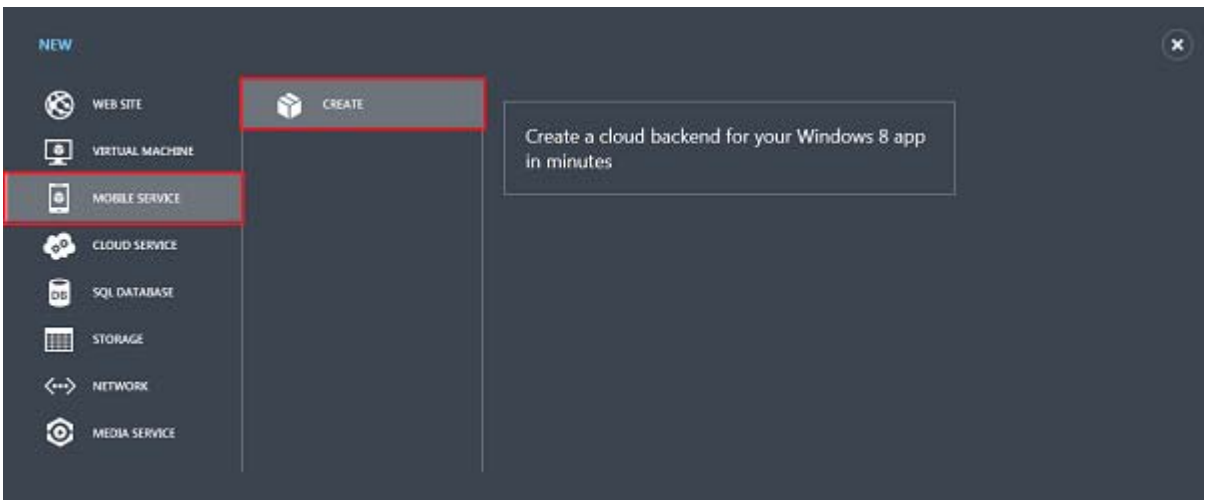
Notice that the saved text is displayed in the second column under **Query and update data**.

Create a new mobile service in the Management Portal

Next, you will create a new mobile service to replace the in-memory list for data storage. Follow these steps to create a new mobile service.

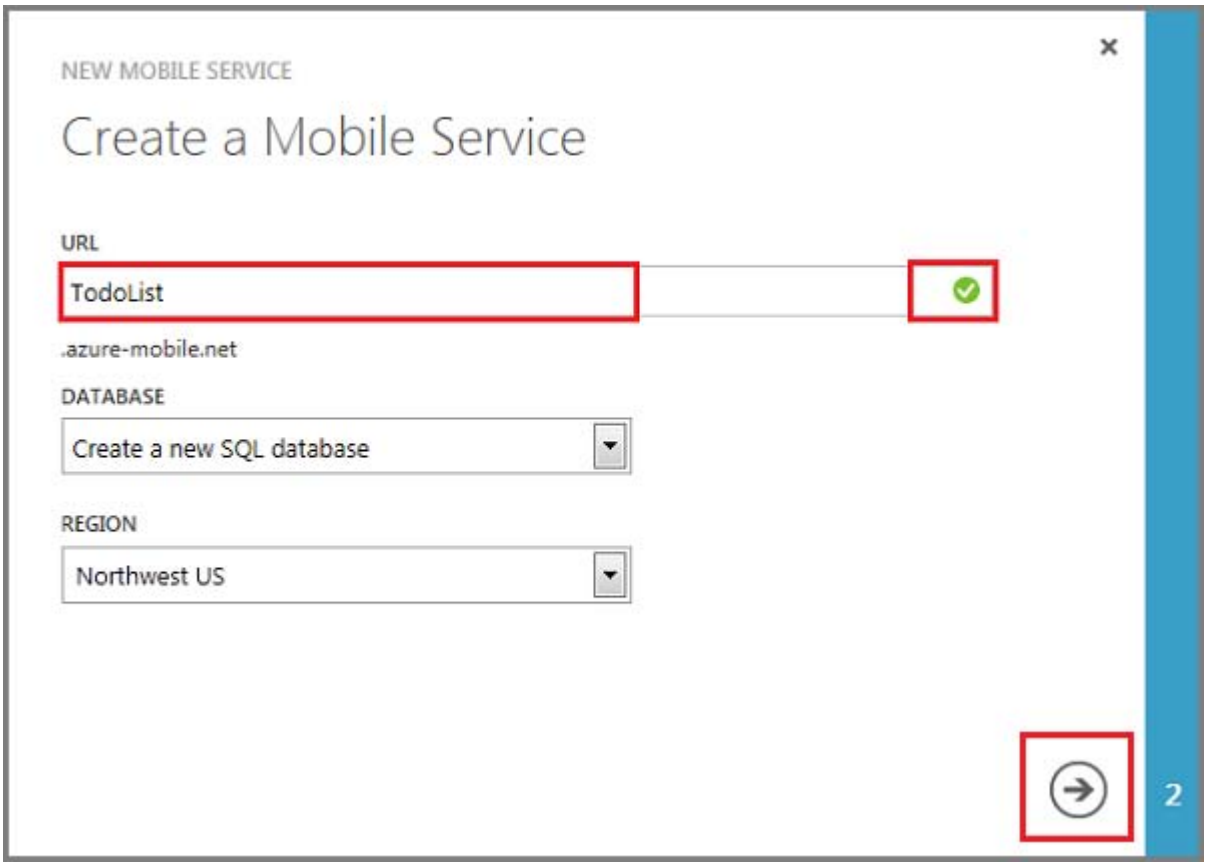
1. Log into the Windows Azure Management Portal.
2. At the bottom of the navigation pane, click **+NEW**.

3. Expand **Compute** and **Mobile Service**, then click **Create**.



This displays the **New Mobile Service** dialog.

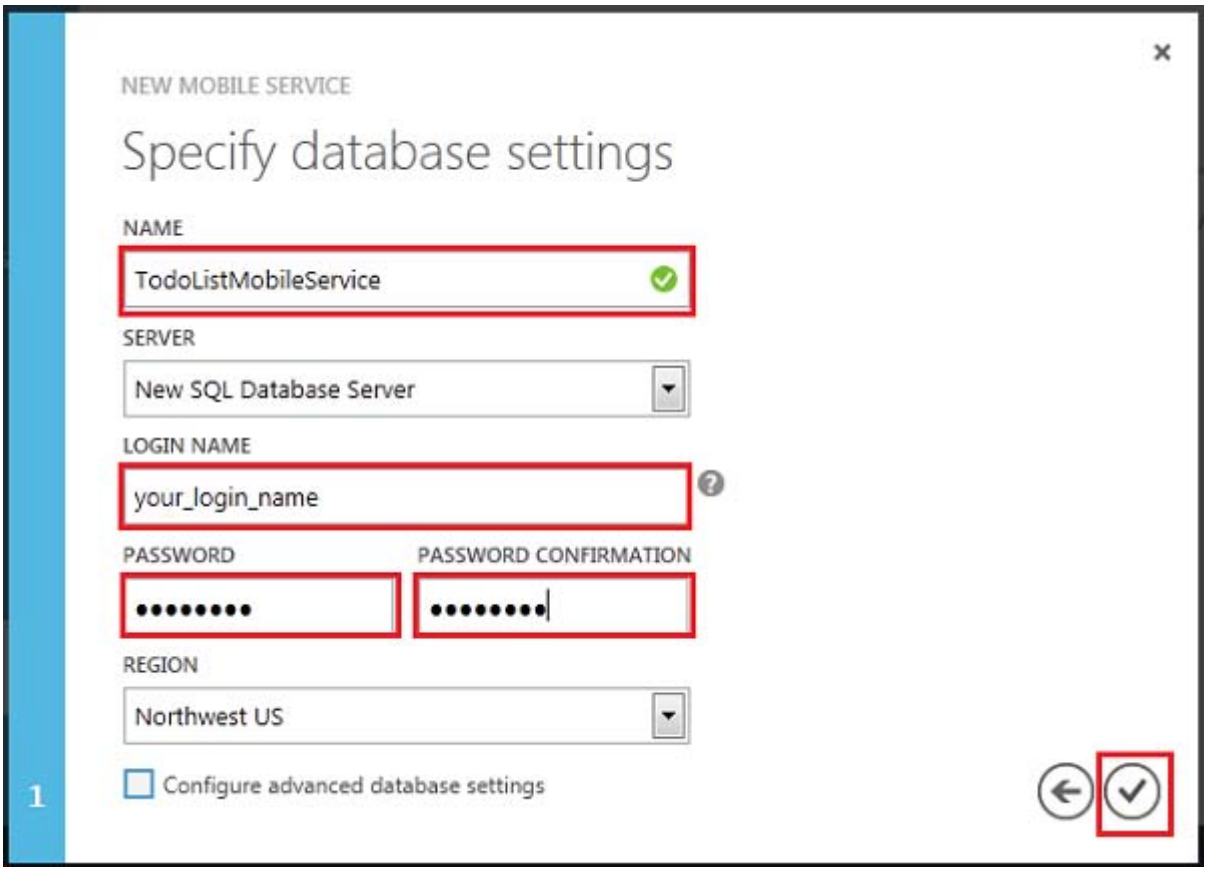
4. In the **Create a mobile service** page, type a subdomain name for the new mobile service in the **URL** textbox and wait for name verification. Once name verification completes, click the right arrow button to go to the next page.



This displays the **Specify database settings** page.

> **Note:** As part of this tutorial, you create a new SQL Database instance and server. You can reuse this new database and administer it as you would any other SQL Database instance. If you already have a database in the same region as the new mobile service, you can instead choose **Use existing Database** and then select that database. The use of a database in a different region is not recommended because of additional bandwidth costs and higher latencies.

5. In **Name**, type the name of the new database, then type **Login name**, which is the administrator login name for the new SQL Database server, type and confirm the password, and click the check button to complete the process.



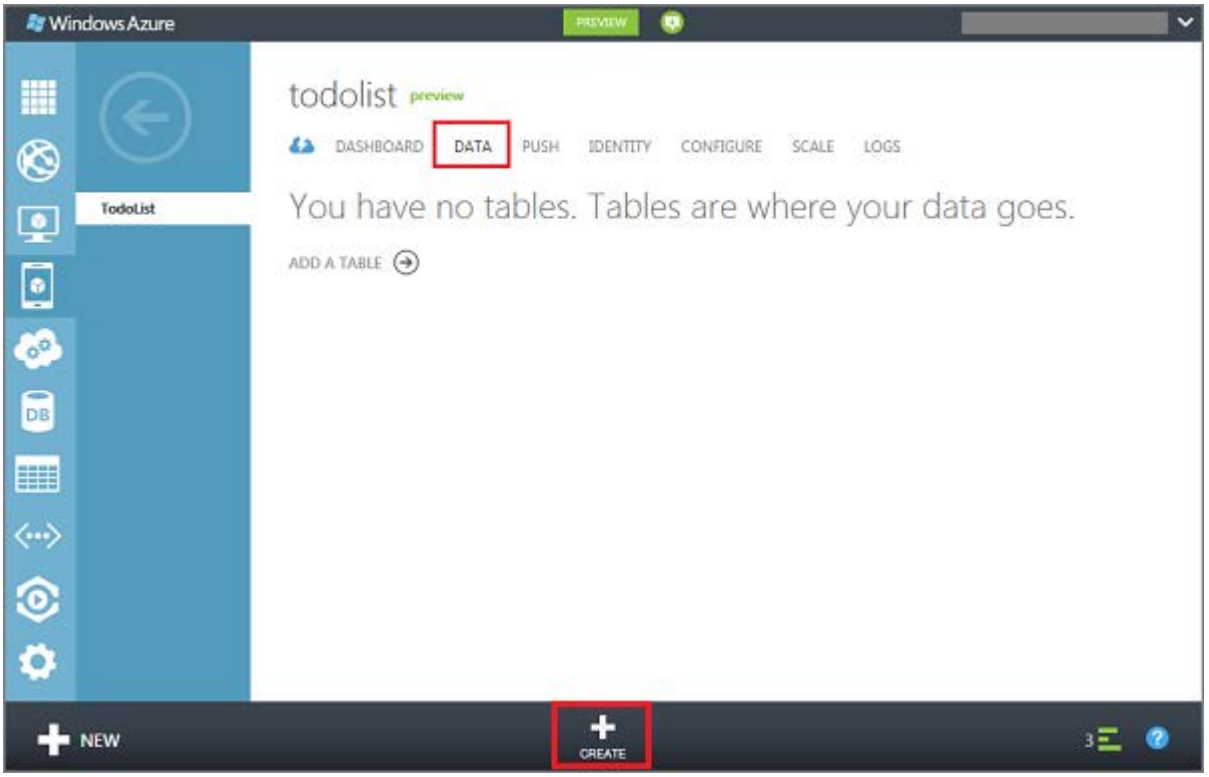
> **Note:** When the password that you supply does not meet the minimum requirements or when there is a mismatch, a warning is displayed.
> We recommend that you make a note of the administrator login name and password that you specify; you will need this information to reuse the SQL Database instance or the server in the future.

You have now created a new mobile service that can be used by your mobile apps. Next, you will add a new table in which to store app data. This table will be used by the app in place of the in-memory collection.

Add a new table to the mobile service

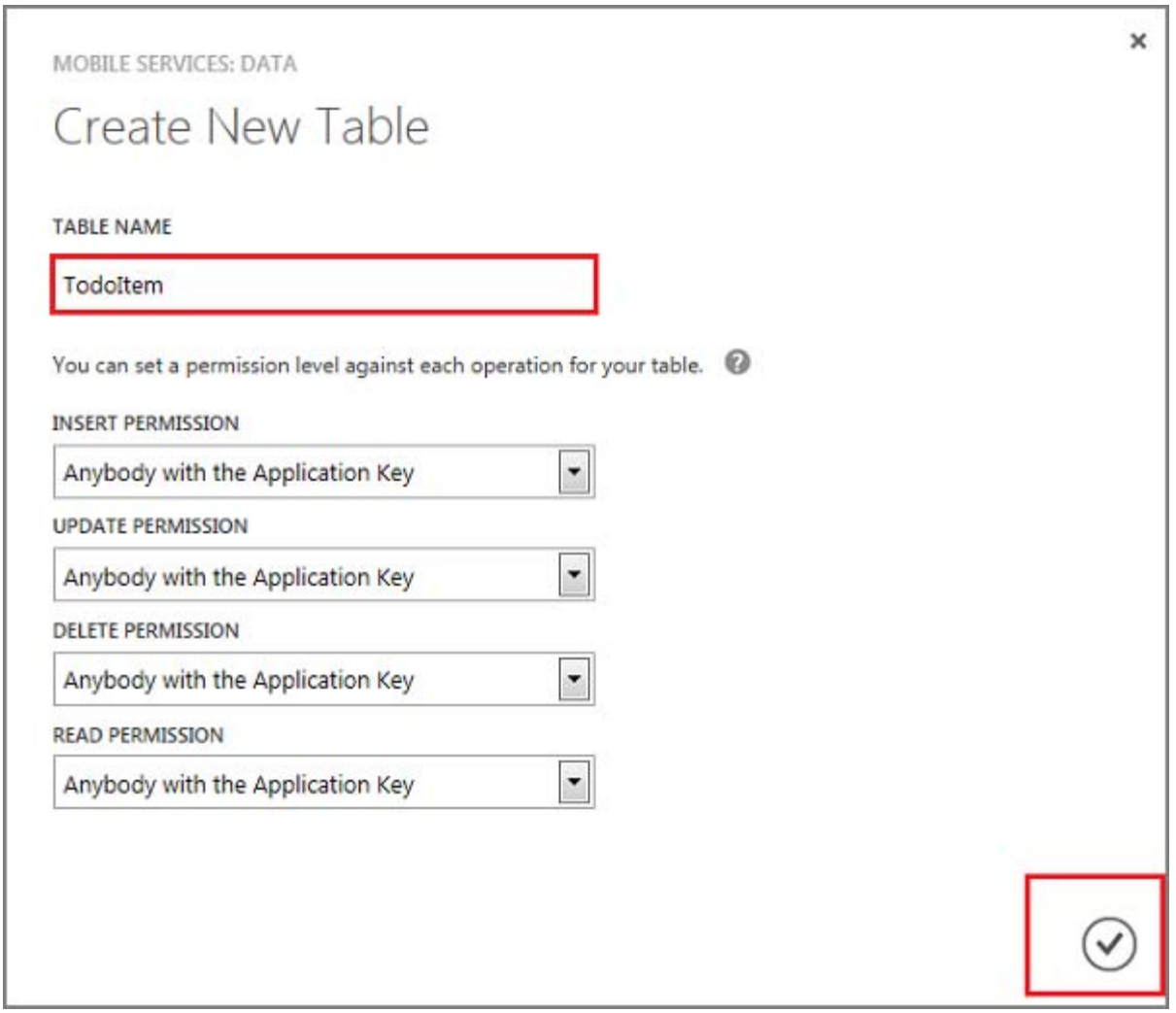
To be able to store app data in the new mobile service, you must first create a new table in the associated SQL Database instance.

1. In the Management Portal, click **Mobile Services**, and then click the mobile service that you just created.
2. Click the **Data** tab, then click **+Create**.



This displays the **Create new table** dialog.

3. In **Table name** type *TodoItem*, then click the check button.



This creates a new storage table **TodoItem** with the default permissions set, which means that any user of the app can access and change data in the table.

> **Note:** The same table name is used in Mobile Services quickstart. However, each table is created in a schema that is specific to a given mobile service. This is to prevent data collisions when multiple mobile services use the same database.

4. Click the new **TodoItem** table and verify that there are no data rows.

5. Click the **Columns** tab and verify that there is only a single **id** column, which is automatically created for you.

   This is the minimum requirement for a table in Mobile Services.

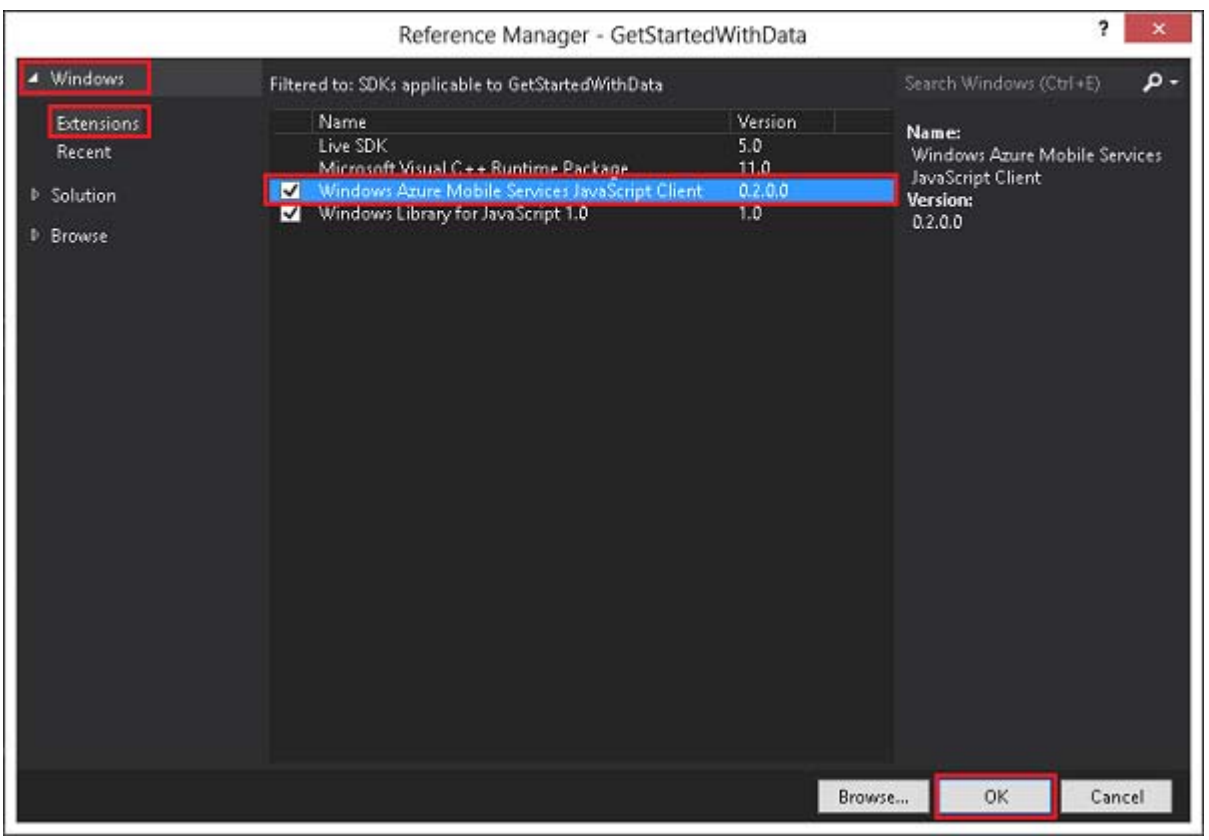
> **Note:** When dynamic schema is enabled on your mobile service, new columns are created automatically when JSON objects are sent to the mobile service by an insert or update operation.

You are now ready to use the new mobile service as data storage for the app.

Update the app to use the mobile service for data access

Now that your mobile service is ready, you can update the app to store items in Mobile Services instead of the local collection.

1. If you haven't already installed the Mobile Services SDK, install it now.
2. In the **Project** menu in Visual Studio, click **Add Reference**, then expand **Windows**, click **Extensions**, check **Windows Azure Mobile Services JavaScript Client**, and click **OK**.



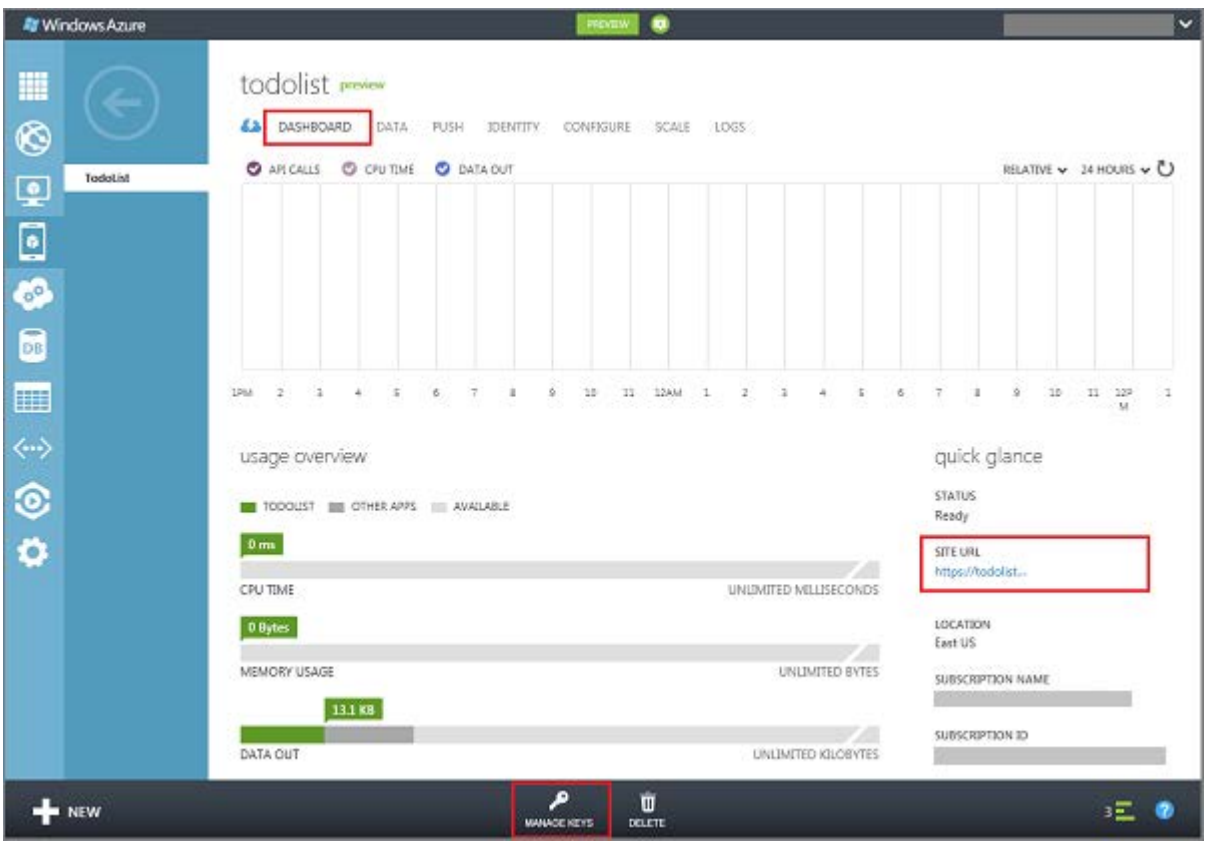
This adds a reference to the Mobile Services client to the project.

3. In the default.html project file, add the following script reference in the page header:

```
<script type="text/javascript"
src="/MobileServicesJavaScriptClient/MobileServices.js"></script>
```

4. In the Management Portal, click **Mobile Services**, and then click the mobile service you just created.

5. Click the **Dashboard** tab and make a note of the **Site URL**, then click **Manage keys** and make a note of the **Application key**.



You will need these values when accessing the mobile service from your app code.

6. In Visual Studio, open the file default.js, uncomment the code that defines the **mobileService** variable, and supply the URL and application key from the mobile service in the **MobileServiceClient** constructor, in that order.

   This creates a new instance of MobileServiceClient that is used to access your mobile service.

7. Uncomment the following line of code:

```
var todoTable = mobileService.getTable('TodoItem');
```

   This code creates a proxy object (**todoTable**) for the SQL Database **TodoItem**.

8. Replace the **InsertTodoItem** method with the following code:

```
var insertTodoItem = function (todoItem) {
    // Inserts a new row into the database. When the operation completes
    // and Mobile Services has assigned an id, the item is added to the
binding list.
    todoTable.insert(todoItem).done(function (item) {
        todoItems.push(item);
    });
};
```

This code inserts a new item into the table.

9. Replace the **RefreshTodoItems** method with the following code:

```
var refreshTodoItems = function () {
    // This code refreshes the entries in the list by querying the
TodoItems table.
    todoTable.read().done(function (results) {
        todoItems = new WinJS.Binding.List(results);
        listItems.winControl.itemDataSource = todoItems.dataSource;
    });
};
```

This sets the binding to the collection of items in the todoTable, which contains all completed items returned from the mobile service.

10. Replace the **UpdateCheckedTodoItem** method with the following code:

```
var updateCheckedTodoItem = function (todoItem) {
    // This code takes a freshly completed TodoItem and updates the
database.
    todoTable.update(todoItem);
};
```

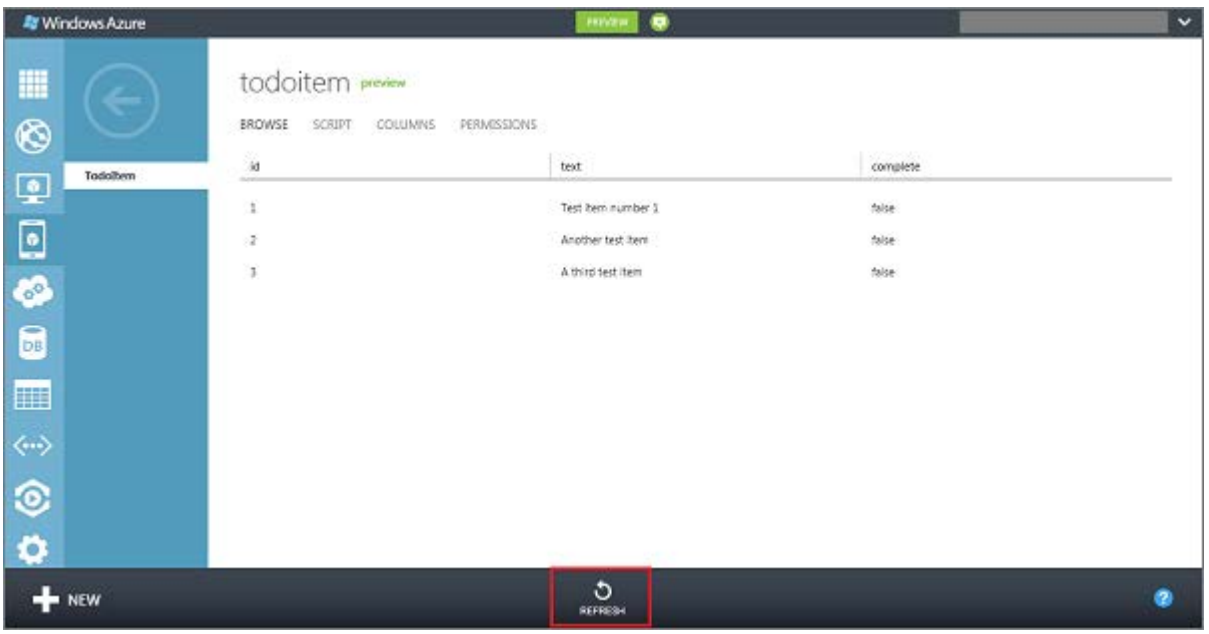
This sends an item update to the mobile service.

Now that the app has been updated to use Mobile Services for backend storage, it's time to test the app against Mobile Services.

Test the app against your new mobile service

1. In Visual Studio, press the F5 key to run the app.
2. As before, type text in **Insert a TodoItem**, and then click **Save**.

   This sends a new item as an insert to the mobile service.
3. In the Management Portal, click **Mobile Services**, and then click your mobile service.
4. Click the **Data** tab, then click **Browse**.



   Notice that the **TodoItem** table now contains data, with id values generated by Mobile Services, and that columns have been automatically added to the table to match the TodoItem class in the app.
5. In the app, check one of the items in the list, then go back to the Browse tab in the portal and click **Refresh**.

   Notice that the complete value has changed from **false** to **true**.
6. In the default.js project file, replace the existing **RefreshTodoItems** method with the following code that filters out completed items:

```
var refreshTodoItems = function () {
    // More advanced query that filters out completed items.
    todoTable.where({ complete: false })
        .read()
        .done(function (results) {
```

```
            todoItems = new WinJS.Binding.List(results);
            listItems.winControl.itemDataSource = todoItems.dataSource;
        });
};
```

7. In the app, check another one of the items in the list and then click the **Refresh** button.

   Notice that the checked item now disappears from the list. Each refresh results in a round-trip to the mobile service, which now returns filtered data.

   This concludes the **Get started with data** section. Next, you will learn more about using server scripts in Mobile Services to validate and change data sent from your app and also how to use paging in queries to control the amount of data handled in a single request.

Validate and modify data in Mobile Services by using server scripts

This section shows you how to leverage server scripts in Windows Azure Mobile Services. Server scripts are registered in a mobile service and can be used to perform a wide range of operations on data being inserted and updated, including validation and data modification. In this tutorial, you will define and register server scripts that validate and modify data. Because the behavior of server side scripts often affects the client, you will also update your Windows Store app to take advantage of these new behaviors.

This tutorial walks you through these basic steps:

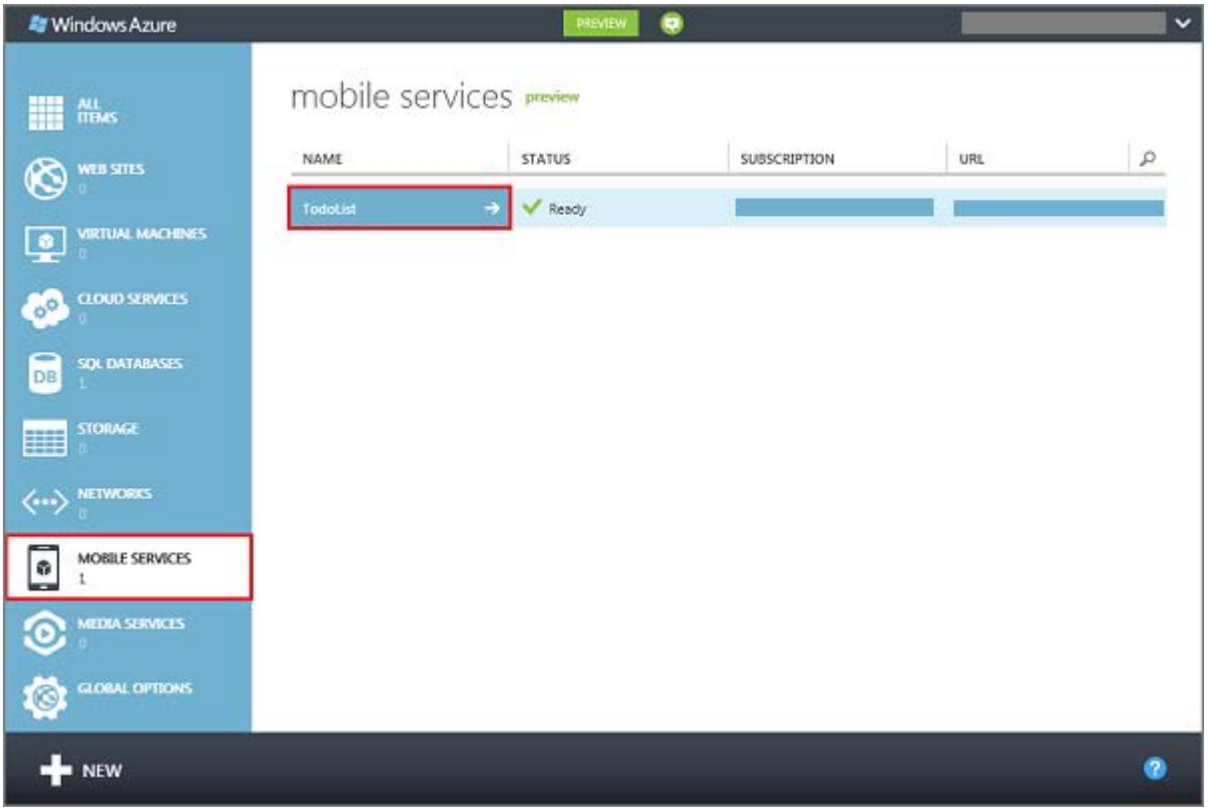
1. Add string length validation
2. Update the client to support validation
3. Add a timestamp on insert
4. Update the client to display the timestamp

This tutorial builds on the steps and the sample app from the previous section Get started with data. Before you begin this tutorial, you must first complete Get started with data.

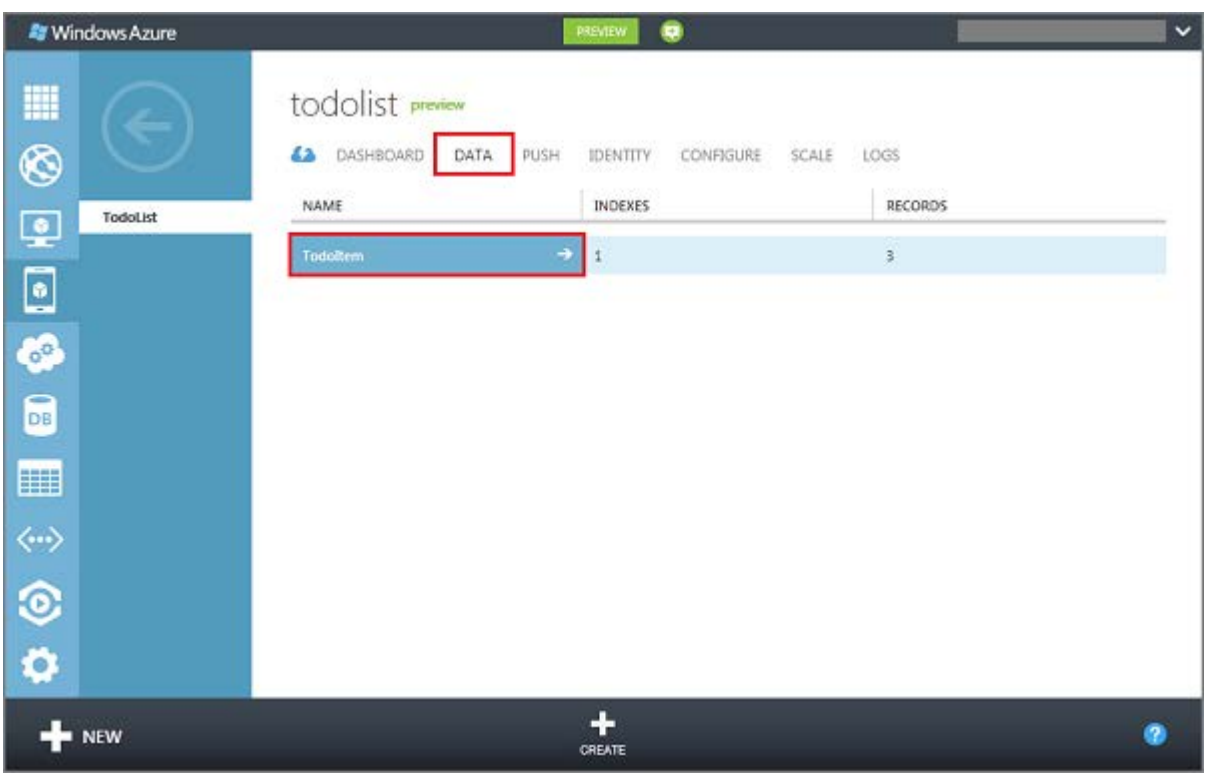
Add validation

It is always a good practice to validate the length of data that is submitted by users. First, you register a script that validates the length of string data sent to the mobile service and rejects strings that are too long, in this case longer than 10 characters.

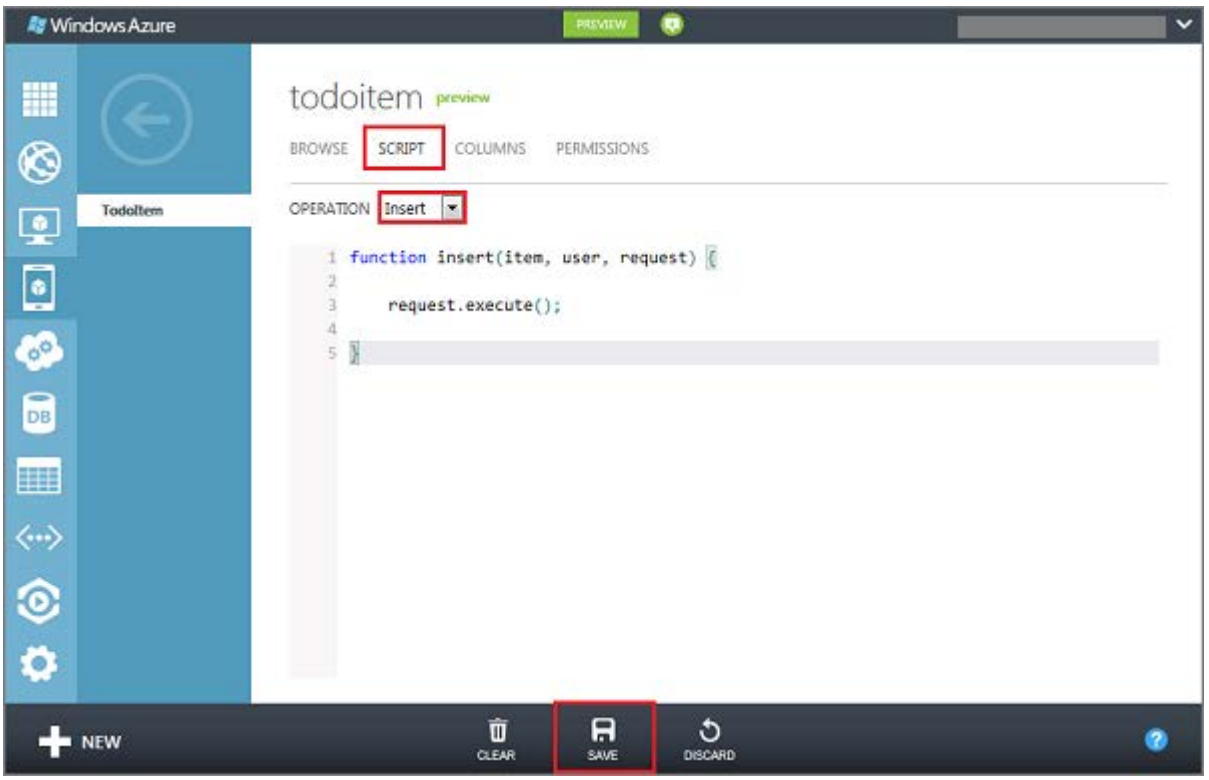
1. Log into the Windows Azure Management Portal, click **Mobile Services**, and then click your app.



2. Click the **Data** tab, then click the **TodoItem** table.



3. Click **Script**, then select the **Insert** operation.



4. Replace the existing script with the following function, and then click **Save**.

```
function insert(item, user, request) {
    if (item.text.length > 10) {
        request.respond(statusCodes.BAD_REQUEST, 'Text length must be
under 10');
    } else {
        request.execute();
    }
}
```

This script checks the length of the **TodoItem.text** property and sends an error response when the length exceeds 10 characters. Otherwise, the **execute** method is called to complete the insert.

> **Note:** You can remove a registered script on the **Script** tab by clicking **Clear** and then **Save**.

Update the client

Now that the mobile service is validating data and sending error responses, you need to update your app to be able to handle error responses from validation.

1. In Visual Studio 2012 Express for Windows 8, open the project that you modified when you completed the tutorial Get started with data.

2. Press the **F5** key to run the app, then type text longer than 10 characters in **Insert a TodoItem** and click **Save**.

   Notice that the app raises an unhandled error as a result of the 400 response (Bad Request) returned by the mobile service.

3. Open the file default.js, then replace the existing **InsertTodoItem** method with the following:

```
var insertTodoItem = function (todoItem) {
    // Inserts a new row into the database. When the operation completes
    // and Mobile Services has assigned an id, the item is added to the
binding list.
    todoTable.insert(todoItem).done(function (item) {
        todoItems.push(item);
    }, function (error) {
        // Create the error message dialog and set its content to the
error
        // message contained in the response.
        var msg = new Windows.UI.Popups.MessageDialog(
            error.request.responseText);
        msg.showAsync();
    });
};
```

   This version of the method includes error handling that displays the error response in a dialog.

Add a timestamp

The previous tasks validated an insert and either accepted or rejected it. Now, you will update inserted data by using a server script that adds a timestamp property to the object before it gets inserted.

1. In the **Scripts** tab in the Management Portal, replace the current **Insert** script with the following function, and then click **Save**.

```
function insert(item, user, request) {
    if (item.text.length > 10) {
        request.respond(statusCodes.BAD_REQUEST, 'Text length must be
under 10');
```

```
    } else {
        item.createdAt = new Date();
        request.execute();
    }
}
```

This function augments the previous insert script by adding a new **createdAt** timestamp property to the object before it gets inserted by the call to **request**.**execute**.

> **Note:** Dynamic schema must be enabled the first time that this insert script runs. With dynamic schema enabled, Mobile Services automatically adds the **createdAt** column to the **TodoItem** table on the first execution. Dynamic schema is enabled by default for a new mobile service, and it should be disabled before the app is published to the Windows Store.

2. In Visual Studio, press the **F5** key to run the app, then type text (shorter than 10 characters) in **Insert a TodoItem** and click **Save**.

   Notice that the new timestamp does not appear in the app UI.

3. Back in the Management Portal, click the **Browse** tab in the **todoitem** table.

   Notice that there is now a **createdAt** column, and the new inserted item has a timestamp value.

   Next, you need to update the Windows Store app to display this new column.

Update the client again

The Mobile Service client will ignore any data in a response that it cannot serialize into properties on the defined type. The final step is to update the client to display this new data.

1. In Visual Studio, open the file default.html, then add the following HTML element in the TemplateItem grid:

```
<div style="-ms-grid-column: 4; -ms-grid-row-align: center; margin-left:
5px"
    data-win-bind="innerText: createdAt"></div>
```

   This displays the new **createdAt** property.

2. Press the **F5** key to run the app.

   Notice that the timestamp is only displayed for items inserted after you updated the insert script.

3. In the default.js file, replace the existing **RefreshTodoItems** method with the following code:

```
var refreshTodoItems = function () {
    // More advanced query that filters out completed items.
    todoTable.where(function () {
        return (this.complete === false && this.createdAt !== null);
    })
    .read()
    .done(function (results) {
        todoItems = new WinJS.Binding.List(results);
        listItems.winControl.itemDataSource = todoItems.dataSource;
    });
};
```

This method updates the query to also filter out items that do not have a timestamp value.

4. Press the **F5** key to run the app.

Notice that all items created without timestamp value disappear from the UI.

You have completed this working with data tutorial. Next you will learn how to refine queries with paging.

Refine Mobile Services queries with paging

This section shows you how to use paging to manage the amount of data returned to your Windows Store app from Windows Azure Mobile Services. In this tutorial, you will use the **Take** and **Skip** query methods on the client to request specific "pages" of data.

> **Note:** To prevent data overflow in mobile device clients, Mobile Services implements an automatic page limit, which defaults to a maximum of 50 items in a response. By specifying the page size, you can explicitly request up to 1,000 items in the response.

This tutorial builds on the steps and the sample app from the previous tutorial Get started with data. Before you begin this tutorial, you must complete at least the first tutorial in the working with data series—Get started with data.

1. In Visual Studio 2012 Express for Windows 8, open the project that you modified when you completed the tutorial Get started with data.

2. Press the **F5** key to run the app, then type text in **Insert a TodoItem** and click **Save**.

3. Repeat the previous step at least three times, so that you have more than three items stored in the TodoItem table.

4. In the default.js file, replace the **RefreshTodoItems** method with the following code:

```
var refreshTodoItems = function () {
    // Define a filtered query that returns the top 3 items.
    todoTable.where({ complete: false })
        .take(3)
        .read()
        .done(function (results) {
            todoItems = new WinJS.Binding.List(results);
            listItems.winControl.itemDataSource = todoItems.dataSource;
        });
};
```

   This query, when executed during data binding, returns the top three items that are not marked as completed.

5. Press the **F5** key to run the app.

   Notice that only the first three results from the TodoItem table are displayed.

6. (Optional) View the URI of the request sent to the mobile service by using message inspection software, such as browser developer tools or Fiddler.

   Notice that the **take(3)** method was translated into the query option **$top=3** in the query URI.

7. Update the **RefreshTodoItems** method once more with the following code:

```
var refreshTodoItems = function () {
    // Define a filtered query that skips the first 3 items and
    // then returns the next 3 items.
    todoTable.where({ complete: false })
        .skip(3)
        .take(3)
        .read()
        .done(function (results) {
            todoItems = new WinJS.Binding.List(results);
            listItems.winControl.itemDataSource = todoItems.dataSource;
```

```
        });
};
```

This query skips the first three results and returns the next three after that. This is effectively the second "page" of data, where the page size is three items.

> **Note:** This tutorial uses a simplified scenario by passing hard-coded paging values to the **Take** and **Skip** methods. In a real-world app, you can use queries similar to the above with a pager control or comparable UI to let users navigate to previous and next pages. You can also call the **includeTotalCount** method to get the total count of items available on the server, along with the paged data.

8. (Optional) Again view the URI of the request sent to the mobile service.

   Notice that the **skip(3)** method was translated into the query option **$skip=3** in the query URI.

Get started with authentication in Mobile Services

This section shows you how to authenticate users in Windows Azure Mobile Services from your app. In this tutorial, you add authentication to the quickstart project using an identity provider that is supported by Mobile Services. After being successfully authenticated and authorized by Mobile Services, the user ID value is displayed.

This tutorial walks you through these basic steps to enable authentication in your app:

1. Register your app for authentication and configure Mobile Services
2. Restrict table permissions to authenticated users
3. Add authentication to the app

This tutorial is based on the Mobile Services quickstart. You must also first complete the tutorial Get started with data in Mobile Services.

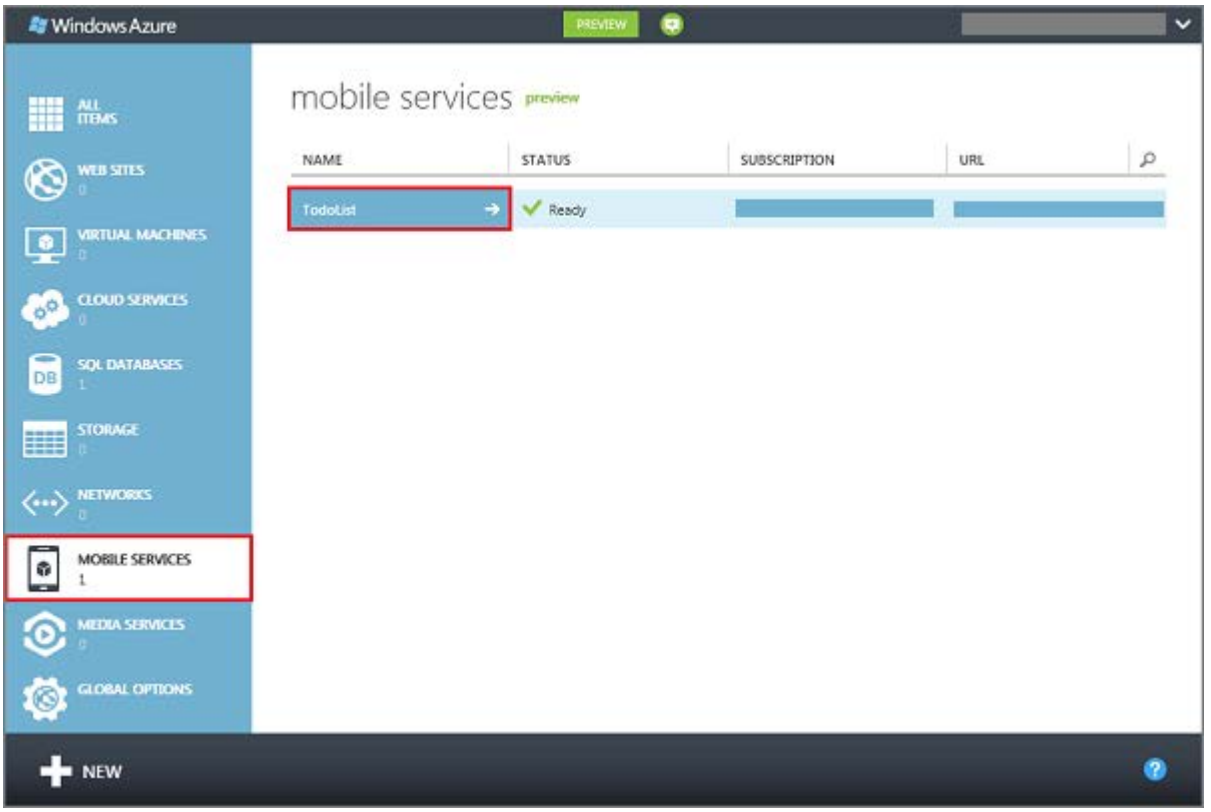
> **Note:** This tutorial demonstrates the basic method provided by Mobile Services to authenticate users by using a variety of identity providers. This method is easy to configure and supports multiple providers. However, this method also requires users to log-in every time your app starts. To instead use Live Connect to provide a single sign-on experience in your Windows Store app, see the later section Single sign-on for Windows Store apps by using Live Connect.

Register your app for authentication and configure Mobile Services

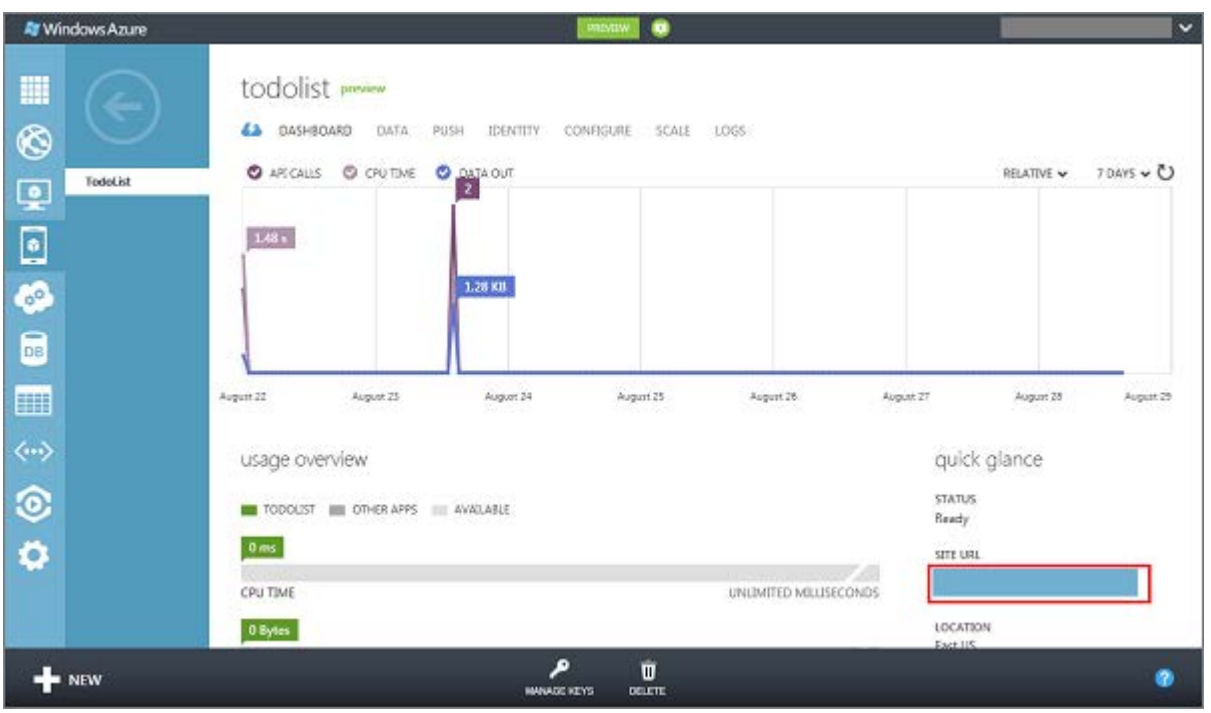
To be able to authenticate users, you must register your app with an identity provider. You must then register the provider-generated client secret with Mobile Services.

> **Note:** This section shows how to register your app to use Facebook as the identity provider. See the Appendix for the steps required to register your app with other identity providers, including Twitter, Microsoft Account, and Google.

1. Log on to the Windows Azure Management Portal, click **Mobile Services**, and then click your mobile service.



2. Click the **Dashboard** tab and make a note of the **Site URL** value.

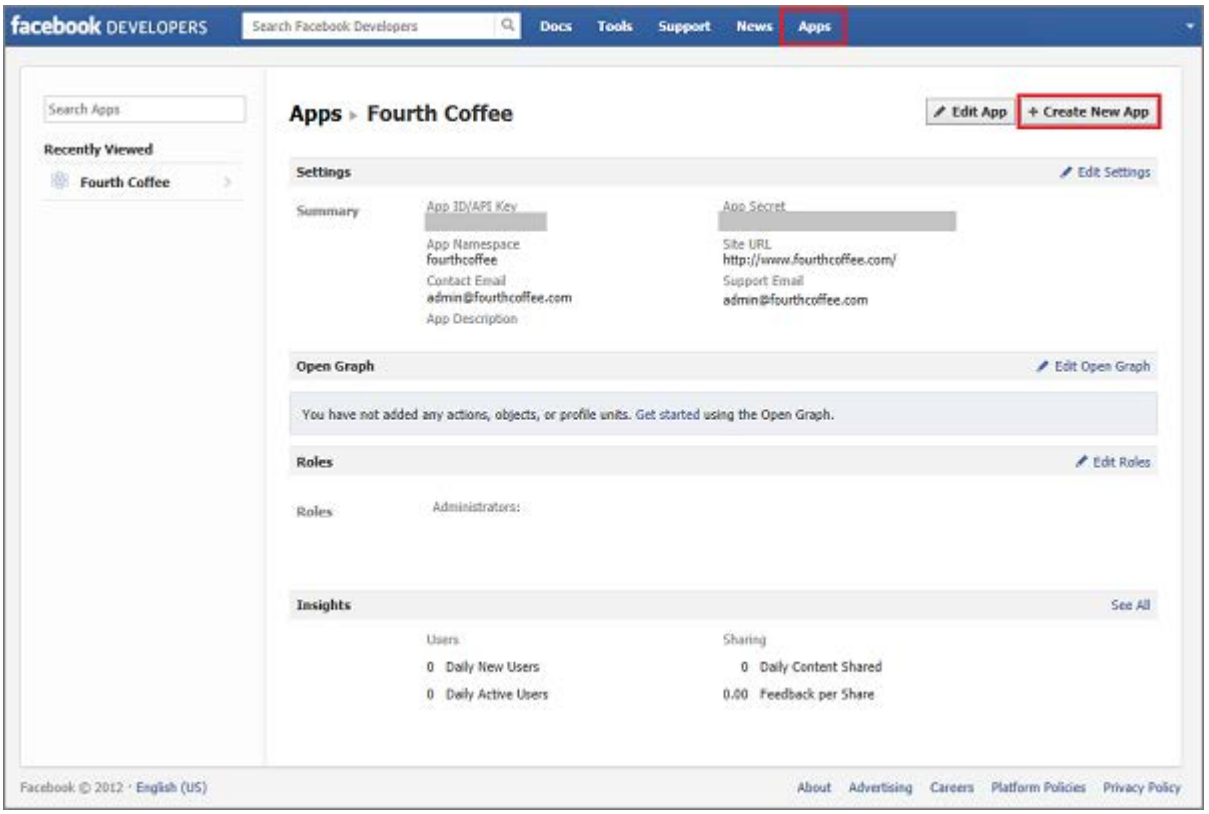


You may need to provide this value to the identity provider when you register your app.

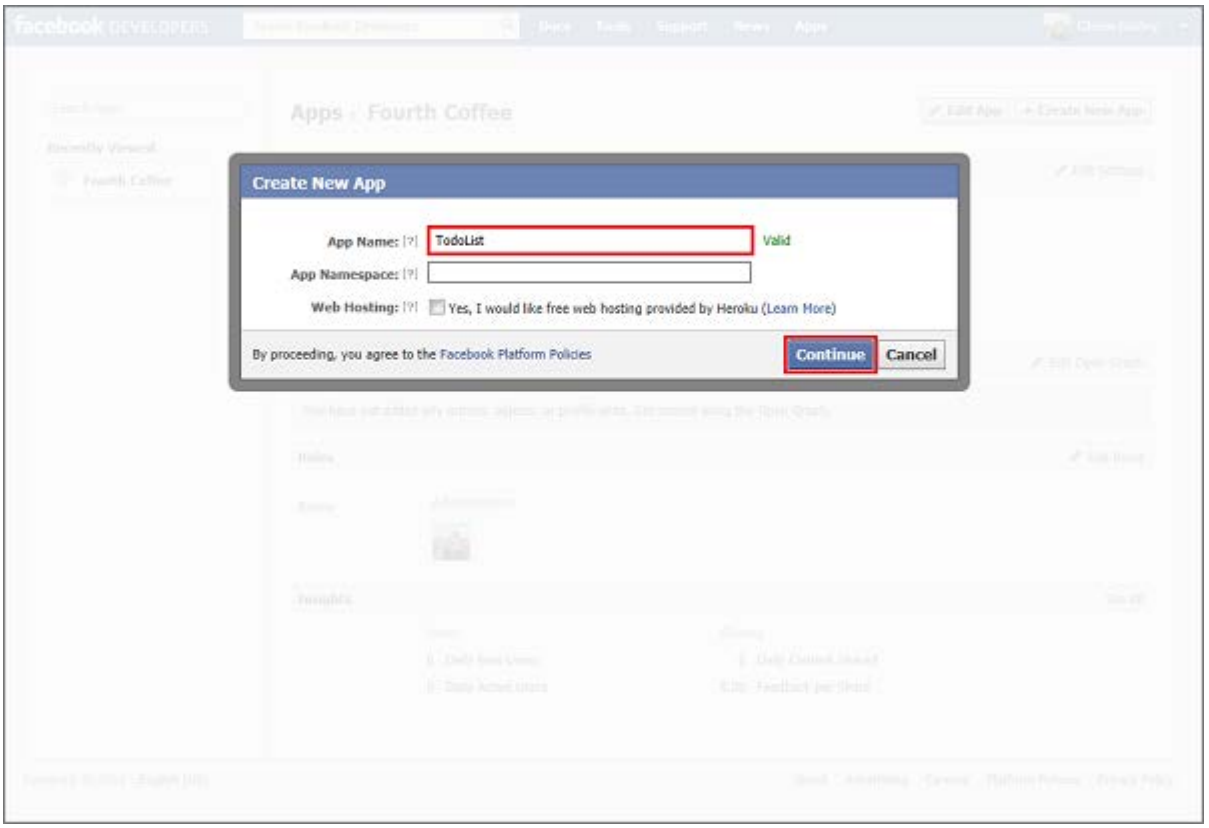
Note: To complete the procedure in this topic, you must have a Facebook account that has a verified email address and a mobile phone number. To create a new Facebook account, go to facebook.com.

3. Navigate to the Facebook Developers web site and sign-in with your Facebook account credentials.
4. (Optional) If you have not already registered, click **Register Now** button, accept the policy, provide any and then click **Done**.

5. Click **Apps**, then click **Create New App**.

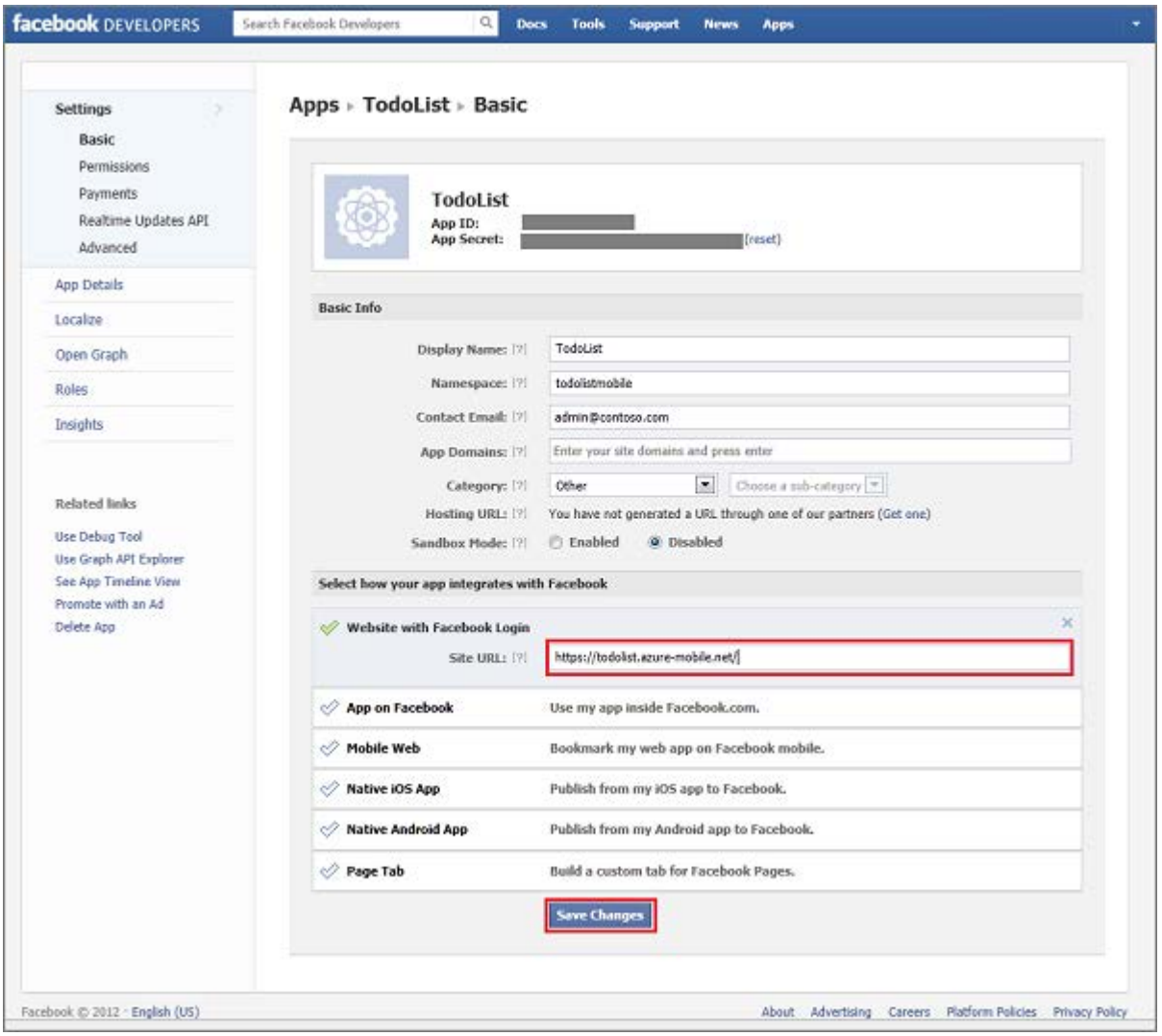


6. Choose a unique name for your app, select **OK**.

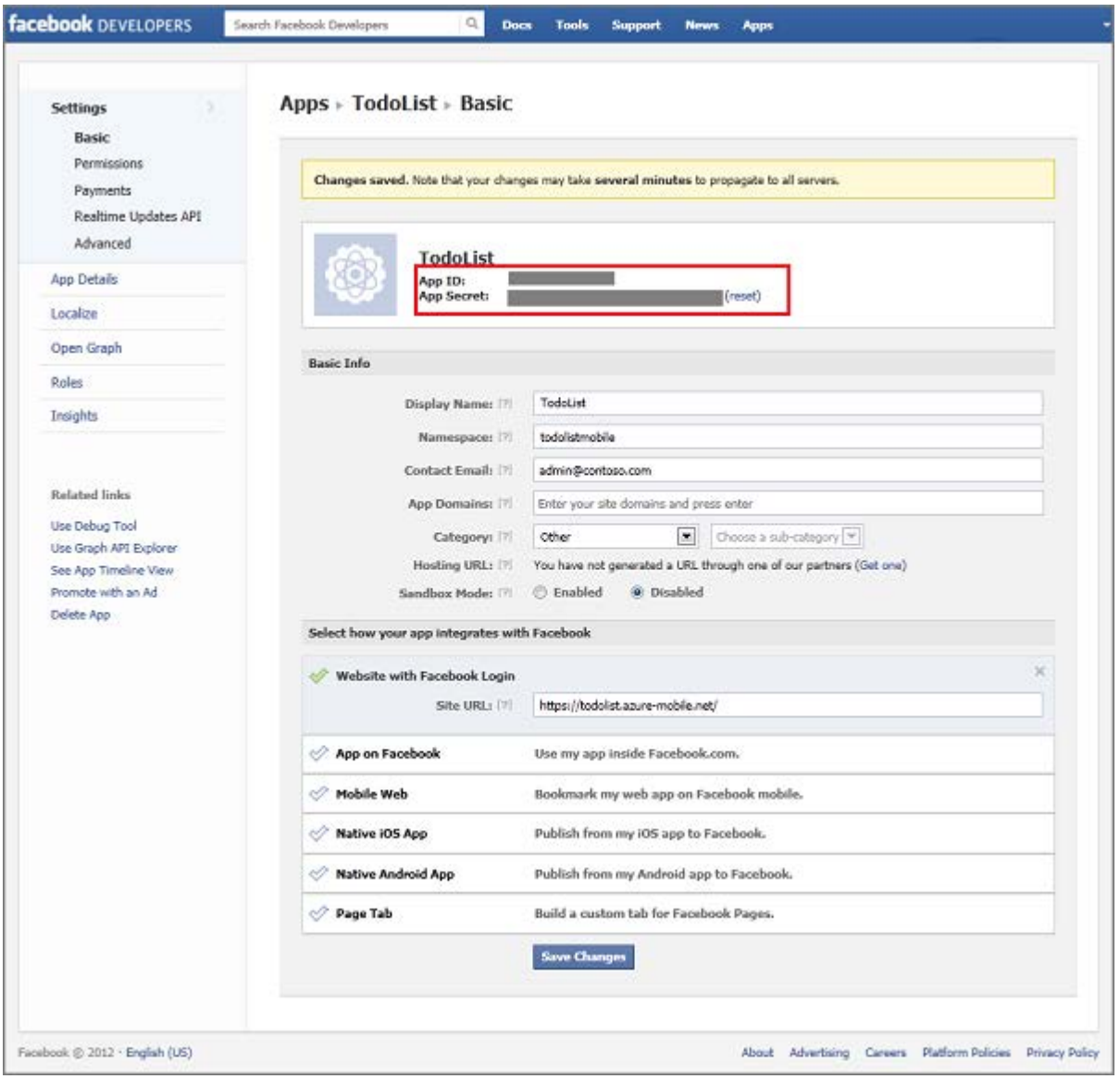


This registers the app with Facebook

7. Under **Select how your app integrates with Facebook**, expand **Website with Facebook Login**, type the URL of your mobile service in **Site URL**, and then click **Save Changes**.



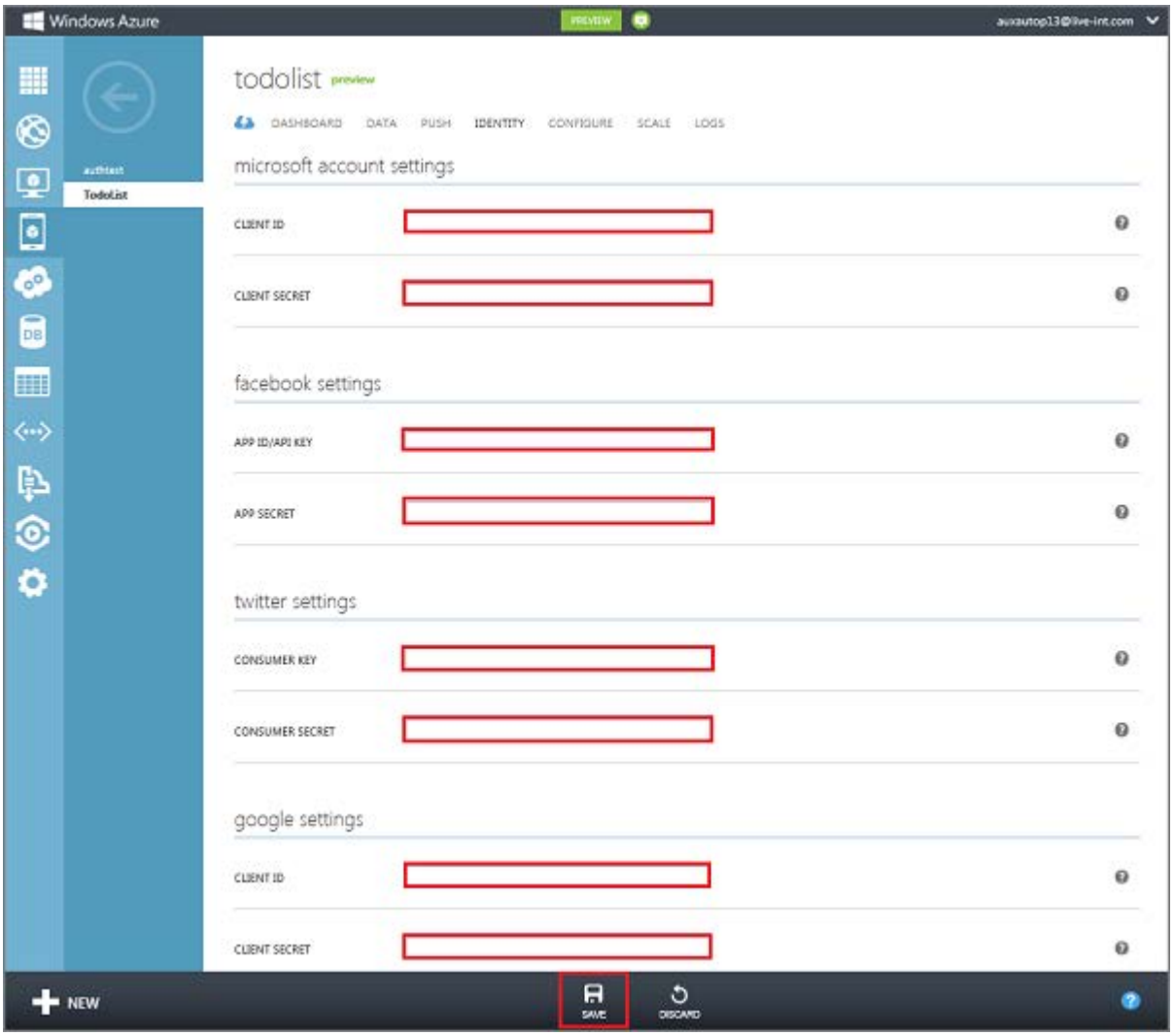
8. Make a note of the values of **App ID** and **App Secret**.



> **Security Note:** The app secret is an important security credential. Do not share this secret with anyone or distribute it with your app.

You are now ready to use a Facebook login for authentication in your app by providing the App ID and App Secret values to Mobile Services.

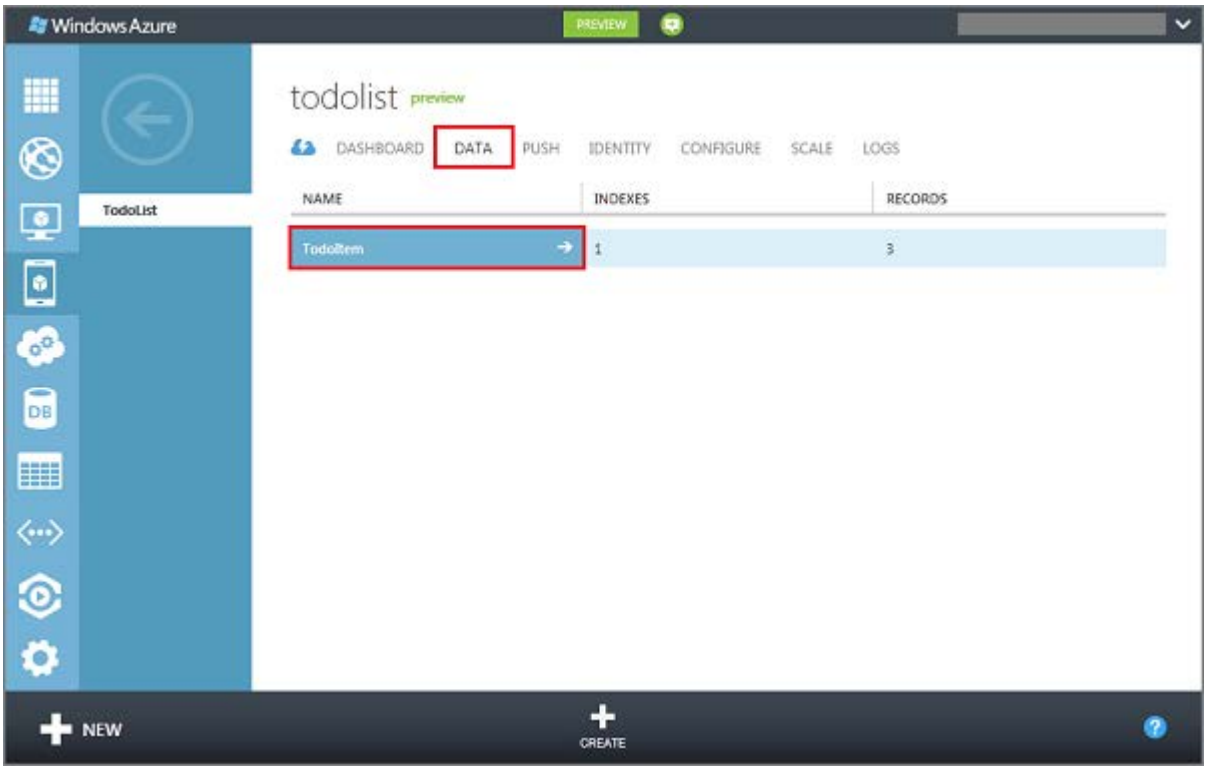
9. Back in the Management Portal, click the **Identity** tab, enter the app identifier and shared secret values obtained from your identity provider (in this case Facebook), and click **Save**.



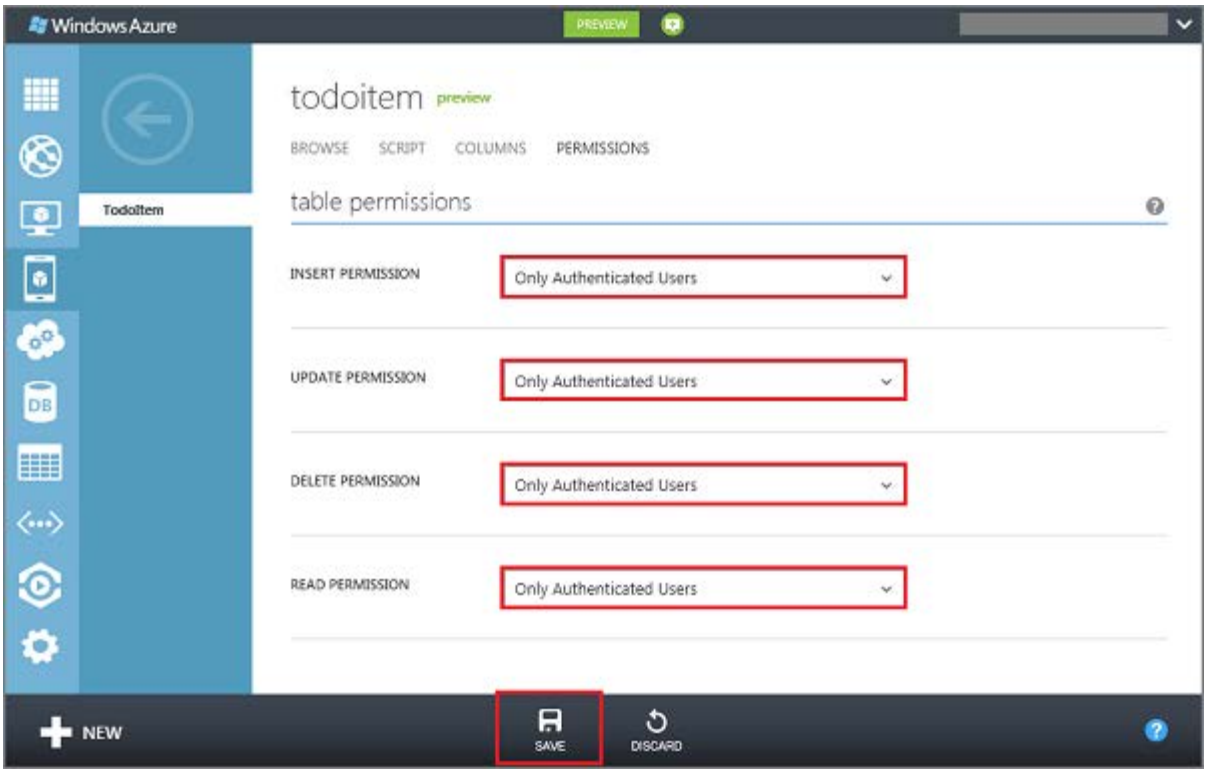
Both your mobile service and your app are now configured to work with your chosen authentication provider.

Restrict permissions to authenticated users

1. In the Management Portal, click the **Data** tab, and then click the **TodoItem** table.



2. Click the **Permissions** tab, set all permissions to **Only authenticated users**, and then click **Save**. This will ensure that all operations against the **TodoItem** table require an authenticated user. This also simplifies the scripts in the next tutorial because they will not have to allow for the possibility of anonymous users.



3. In Visual Studio 2012 Express for Windows 8, open the project that you created when you completed the tutorial Get started with data in Mobile Services.

4. Press the F5 key to run this quickstart-based app; verify that an unhandled exception with a status code of 401 (Unauthorized) is raised after the app starts.

   This happens because the app attempts to access Mobile Services as an unauthenticated user, but the *TodoItem* table now requires authentication.

   Next, you will update the app to authenticate users before requesting resources from the mobile service.

Add authentication to the app

1. Open the project file default.js and in the **app.OnActivated** method overload, replace the call to the **refreshTodoItems** method with the following code:

```
var userId = null;


// Request authentication from Mobile Services using a Facebook login.
var login = function () {
```

```
    return new WinJS.Promise(function (complete) {
        mobileService.login("facebook").done(function (results) {;
            userId = results.userId;
            refreshTodoItems();
            var message = "You are now logged in as: " + userId;
            var dialog = new Windows.UI.Popups.MessageDialog(message);
            dialog.showAsync().done(complete);
        }, function (error) {
            userId = null;
            var dialog = new Windows.UI.Popups
                .MessageDialog("An error occurred during login", "Login
Required");
            dialog.showAsync().done(complete);
        });
    });
}


var authenticate = function () {
    login().then(function () {
        if (userId === null) {


        // Authentication failed, try again.
        authenticate();
    }
});


}
authenticate();
```

This creates a member variable for storing the current user and a method to handle the authentication process. The user is authenticated by using a Facebook login.

> **Note:** When you are using an identity provider other than Facebook, change the value passed to the **login** method above to one of the following: **microsoftaccount**, **facebook**, **twitter**, or **google**.

2. Press the F5 key to run the app and sign into the app with your chosen identity provider.

When you are successfully logged-in, the app should run without errors, and you should be able to query Mobile Services and make updates to data. In the next tutorial, you will take the user ID value provided by Mobile Services based on an authenticated user and use it to filter the data returned by Mobile Services.

Use scripts to authorize users in Mobile Services

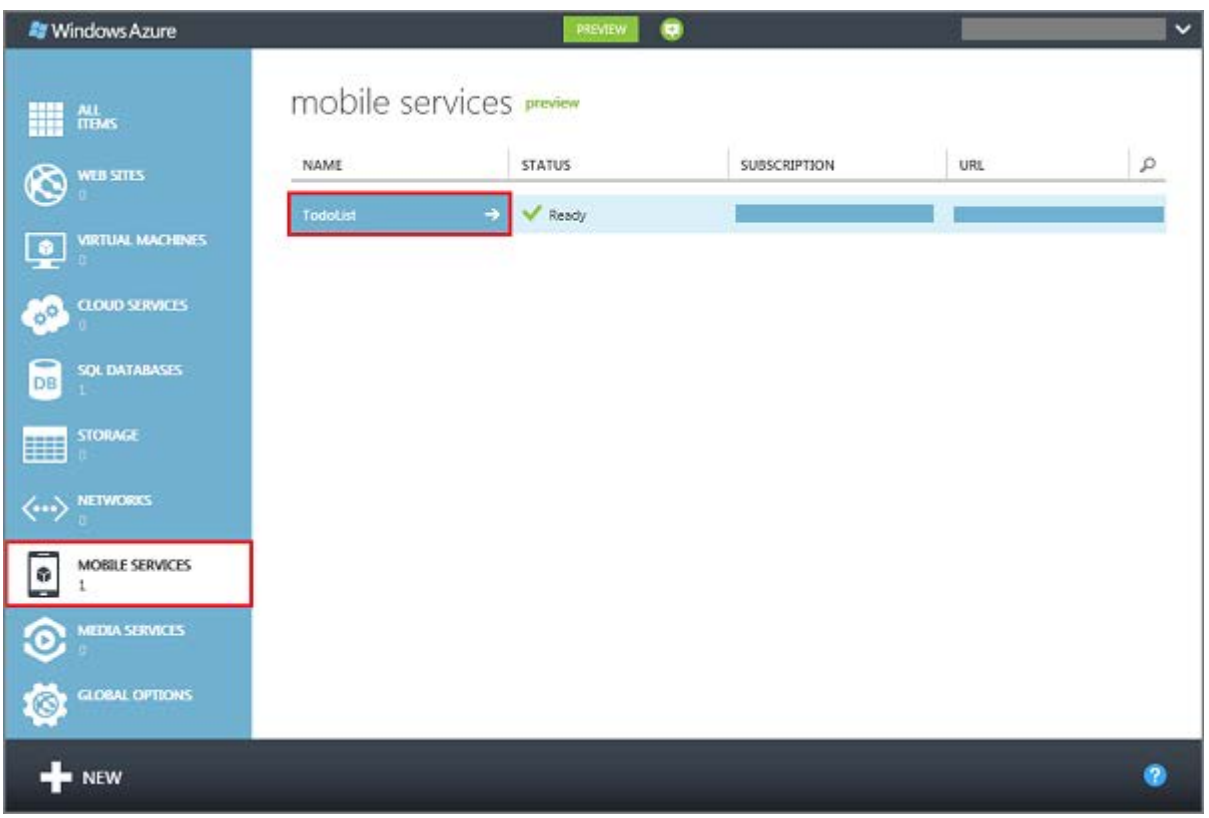
This section shows you how to use server scripts to authorize authenticated users for accessing data in Windows Azure Mobile Services from a Windows Store app. In this tutorial you register scripts with Mobile Services to filter queries based on the userId of an authenticated user, ensuring that each user can see only their own data.

This tutorial is based on the Mobile Services quickstart and builds on the previous tutorial Get started with authentication. Before you start this tutorial, you must first complete Get started with authentication.

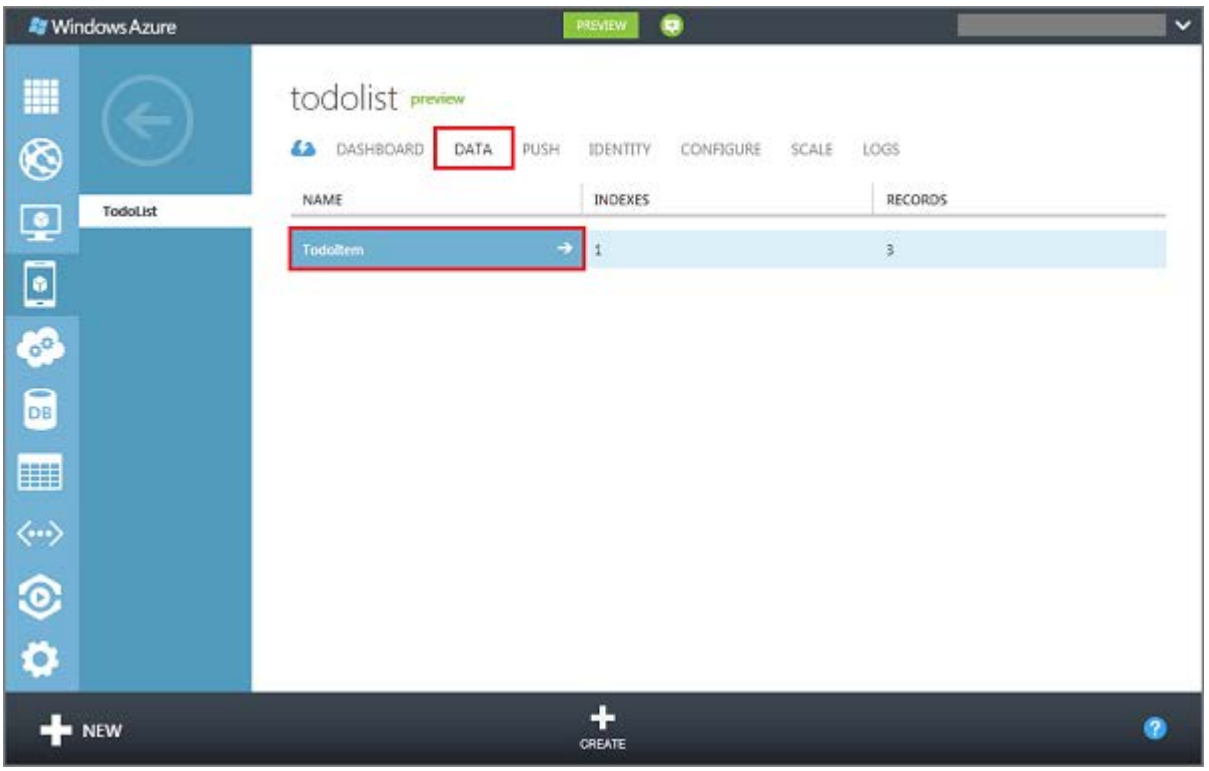
Register scripts

Because the quickstart app reads and inserts data, you need to register scripts for these operations against the TodoItem table.

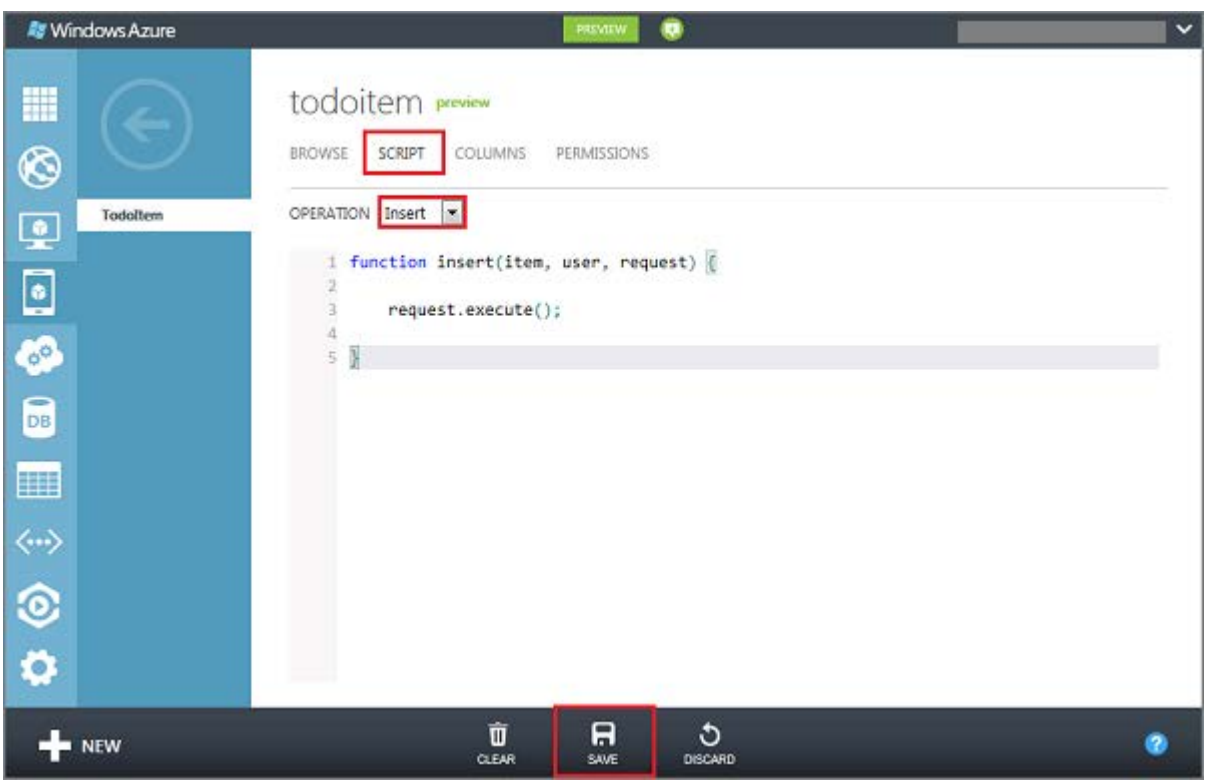
1. Log on to the Windows Azure Management Portal, click **Mobile Services**, and then click your app.



2. Click the **Data** tab, then click the **TodoItem** table.



3. Click **Script**, then select the **Insert** operation.



4. Replace the existing script with the following function, and then click **Save**.

```
function insert(item, user, request) {
  item.userId = user.userId;
  request.execute();
}
```

This script adds a userId value to the item, which is the user ID of the authenticated user, before it is inserted into the TodoItem table.

> **Note:** Dynamic schema must be enabled the first time that this insert script runs. With dynamic schema enabled, Mobile Services automatically adds the **userId** column to the **TodoItem** table on the first execution. Dynamic schema is enabled by default for a new mobile service, and it should be disabled before the app is published to the Windows Store.

5. Repeat steps 3 and 4 to replace the existing **Read** operation with the following function:

```
function read(query, user, request) {
   query.where({ userId: user.userId });
   request.execute();
}
```

This script filters the returned TodoItem objects so that each user only receives the items that they inserted.

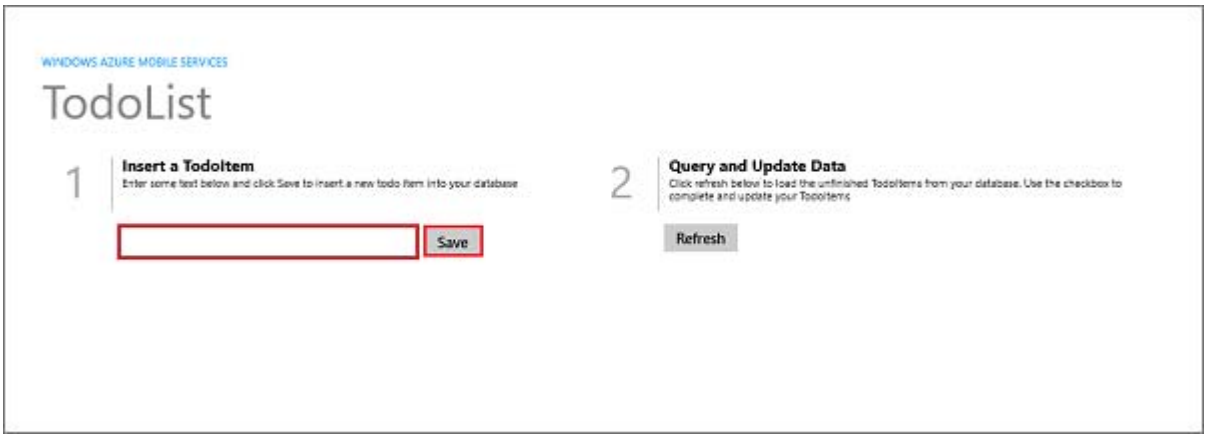
Test the app

1. In Visual Studio 2012 Express for Windows 8, open the project that you modified when you completed the tutorial Get started with authentication.

2. Press the F5 key to run the app, then log-on with your chosen identity provider.

   Notice that this time, although there are items already in the TodoItem table from preview tutorials, no items are returned. This happens because previous items were inserted without the userId column and now have null values.

3. In the app, enter text in **Insert a TodoItem** and then click **Save**.



This inserts both the text and the userId in the TodoItem table in the mobile service. Because the new item has the correct userId value, it is returned by the mobile service and displayed in the second column.

4. Back in the **todoitem** table in the Management Portal, click **Browse** and verify that each newly added item how has an associated userId value.

5. (Optional) If you have additional login accounts, you can verify that users can only see their own data by closing the app (Alt+F4) and then running it again. When the login credentials dialog is displayed, enter a different login, and then verify that the items entered under the previous account are not displayed.

   This concludes the tutorials that demonstrate the basics of working with authentication. The next section shows how to use Live Connect to provide a single sign-on experience in your Windows Store app.

Single sign-on for Windows Store apps by using Live Connect

This section shows you how to authenticate users in Windows Azure Mobile Services from a Windows Store app. In this tutorial, you add authentication to the quickstart project using Live Connect. When successfully authenticated by Live Connect, a logged-in user is welcomed by name and the user ID value is displayed.

This tutorial walks you through these basic steps to enable Live Connect authentication:

1. Register your app for authentication and configure Mobile Services
2. Restrict table permissions to authenticated users
3. Add authentication to the app

This tutorial requires the following:

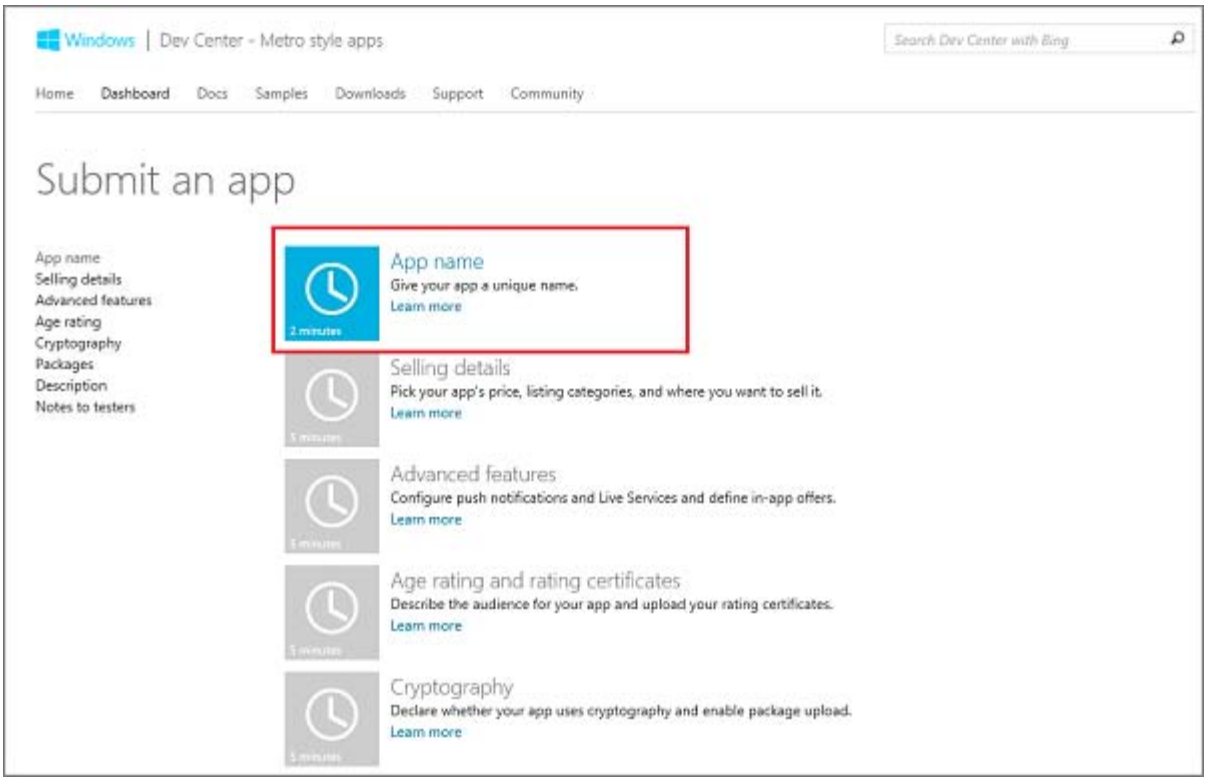
- Live SDK for Windows
- Microsoft Visual Studio 2012 Express for Windows 8 RC, or a later version

This tutorial is based on the Mobile Services quickstart. You must also first complete the tutorial Get started with data in Mobile Services.

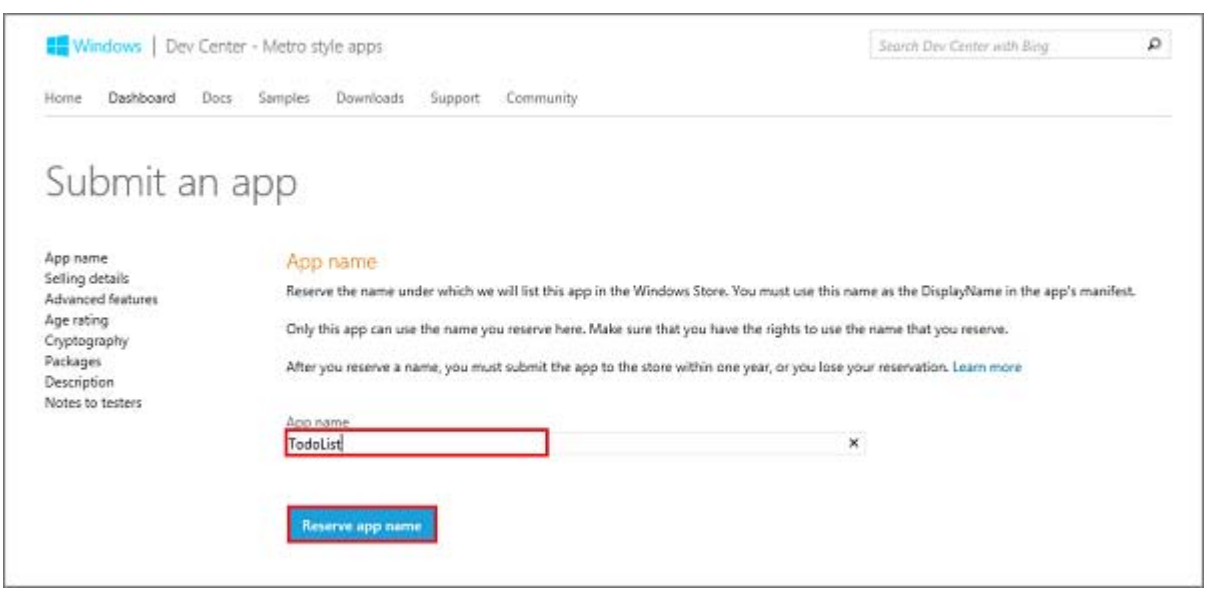
Register your app for the Windows Store

To be able to authenticate users, you must submit your app to the Windows Store. You must then register the client secret to integrate Live Connect with Mobile Services.

1. If you have not already registered your app, navigate to the Submit an app page at the Dev Center for Windows Store apps, log on with your Microsoft account, and then click **App name**.



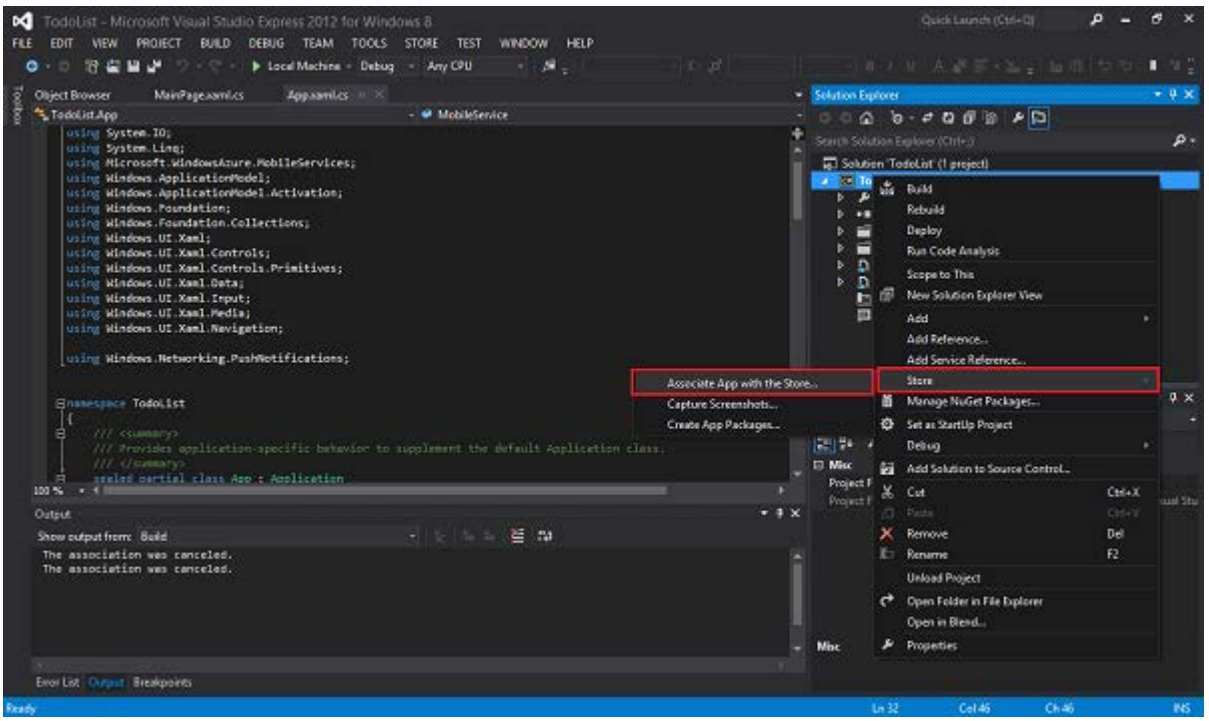
2. Type a name for your app in **App name**, click **Reserve app name**, and then click **Save**.



This creates a new Windows Store registration for your app.

3. In Visual Studio 2012 Express for Windows 8, open the project that you created when you completed the tutorial Get started with data in Mobile Services.

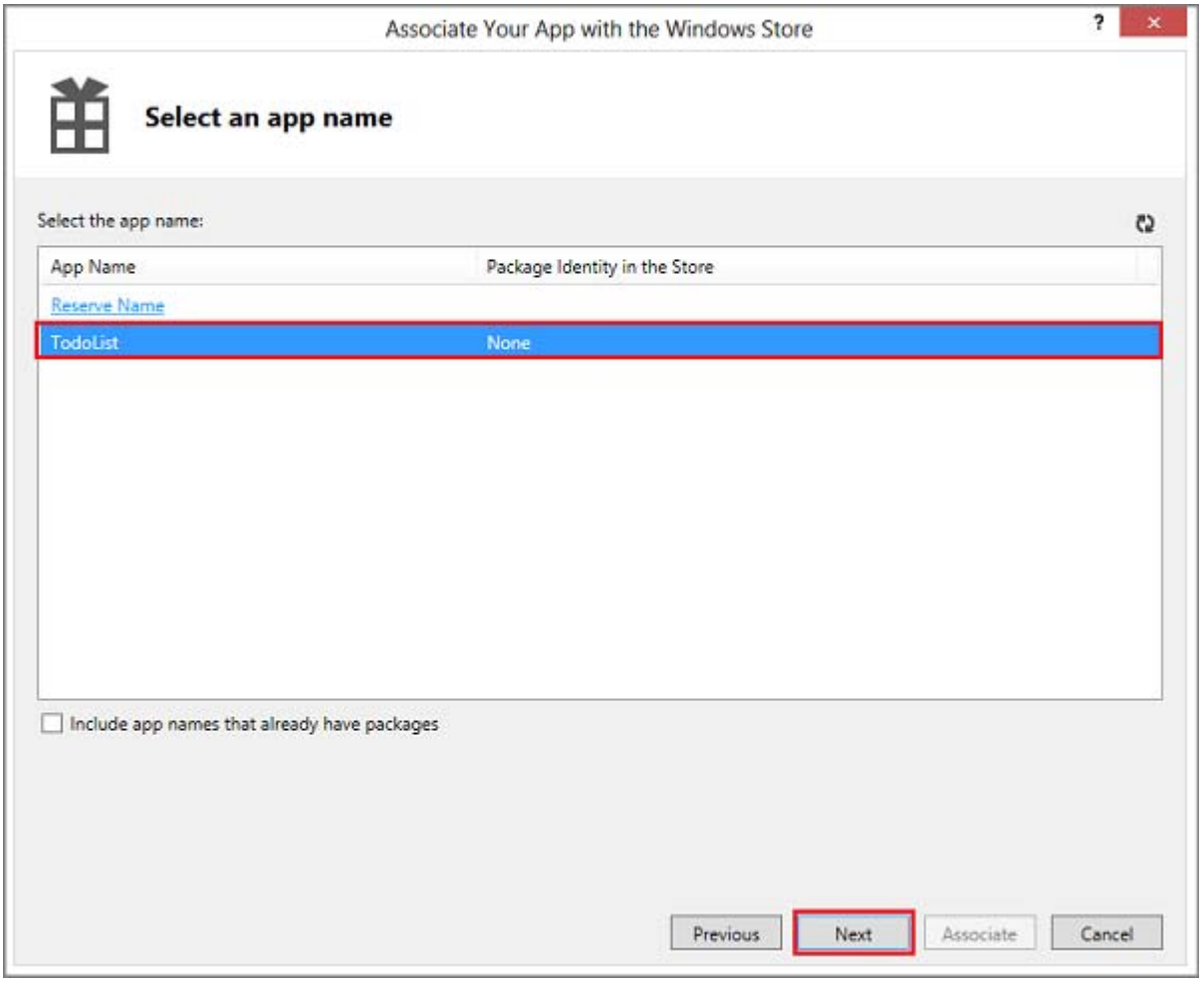
4. In solution explorer, right-click the project, click **Store**, and then click **Associate App with the Store...**.



This displays the **Associate Your App with the Windows Store** Wizard.

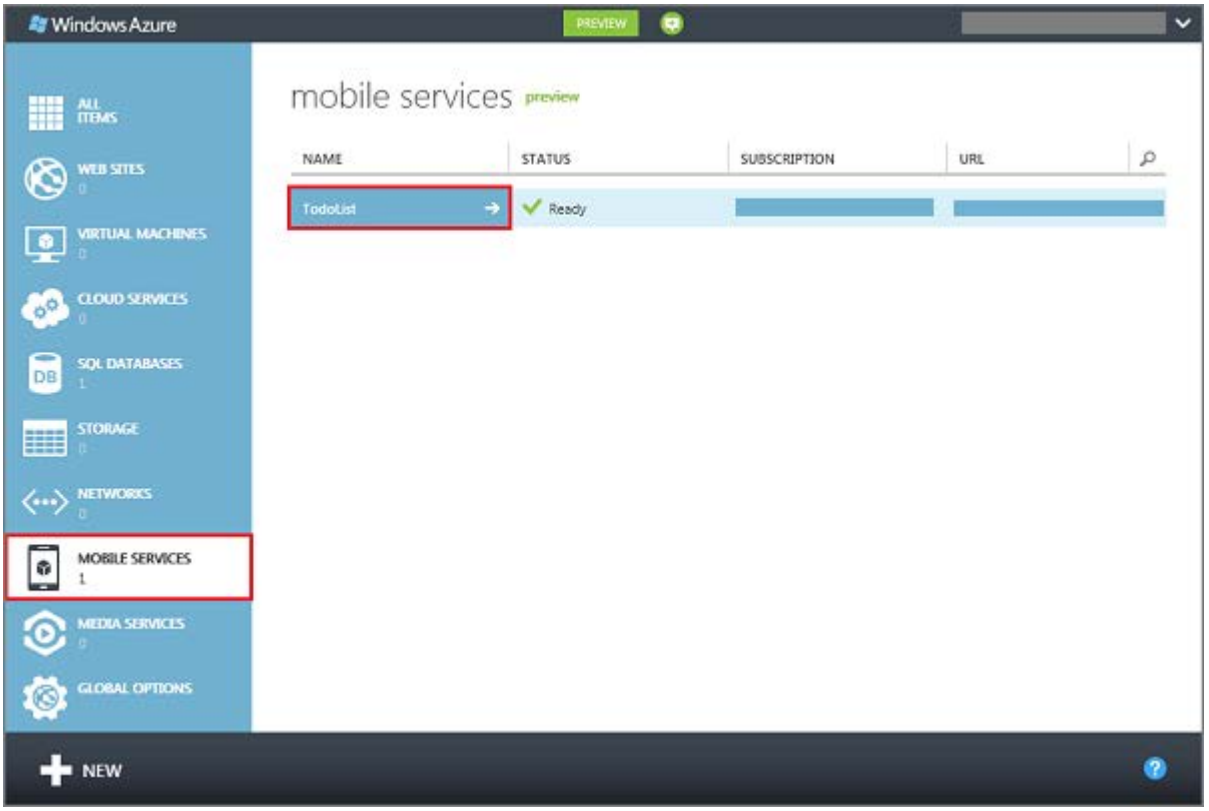
5. In the wizard, click **Sign in** and then login with your Microsoft account.

6. Select the app that you registered in step 2, click **Next**, and then click **Associate**.

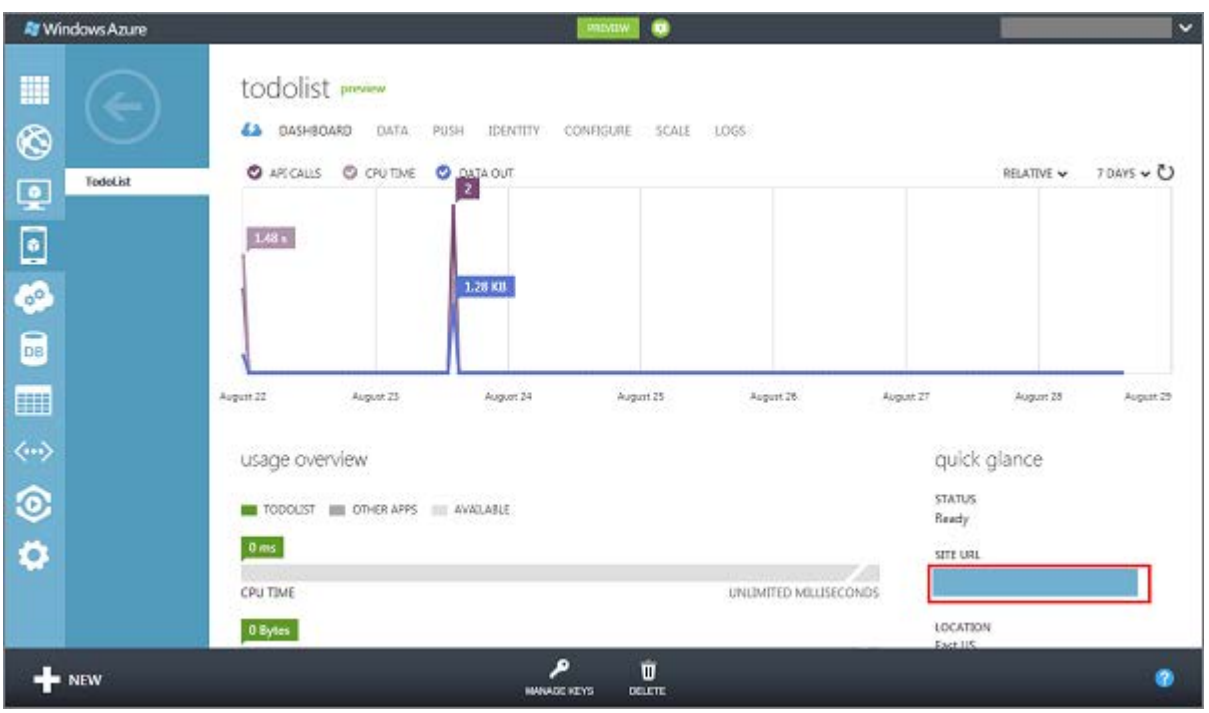


This adds the required Windows Store registration information to the application manifest.

7. Log on to the Windows Azure Management Portal, click **Mobile Services**, and then click your mobile service.

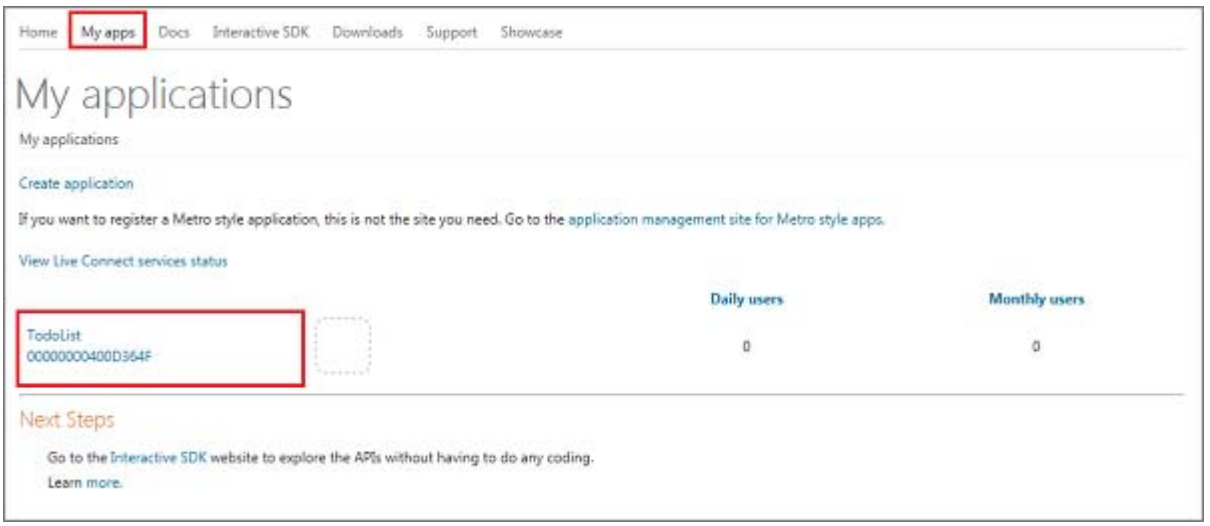


8. Click the **Dashboard** tab and make a note of the **Site URL** value.

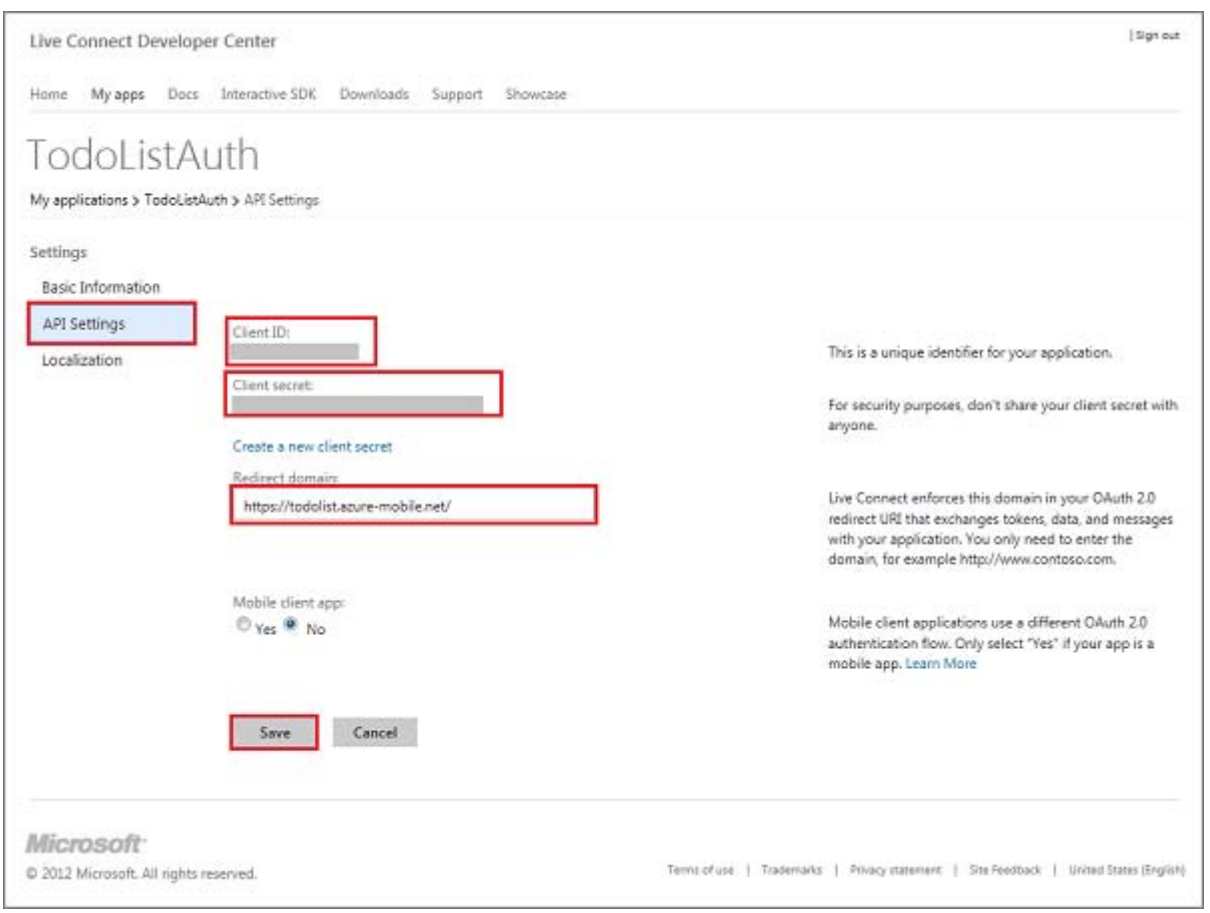


You will use this value to define the redirect domain.

9. Navigate to the My Applications page in the Live Connect Developer Center and click on your app in the **My applications** list.



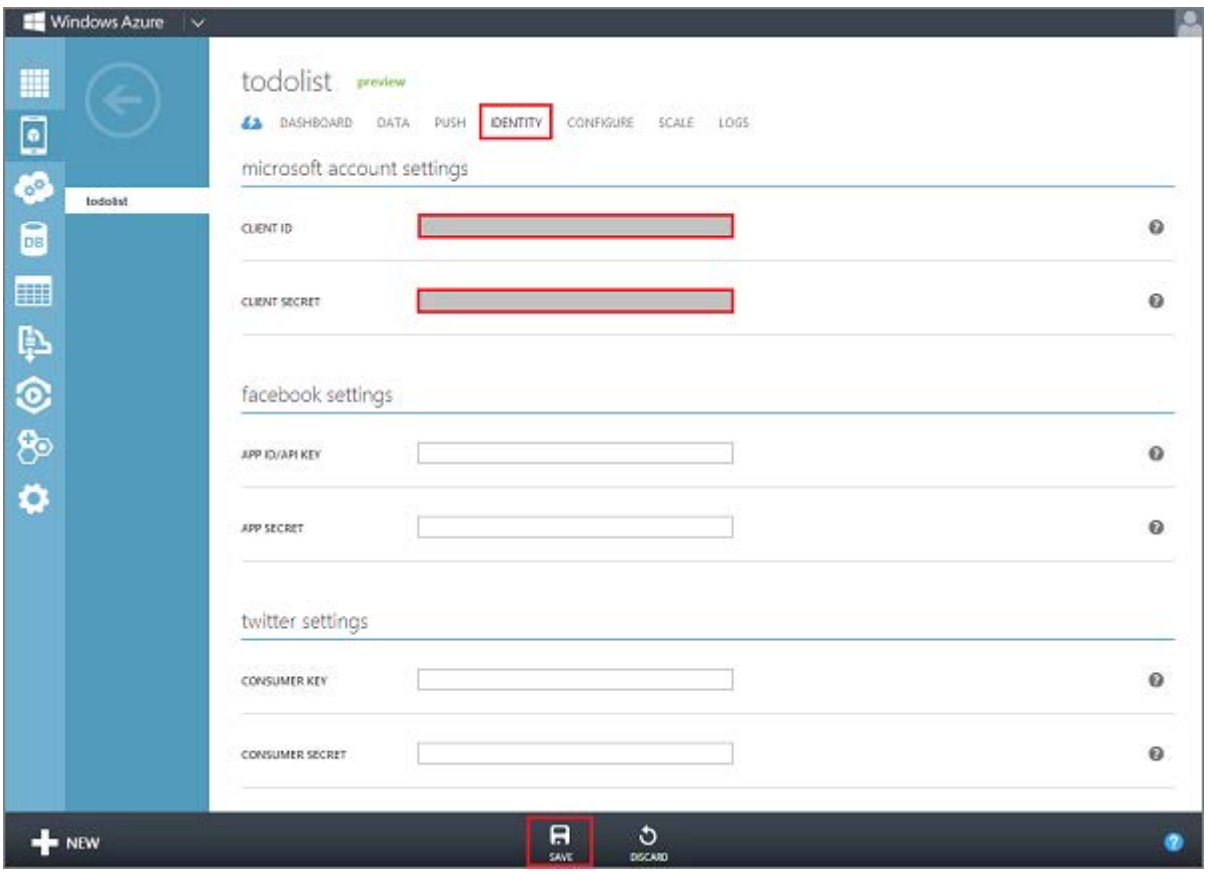
10. Click **Edit settings**, then **API Settings** and make a note of the value of **Client secret**.



Security Note: The client secret is an important security credential. Do not share the client secret with anyone or distribute it with your app.

11. In **Redirect domain**, enter the URL of your mobile service from Step 8, and then click **Save**.

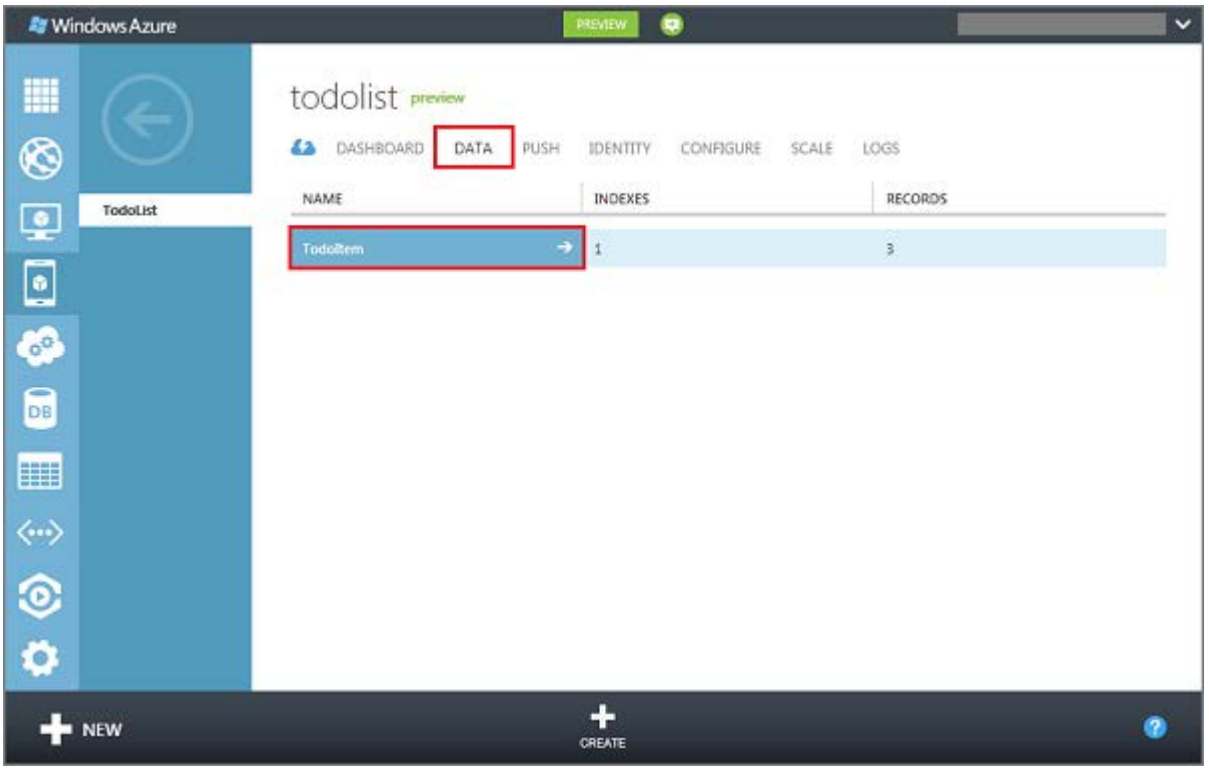
12. Back in the Management Portal, click the **Identity** tab, enter the **Client secret** obtained from Windows Store, and click **Save**.



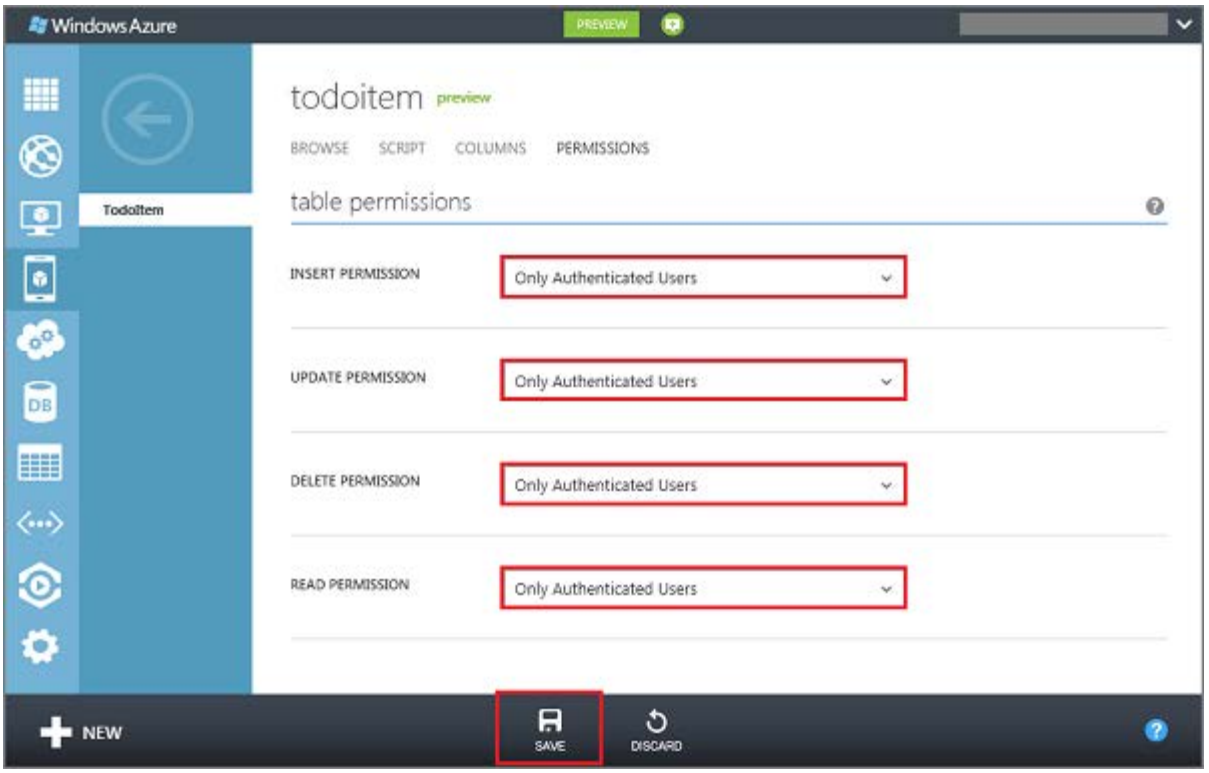
Both your mobile service and your app are now configured to work with Live Connect.

Restrict permissions to authenticated users

1. In the Management Portal, click the **Data** tab, and then click the **TodoItem** table.



2. Click the **Permissions** tab, set all permissions to **Only authenticated users**, and then click **Save**. This will ensure that all operations against the **TodoItem** table require an authenticated user. This also simplifies the scripts in the next tutorial because they will not have to allow for the possibility of anonymous users.



3. In Visual Studio 2012 Express for Windows 8, open the project that you created when you completed the tutorial Get started with data in Mobile Services.

4. Press the F5 key to run this quickstart-based app; verify that an exception with a status code of 401 (Unauthorized) is raised.

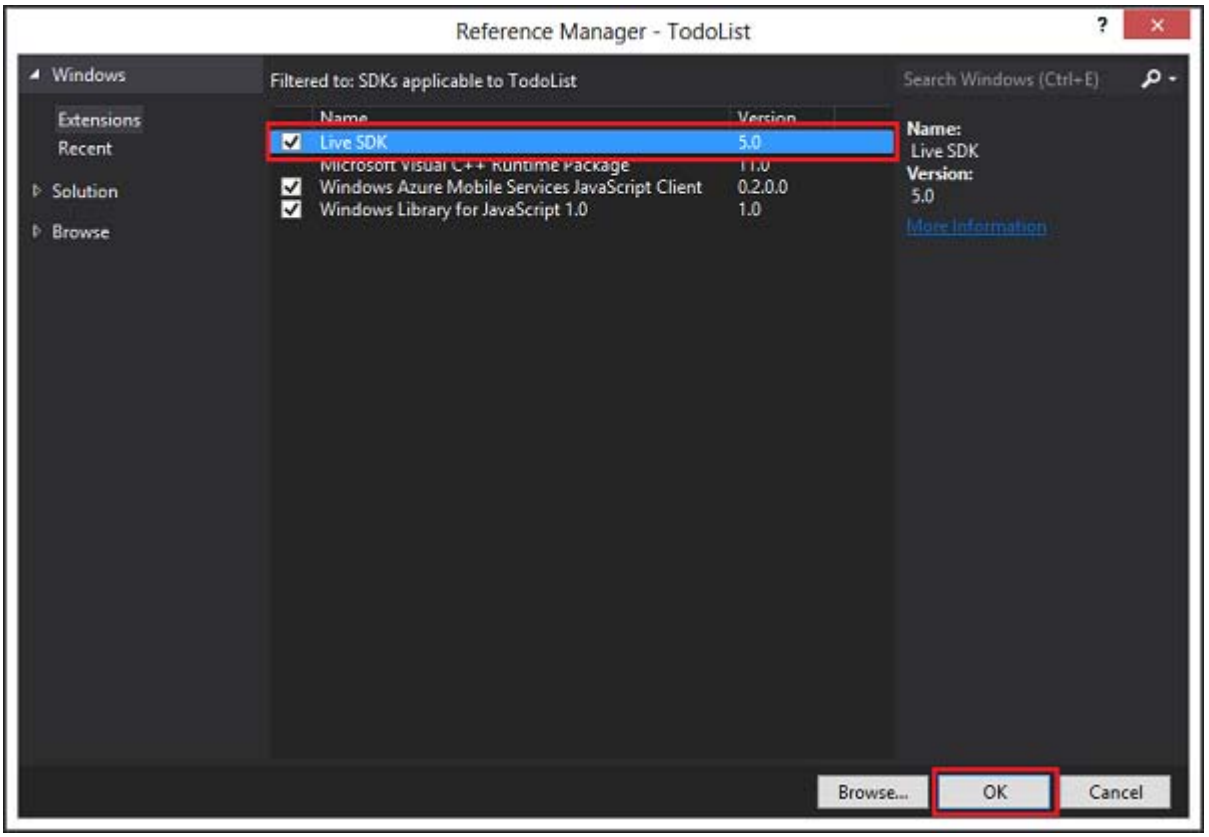
   This happens because the app is accessing Mobile Services as an unauthenticated user, but the *TodoItem* table now requires authentication.

   Next, you will update the app to authenticate users with Live Connect before requesting resources from the mobile service.

Add authentication to the app

1. Download and install the Live SDK for Windows.

2. In the **Project** menu in Visual Studio, click **Add Reference**, then expand **Windows**, click **Extensions**, check **Live SDK**, and then click **OK**.



This adds a reference to the Live SDK to the project.

3. Open the default.html project file and add the following <script> element in the <head> element.

```
<script src="///LiveSDKHTML/js/wl.js"></script>
```

This enables Microsoft IntelliSense in the default.html file.

4. Open the project file default.js and add the following comment to the top of the file.

```
/// <reference path="///LiveSDKHTML/js/wl.js" />
```

This enables Microsoft IntelliSense in the default.js file.

5. In the **app.OnActivated** method overload, replace the call to the **refreshTodoItems** method with the following code:

```
var session = null;


var logout = function () {
```

```
    return new WinJS.Promise(function (complete) {
        WL.getLoginStatus().then(function () {
            if (WL.canLogout()) {
                WL.logout(complete);
            }
            else {
                complete();
            }
        });
    });
};


var login = function () {
    return new WinJS.Promise(function (complete) {
        WL.login({ scope: "wl.basic"}).then(function (result) {
            session = result.session;



        WinJS.Promise.join([
            WL.api({ path: "me", method: "GET" }),
            client.login(result.session.authentication_token)
        ]).done(function (results) {
            var profile = results[0];
            var mobileServicesUser = results[1];
            refreshTodoItems();
            var title = "Welcome " + profile.first_name + "!";
            var message = "You are now logged in as: " +
mobileServicesUser.userId;
            var dialog = new Windows.UI.Popups.MessageDialog(message,
title);
            dialog.showAsync().done(complete);
        });
    }, function (error) {
        session = null;
        var dialog = new Windows.UI.Popups.MessageDialog("You must log
in.", "Login Required");
        dialog.showAsync().done(complete);
    });
});
```

```
}


var authenticate = function () {

    // Force a logout to make it easier to test with multiple Microsoft
Accounts
    logout().then(login).then(function () {
        if (session === null) {
            // Authentication failed, try again.
            authenticate();
        }
    });
}


WL.init({
    redirect_uri: "<< INSERT REDIRECT DOMAIN HERE >>"
});


authenticate();
```

This initializes the Live Connect client, forces a logout, sends a new login request to Live Connect, sends the returned authentication token to Mobile Services, and then displays information about the logged-in user.

> **Note:** This code forces a logout, when possible, to make sure that the user is prompted for credentials each time the application runs. This makes it easier to test the application with different Microsoft Accounts to ensure that the authentication is working correctly. This mechanism will only work if the logged in user does not have a connected Microsoft account.

6. Update string << *INSERT REDIRECT DOMAIN HERE* >> from the previous step with the redirect domain that was specified when setting up the app in Live Connect, in the format **https://*service-name*.azure-mobile.net/**.

7. Press the F5 key to run the app and sign into Live Connect with your Microsoft Account.

When you are successfully logged-in, the app should run without errors, and you should be able to query Mobile Services and make updates to data.

Get started with push notifications in Mobile Services

This section shows you how to use Windows Azure Mobile Services to send push notifications to a Windows Store app. In this tutorial you add push notifications using the Windows Push Notification service (WNS) to the quickstart project. When complete, your mobile service will send a push notification each time a record is inserted.

> **Note:** This tutorial demonstrates a simplified way of sending push notifications by attaching a push notification channel to the inserted record. Be sure to follow along with the next tutorial to get a better idea of how to incorporate push notifications into your real-world apps.

This tutorial walks you through these basic steps to enable push notifications:

1. Register your app for push notifications and configure Mobile Services
2. Add push notifications to the app
3. Update scripts to send push notifications
4. Insert data to receive notifications

This tutorial requires the following:

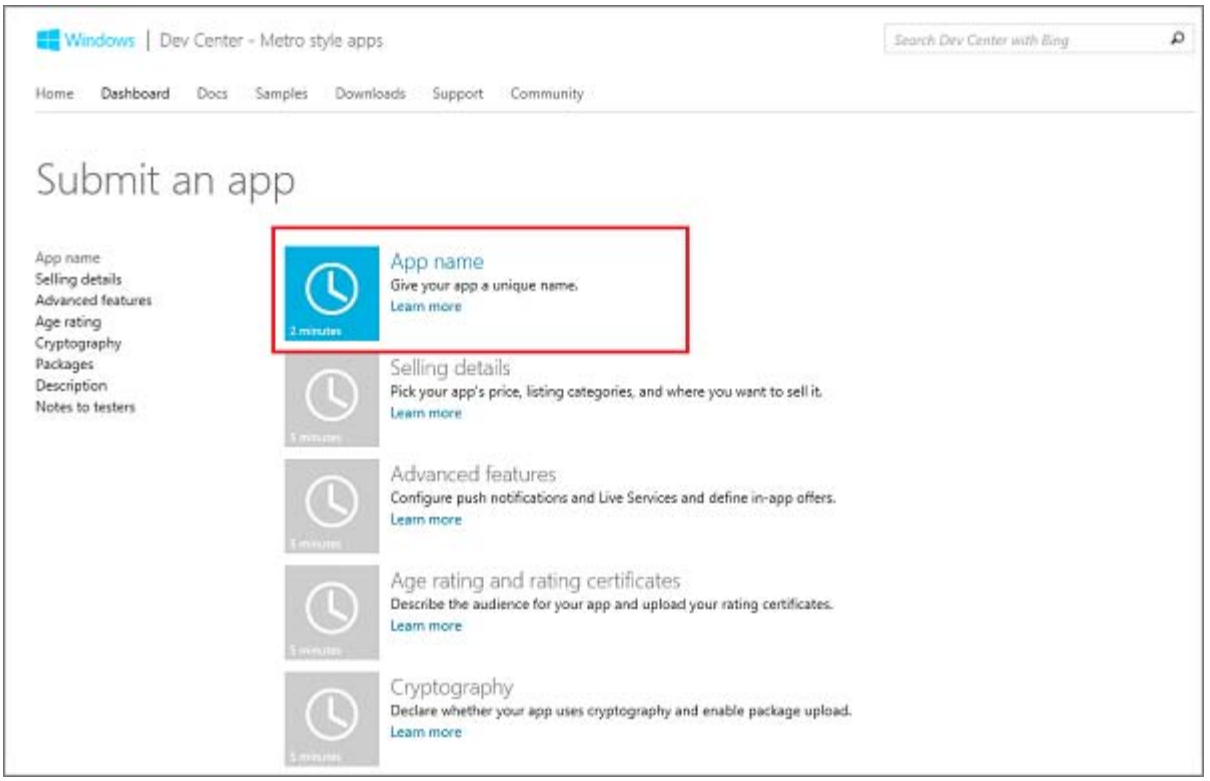
- Microsoft Visual Studio 2012 Express for Windows 8 RC, or a later version

This tutorial is based on the Mobile Services quickstart. Before you start this tutorial, you must first complete Get started with data in Mobile Services.

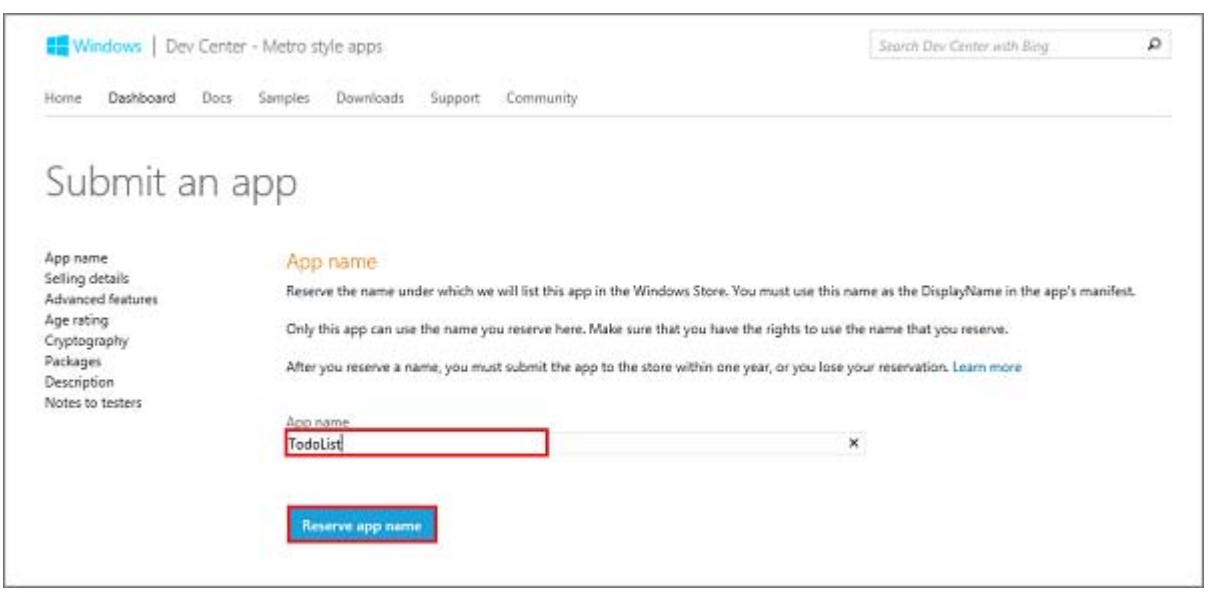
Register your app for the Windows Store

To be able to send push notifications to Windows Store apps from Mobile Services, you must submit your app to the Windows Store. You must then configure your mobile service to integrate with WNS.

1. If you have not already registered your app, navigate to the Submit an app page at the Dev Center for Windows Store apps, log on with your Microsoft account, and then click **App name**.



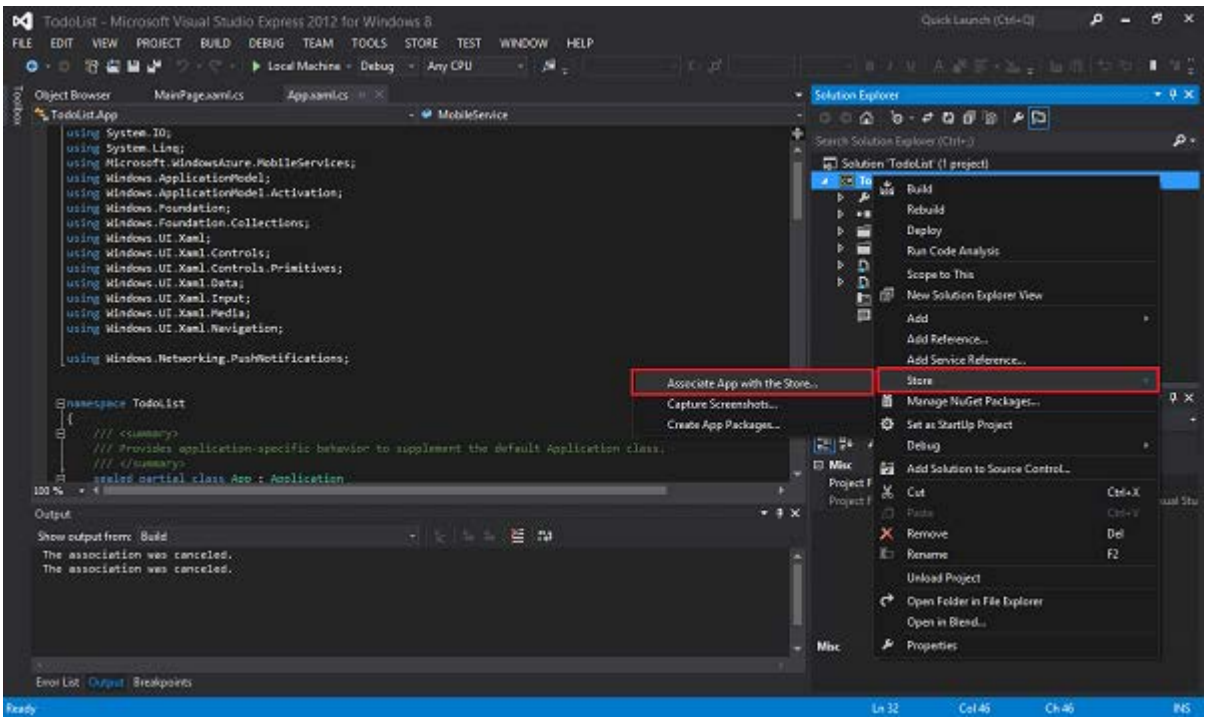
2. Type a name for your app in **App name**, click **Reserve app name**, and then click **Save**.



This creates a new Windows Store registration for your app.

3. In Visual Studio 2012 Express for Windows 8, open the project that you created when you completed the tutorial Get started with data in Mobile Services.

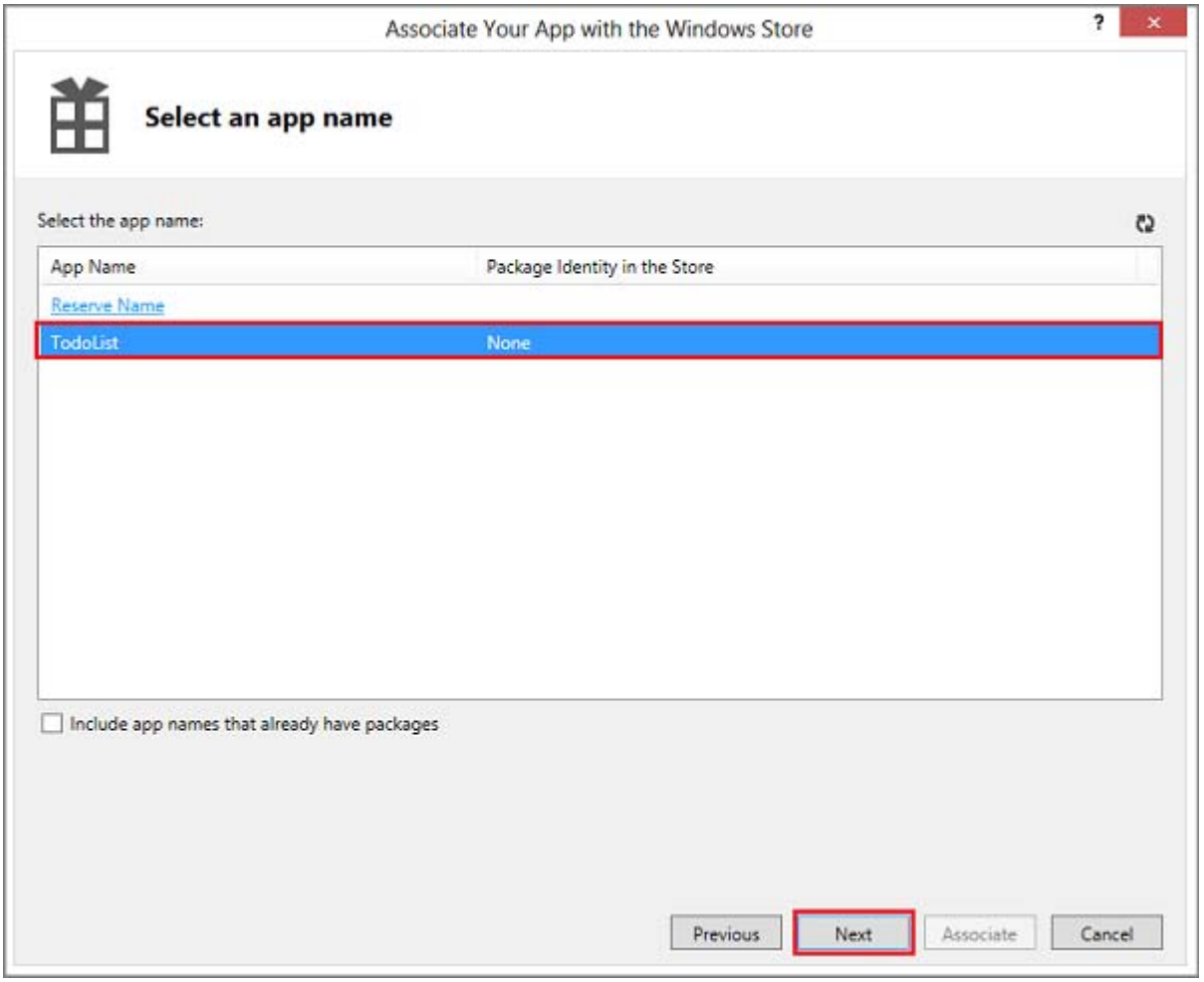
4. In solution explorer, right-click the project, click **Store**, and then click **Associate App with the Store...**.



This displays the **Associate Your App with the Windows Store** Wizard.

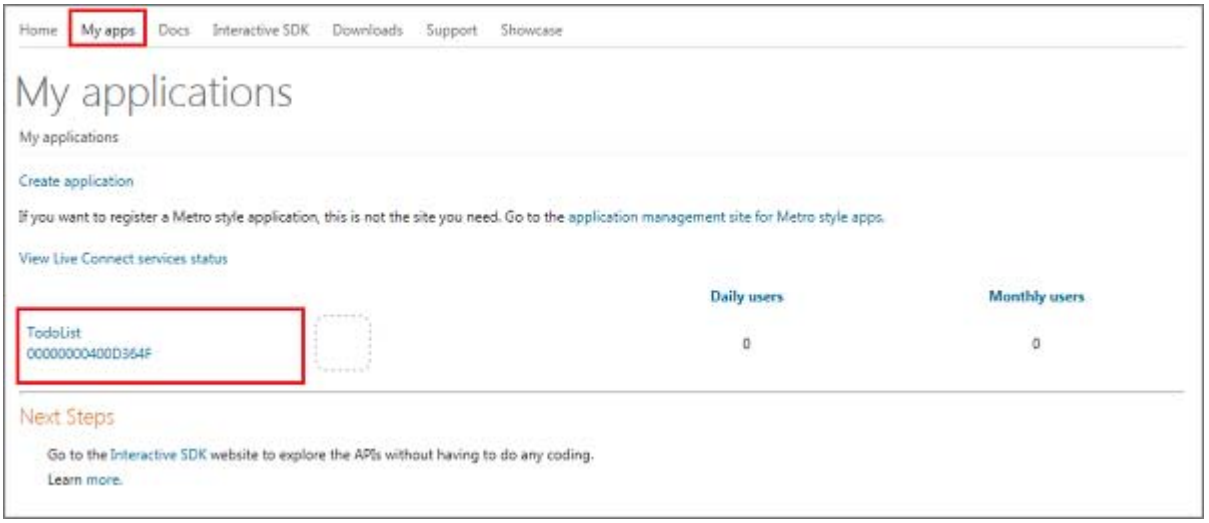
5. In the wizard, click **Sign in** and then login with your Microsoft account.

6. Select the app that you registered in step 2, click **Next**, and then click **Associate**.

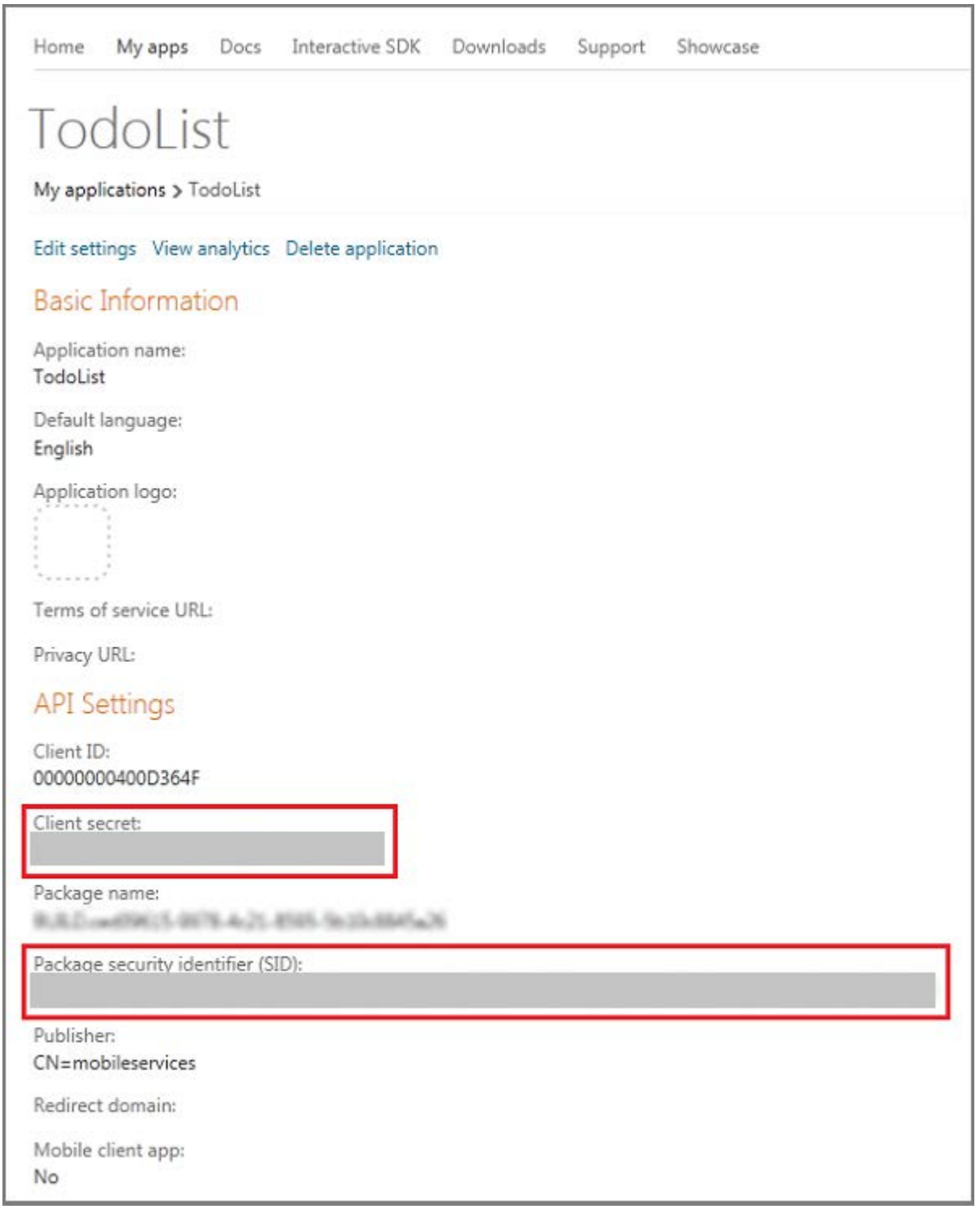


This adds the required Windows Store registration information to the application manifest.

7. Navigate to the My Applications page in the Live Connect Developer Center and click on your app in the **My applications** list.

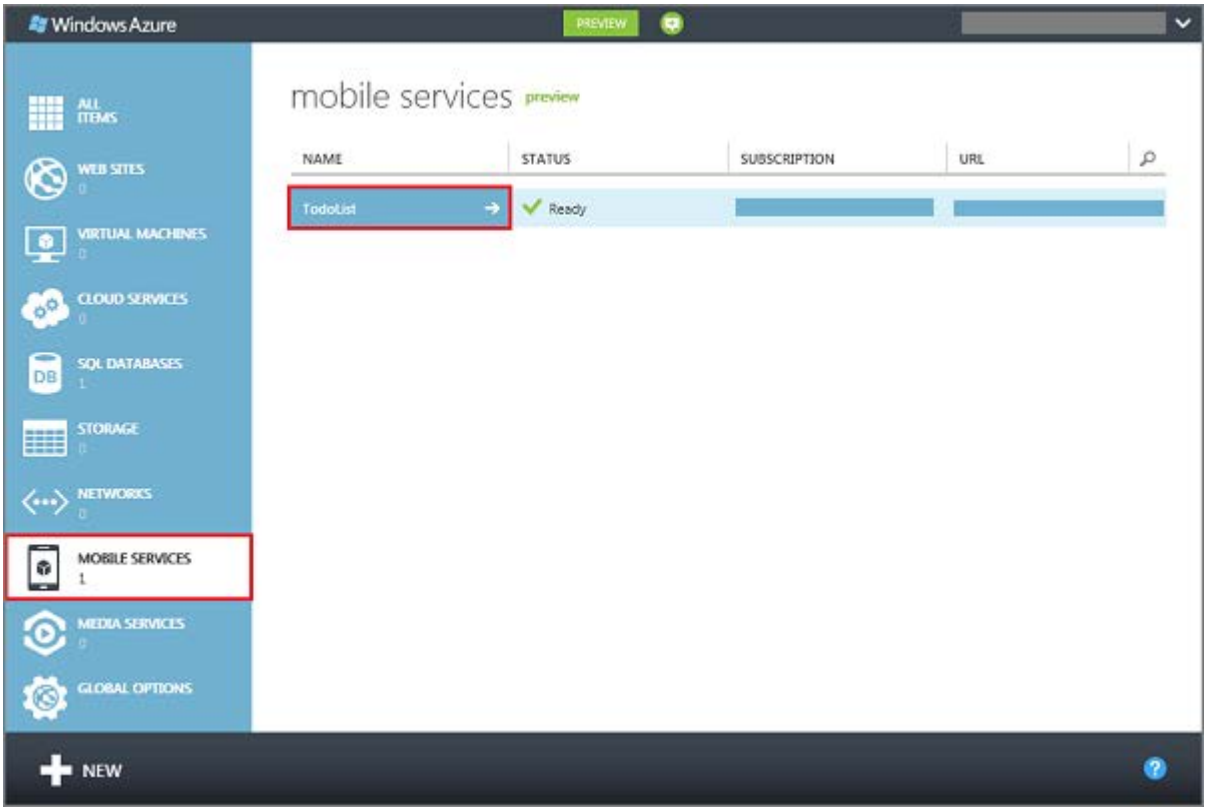


8. Under **API Settings**, make a note of the values of **Client secret** and **Package security identifier (SID)**.

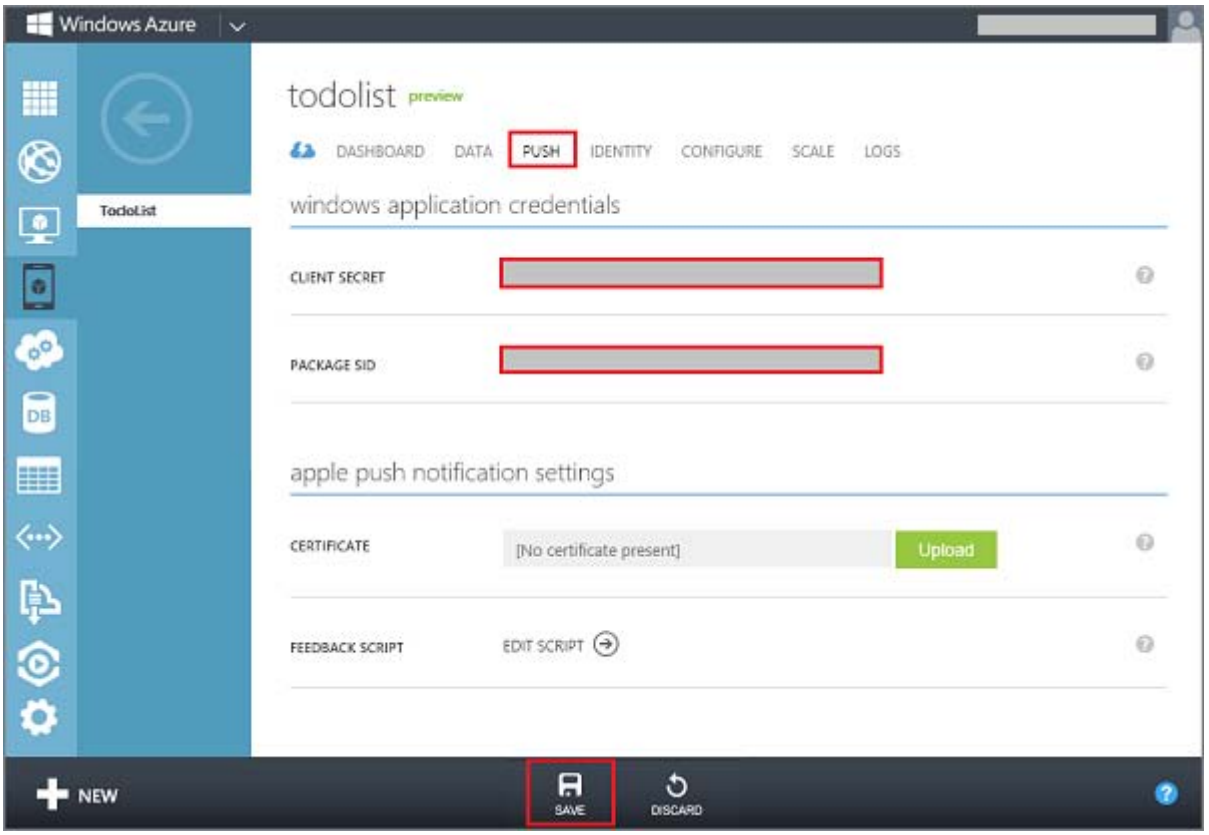


Security Note: The client secret and package SID are important security credentials. Do not share these secrets with anyone or distribute them with your app.

9. Log on to the Windows Azure Management Portal, click **Mobile Services**, and then click your app.



10. Click the **Push** tab, enter the **Client secret** and **Package SID** values obtained from WNS in Step 4, and then click **Save**.



Both your mobile service and your app are now configured to work with WNS.

Add push notifications to your app

1. Open the file default.js and insert the following code fragment into the app.OnActivated method overload, just after the args.setPromise method:

```
// Get the channel for the application.
var channel;
var channelOperation = Windows.Networking.PushNotifications
    .PushNotificationChannelManager
    .createPushNotificationChannelForApplicationAsync()
    .then(function (newChannel) {
        channel = newChannel;
    });
```

This code acquires and stores a push notification channel each time the application is launched.

2. Replace the **click** event listener definition for **buttonSave** with the following code:

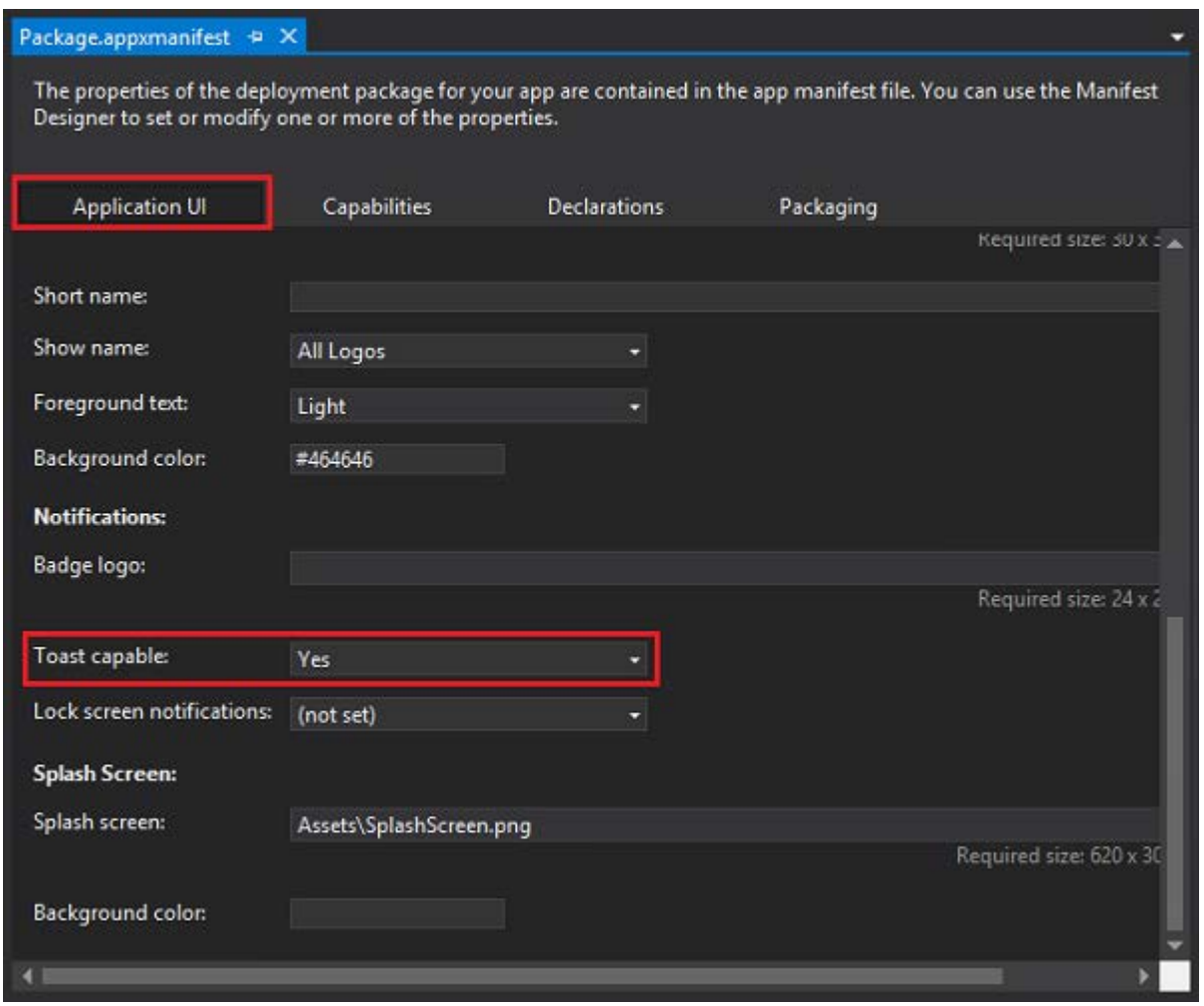
```
buttonSave.addEventListener("click", function () {
```

```
        insertTodoItem({
            text: textInput.value,
            complete: false,
            channel: channel.uri
        });
    });
```

This sets the client's current channel value on the item before it is sent to the mobile service.

> **Note:** When dynamic schema is enabled on your mobile service, a new 'channel' column is automatically added to the **TodoItem** table when a new item that contains this property is inserted.

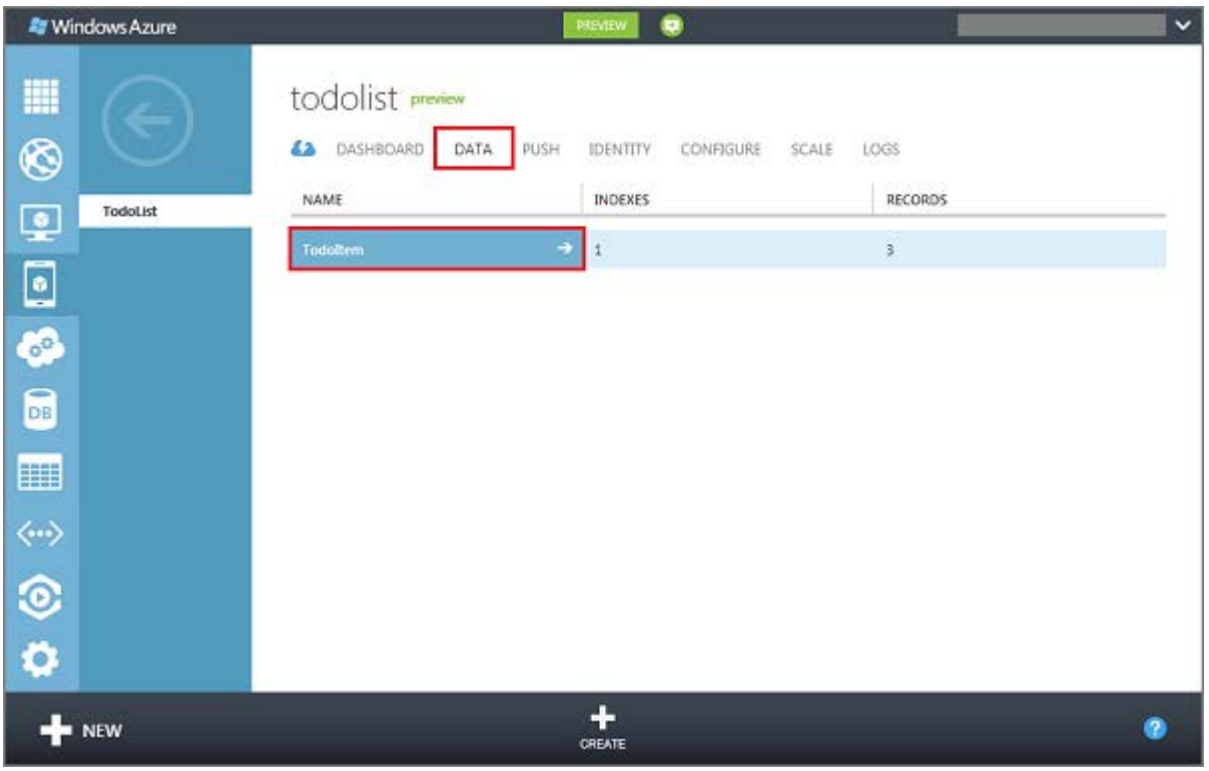
3. (Optional) If you are not using the Management Portal-generated quickstart project, open the Package.appxmanifest file and make sure that in the **Application UI** tab, **Toast capable** is set to **Yes**.



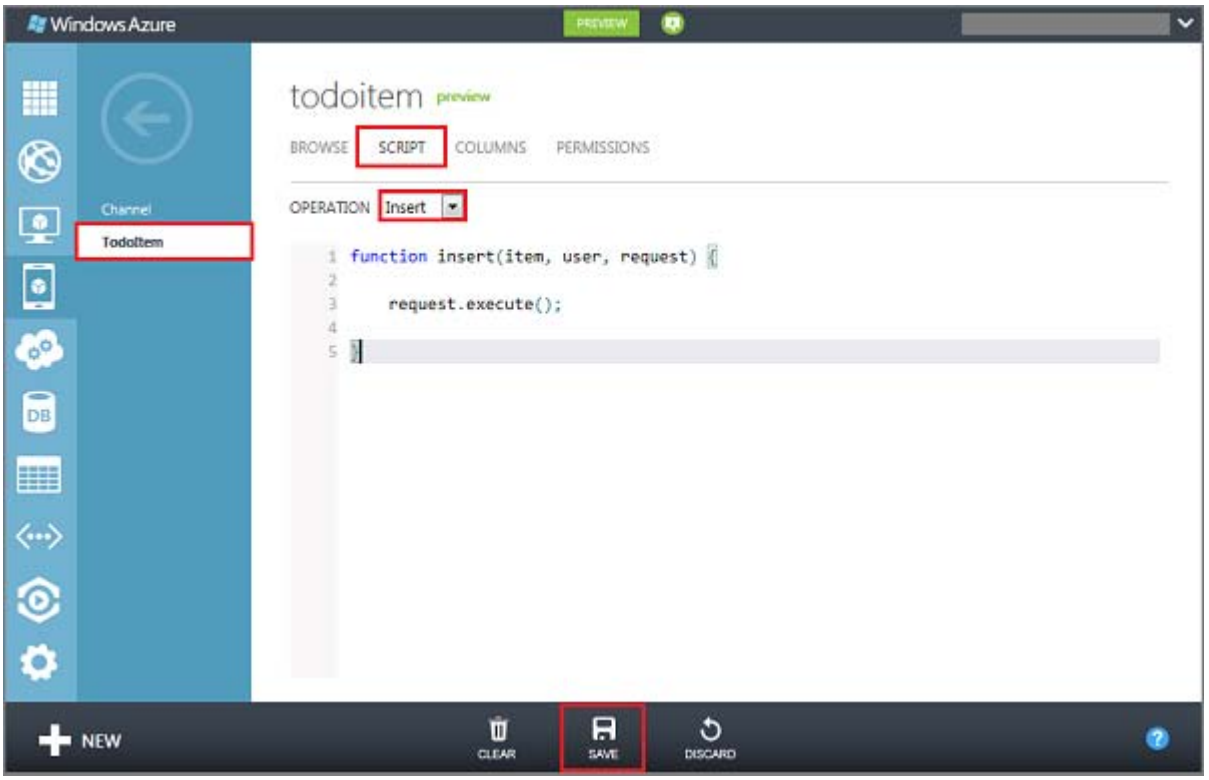
This makes sure that your app can raise toast notifications. These notifications are already enabled in the downloaded quickstart project.

Update the registered insert script in the Management Portal

1. In the Management Portal, click the **Data** tab and then click the **TodoItem** table.



2. In **todoitem**, click the **Script** tab and select **Insert**.



This displays the function that is invoked when an insert occurs in the **TodoItem** table.

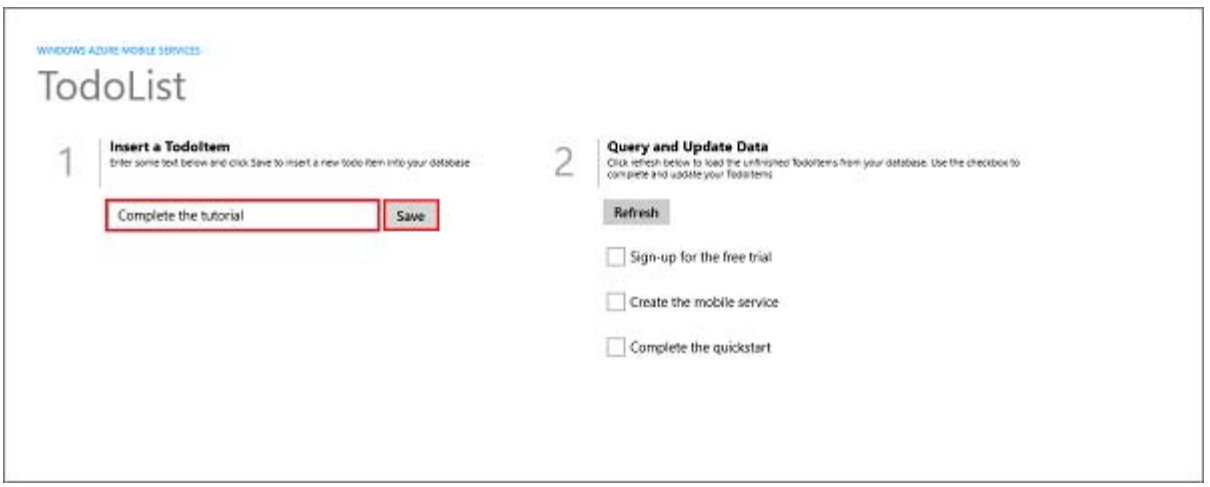
3. Replace the insert function with the following code, and then click **Save**:

```
function insert(item, user, request) {
    request.execute({
        success: function() {
            // Write to the response and then send the notification in the
background
            request.respond();
            push.wns.sendToastText04(item.channel, {
                text1: item.text
            }, {
                success: function(pushResponse) {
                    console.log("Sent push:", pushResponse);
                }
            });
        }
    });
}
```

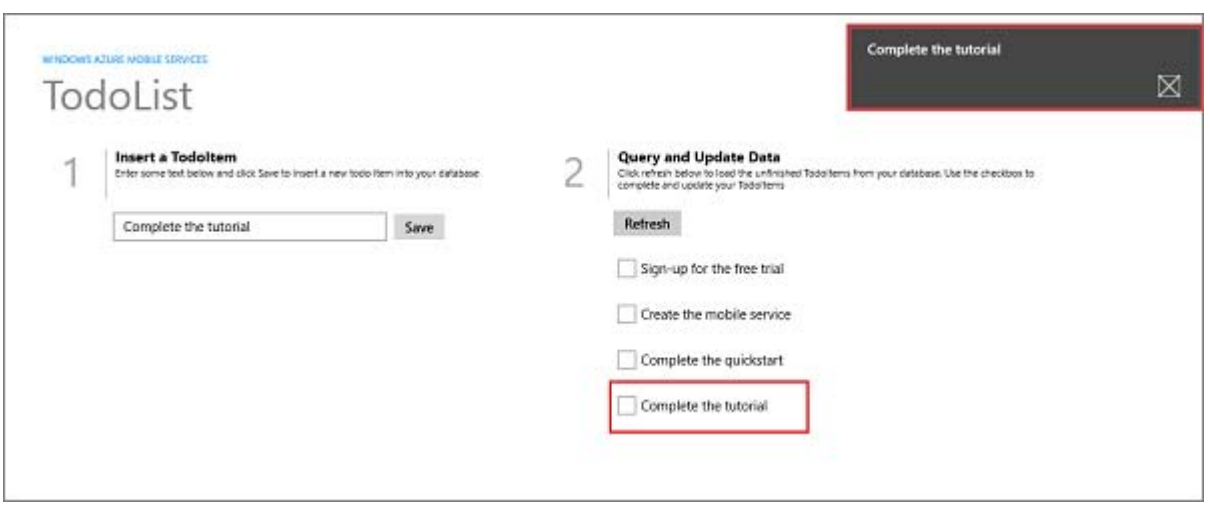
This registers a new insert script, which sends a push notification (the inserted text) to the channel provided in the insert request.

Test push notifications in your app

1. In Visual Studio, press the F5 key to run the app.
2. In the app, type text in **Insert a TodoItem**, and then click **Save**.



Note that after the insert completes, the app receives a push notification from WNS.



Next steps

In this simple example a user receives a push notification with the data that was just inserted. The channel used by WNS is supplied to the mobile service by the client in the request. In the next

tutorial, you will create a separate Channel table in which to store channel URIs and send a push notification out to all stored channels when an insert occurs.

Push notifications to users by using Mobile Services

This section extends the previous push notification tutorial by adding a new table to store Windows Push Notification Service (WNS) channel URIs. These channels can then be used to send push notifications to users of the Windows Store app.

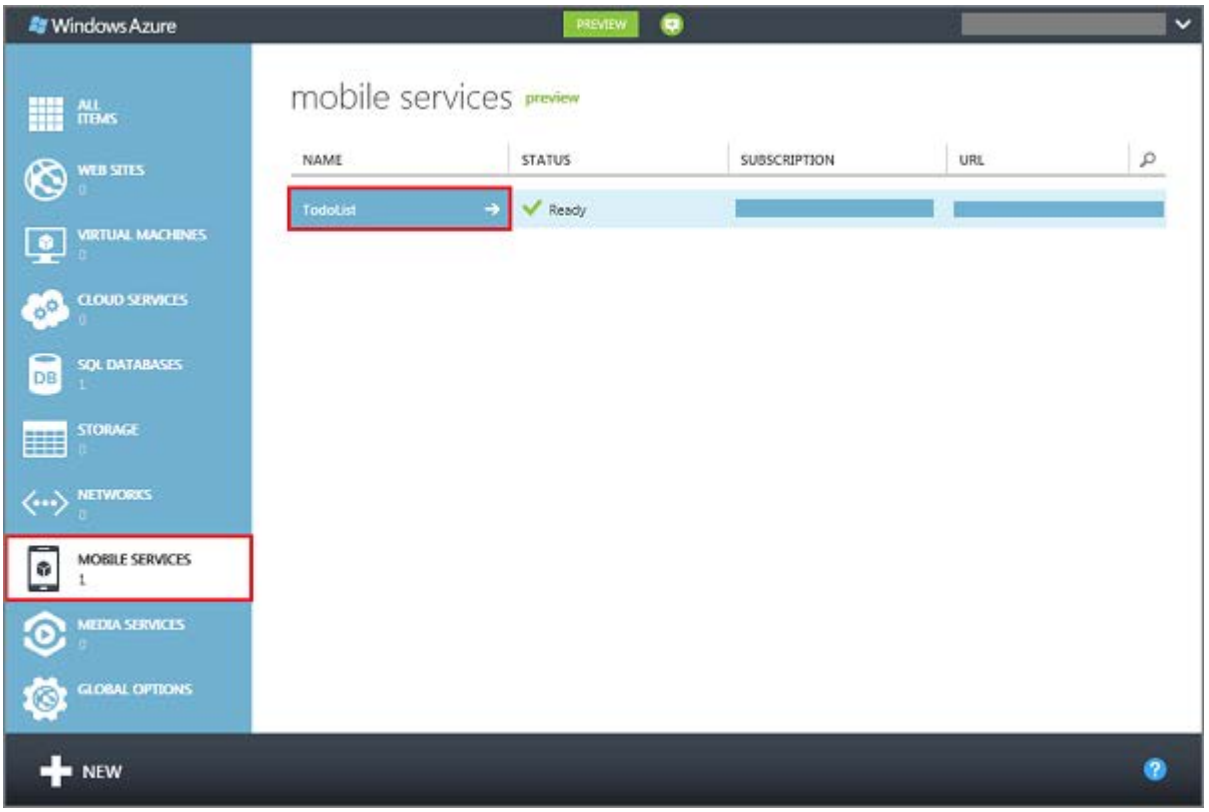
This tutorial walks you through these steps to update push notifications in your app:

1. Create the Channel table
2. Update the app
3. Update server scripts
4. Verify the push notification behavior

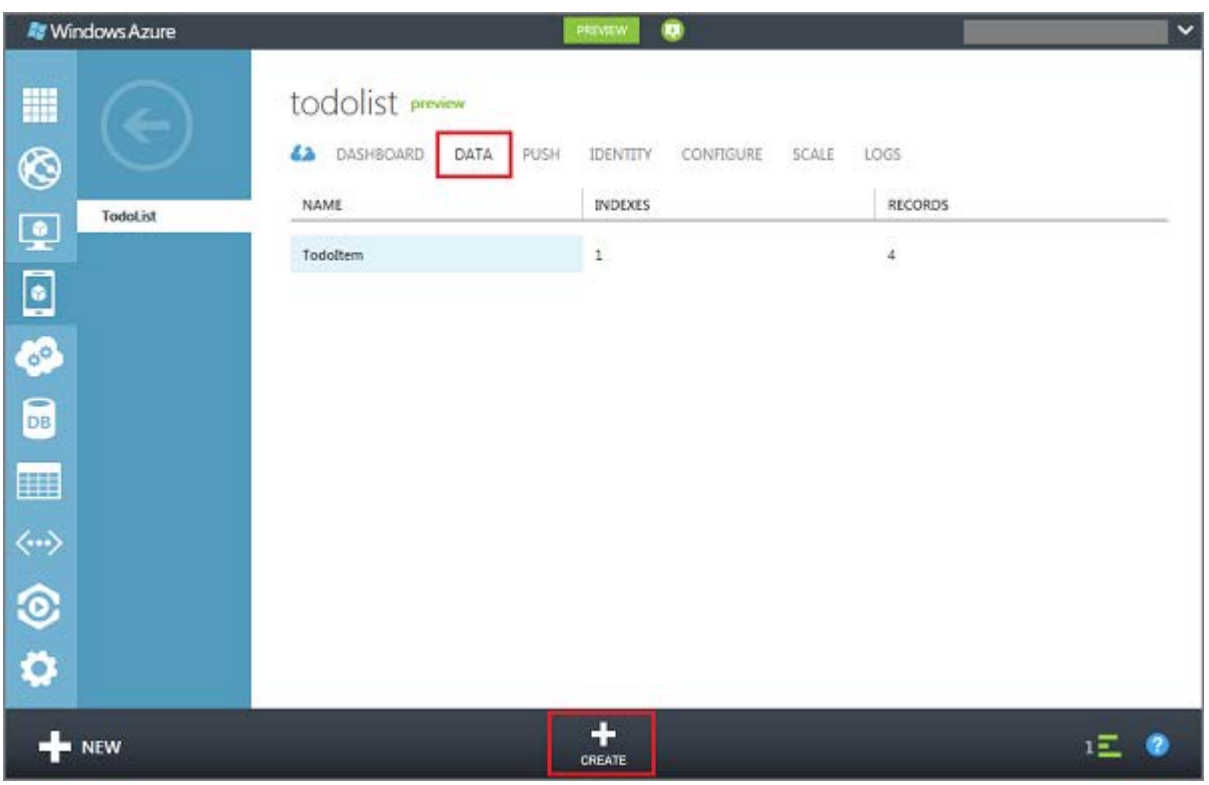
This tutorial is based on the Mobile Services quickstart and builds on the previous tutorial Get started with push notifications. Before you start this tutorial, you must first complete Get started with push notifications.

Create a new table

1. Log into the Windows Azure Management Portal, click **Mobile Services**, and then click your app.

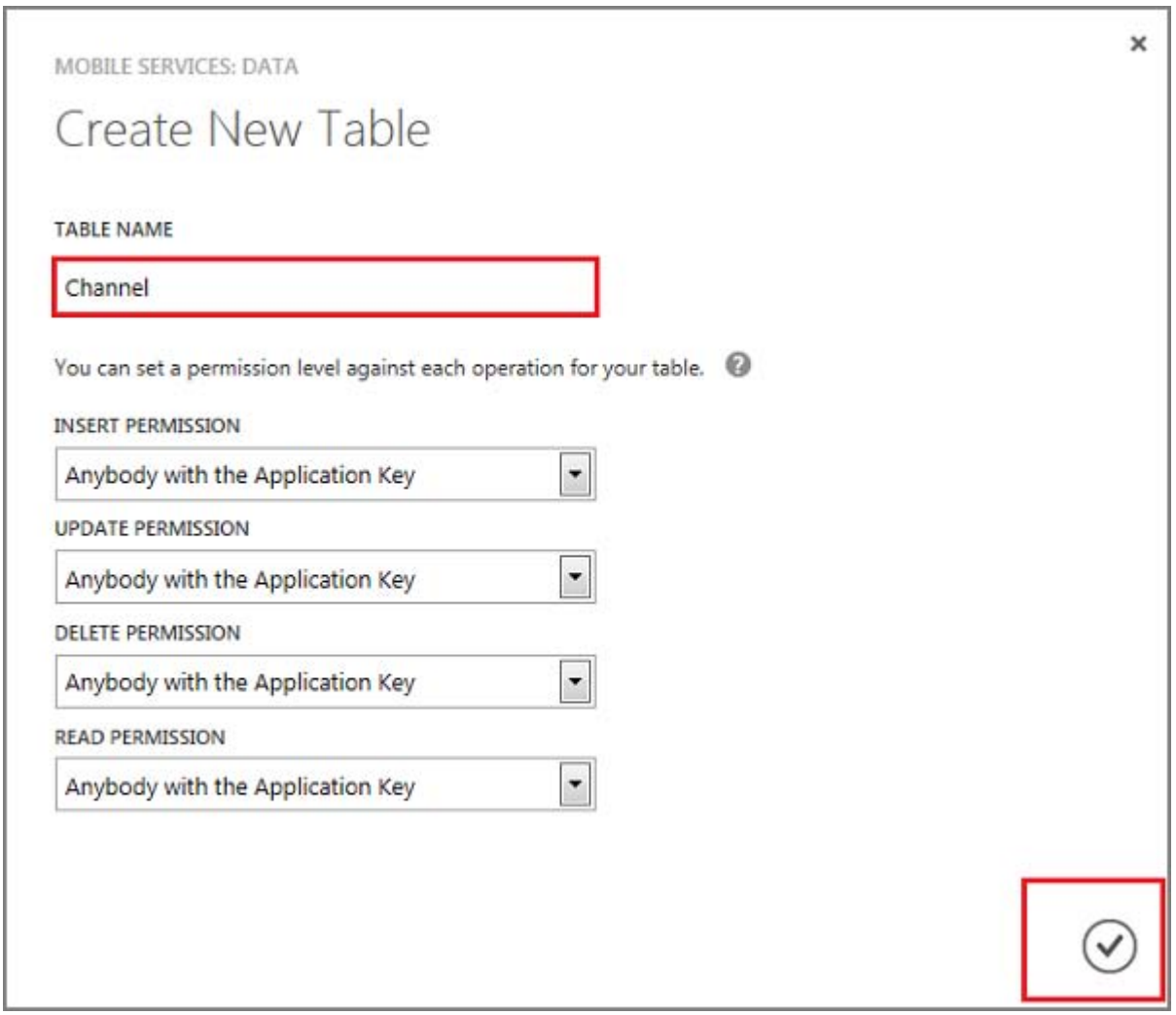


2. Click the **Data** tab, and then click **Create**.



This displays the **Create new table** dialog.

3. Keeping the default **Anybody with the application key** setting for all permissions, type *Channel* in **Table name**, and then click the check button.



This creates the **Channel** table, which stores the channel URIs used to send push notifications separate from item data.

Next, you will modify the push notifications app to store data in this new table instead of in the **TodoItem** table.

Update your app

1. In Visual Studio 2012 Express for Windows 8, open the project from the tutorial Get started with push notifications, open up file default.js, and replace the **click** event listener definition for **buttonSave** with the original version, as follows:

```
buttonSave.addEventListener("click", function () {
    insertTodoItem({
        text: textInput.value,
        complete: false
    });
});
```

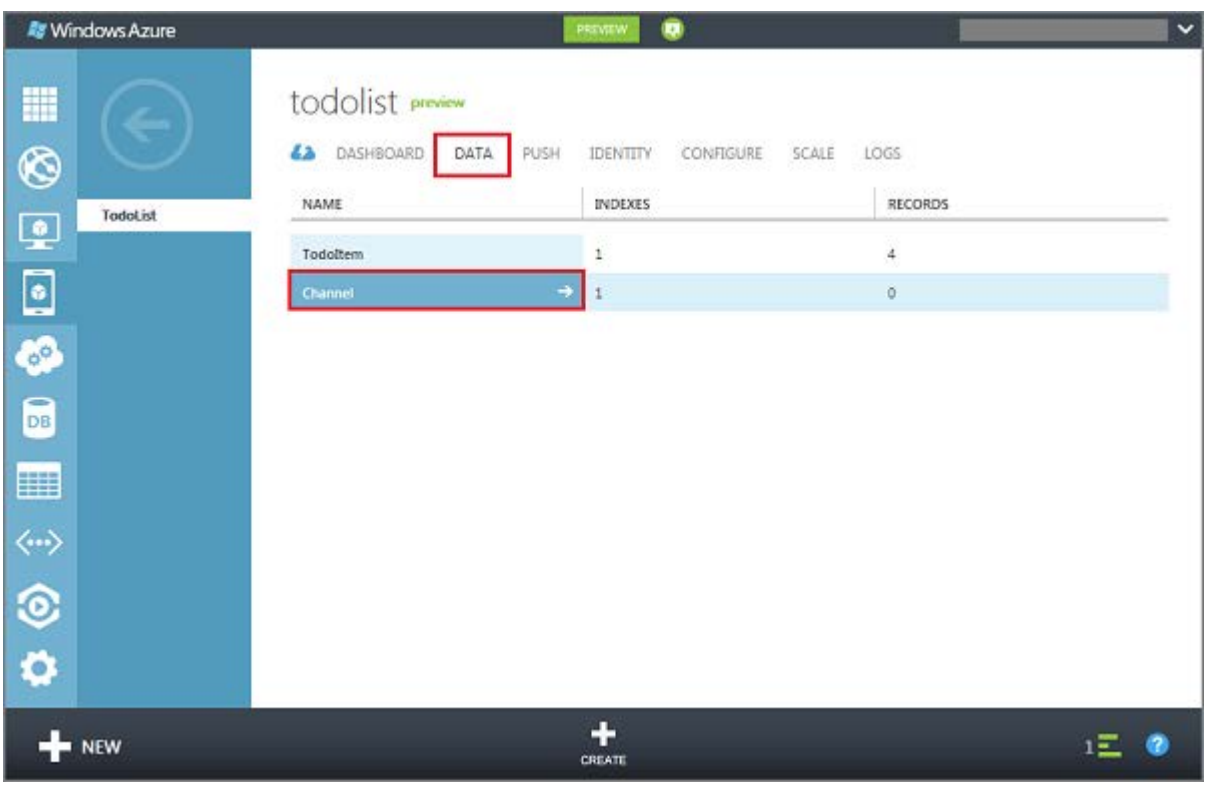
2. In default.js, insert the following code after the code that creates the MobileServiceClient instance:

```
// Insert the new channel URI into the Channel table.
var channelTable = client.getTable('Channel');
channelTable.insert({ uri: channel.uri });
```

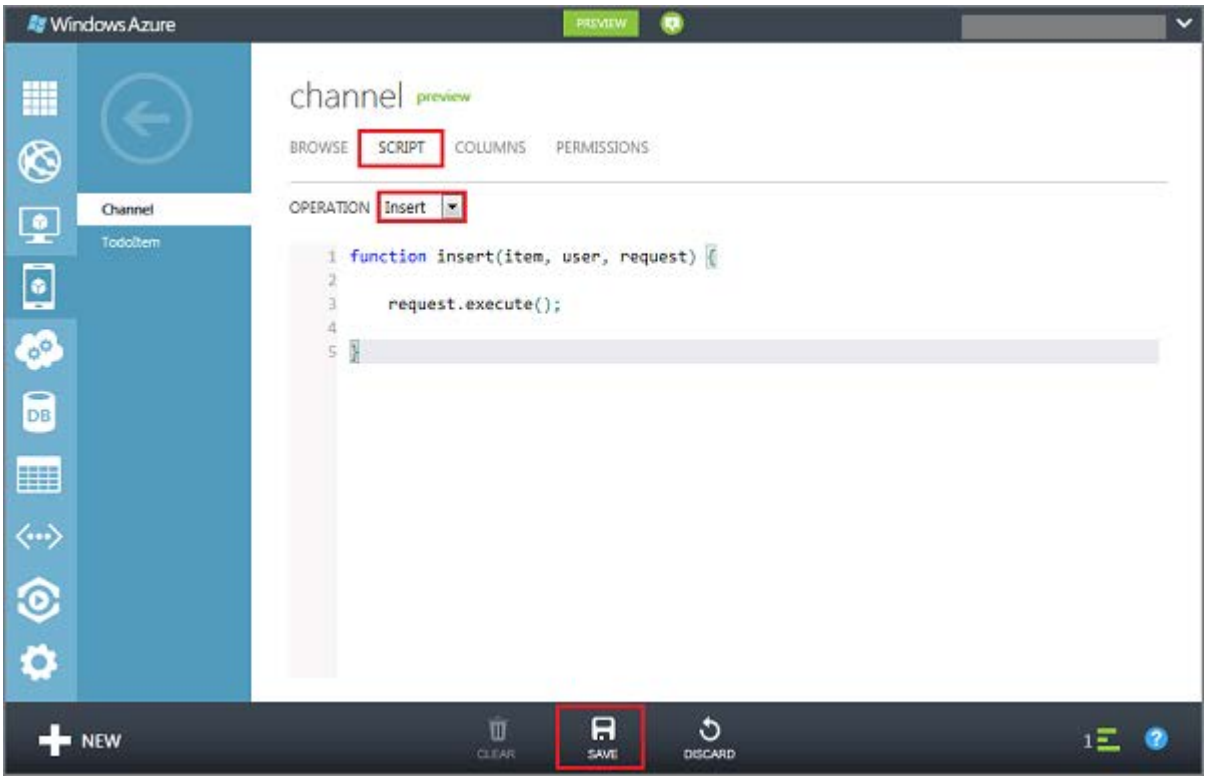
This code inserts the current channel into the Channel table.

Update server scripts

1. In the Management Portal, click the **Data** tab and then click the **Channel** table.



2. In **channel**, click the **Script** tab and select **Insert**.



This displays the function that is invoked when an insert occurs in the **Channel** table.

3. Replace the insert function with the following code, and then click **Save**:

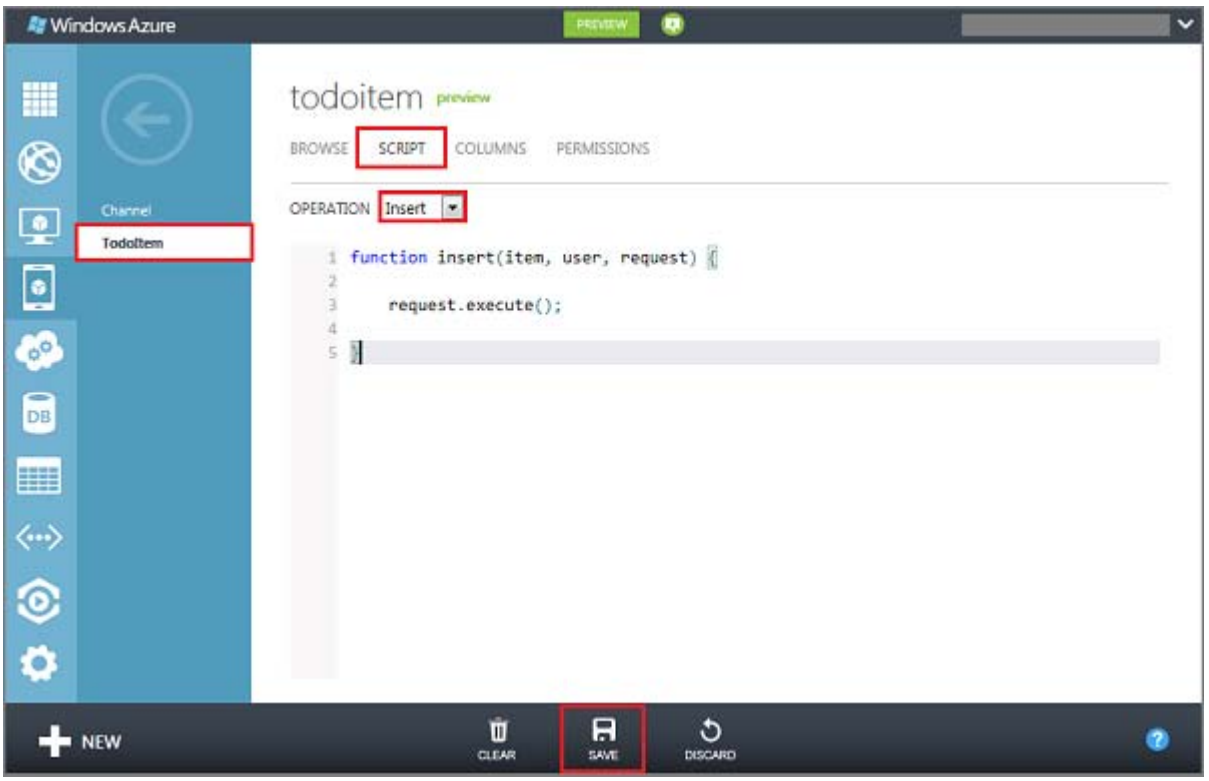
```
function insert(item, user, request) {
    var channelTable = tables.getTable('Channel');
    channelTable
        .where({ uri: item.uri })
        .read({ success: insertChannelIfNotFound });


function insertChannelIfNotFound(existingChannels) {
    if (existingChannels.length &gt; 0) {
        request.respond(200, existingChannels[0]);
    } else {
        request.execute();
    }
}


}
```

This script checks the **Channel** table for an existing channel with the same URI. The insert only proceeds if no matching channel was found. This prevents duplicate channel records.

4. Click **TodoItem**, click **Script** and select **Insert**.



5. Replace the insert function with the following code, and then click **Save**:

```
function insert(item, user, request) {
    request.execute({
        success: function() {
            request.respond();
            sendNotifications();
        }
    });


function sendNotifications() {
    var channelTable = tables.getTable('Channel');
    channelTable.read({
        success: function(channels) {
            channels.forEach(function(channel) {
                push.wns.sendToastText04(channel.uri, {
```

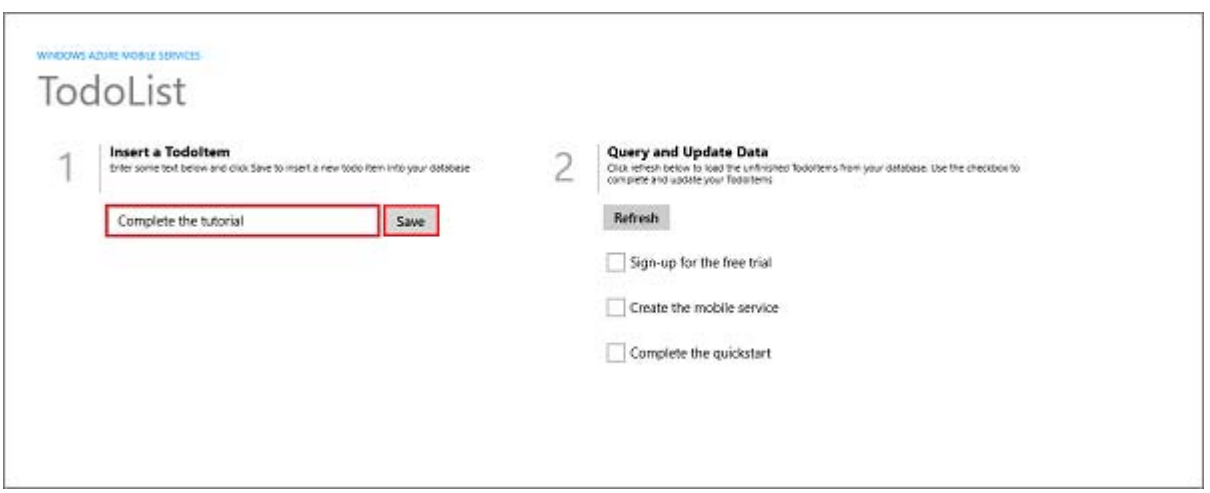
```
                    text1: item.text
                }, {
                    success: function(pushResponse) {
                        console.log("Sent push:", pushResponse);
                    }
                });
            });
        }
    });
}


}
```

This insert script sends a push notification (with the text of the inserted item) to all channels stored in the **Channel** table.

Test the app

1. In Visual Studio, press the F5 key to run the app.
2. In the app, type text in **Insert a TodoItem**, and then click **Save**.



Note that after the insert completes, the app still receives a push notification from WNS.

3. (Optional) Run the app on two machines at the same time, and repeat the previous step.

   The notification is sent to all running app instances.

Learn more about Mobile Services

This concludes the tutorials that demonstrate the basics of working with Mobile Services. To learn more about Mobile Services, browse to the following web sites:

Mobile Services developer center (http://www.windowsazure.com/en-us/develop/mobile/)
Includes links to all relevant information about Mobile Services.

Mobile Services forums (http://social.msdn.microsoft.com/Forums/en-US/azuremobile/threads)
Find the latest questions and answers about Mobile Services in the Windows Azure platform forums.

Mobile Services client SDK for Windows 8 (http://aka.ms/zumosdk)
Download location for the Mobile Services client SDK for Windows Store apps.

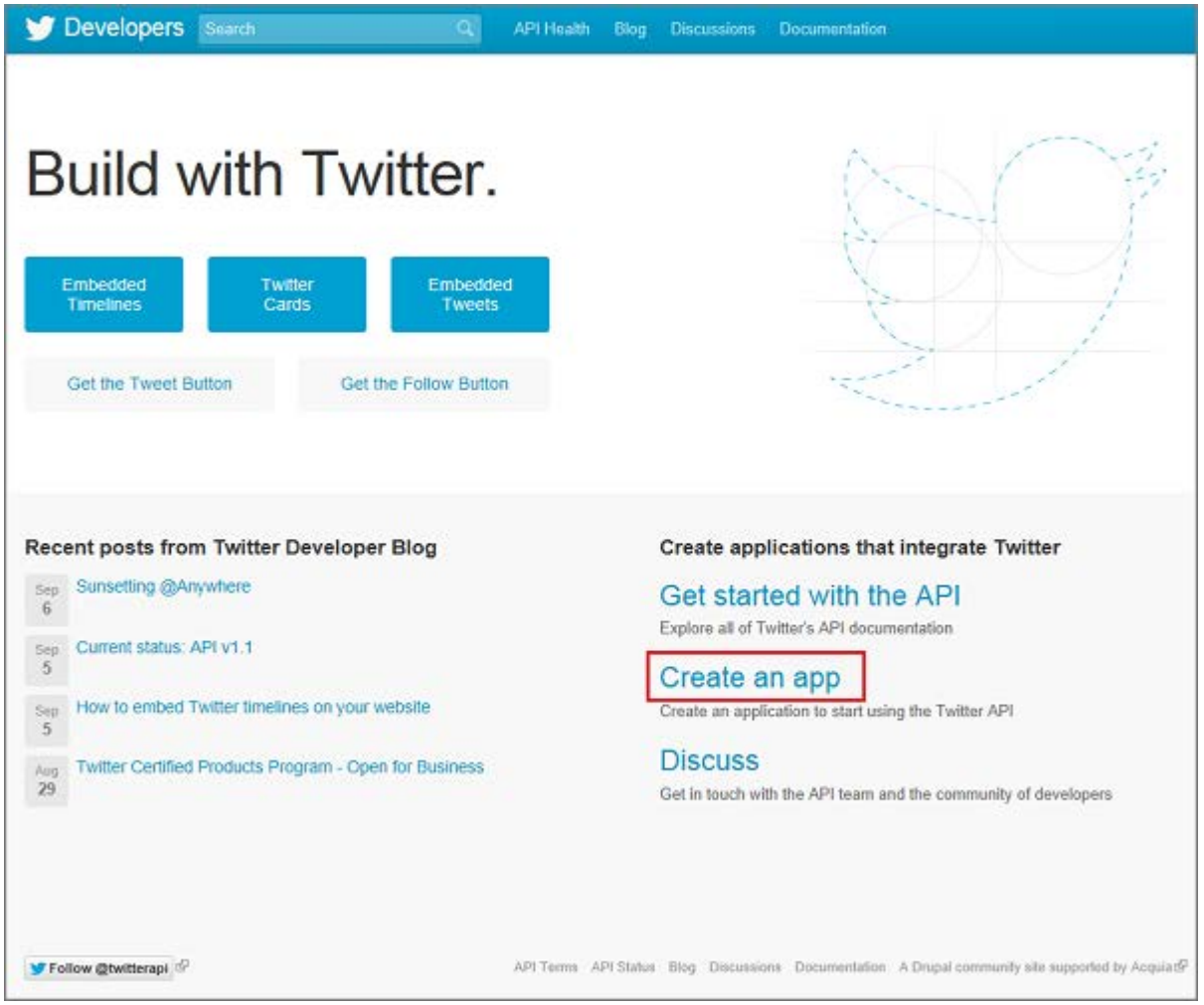
Mobile Services technical references (http://aka.ms/zumodocs)
Reference documentation for Mobile Services client libraries and server scripts.

Appendix A: Register your apps for Twitter login with Mobile Services

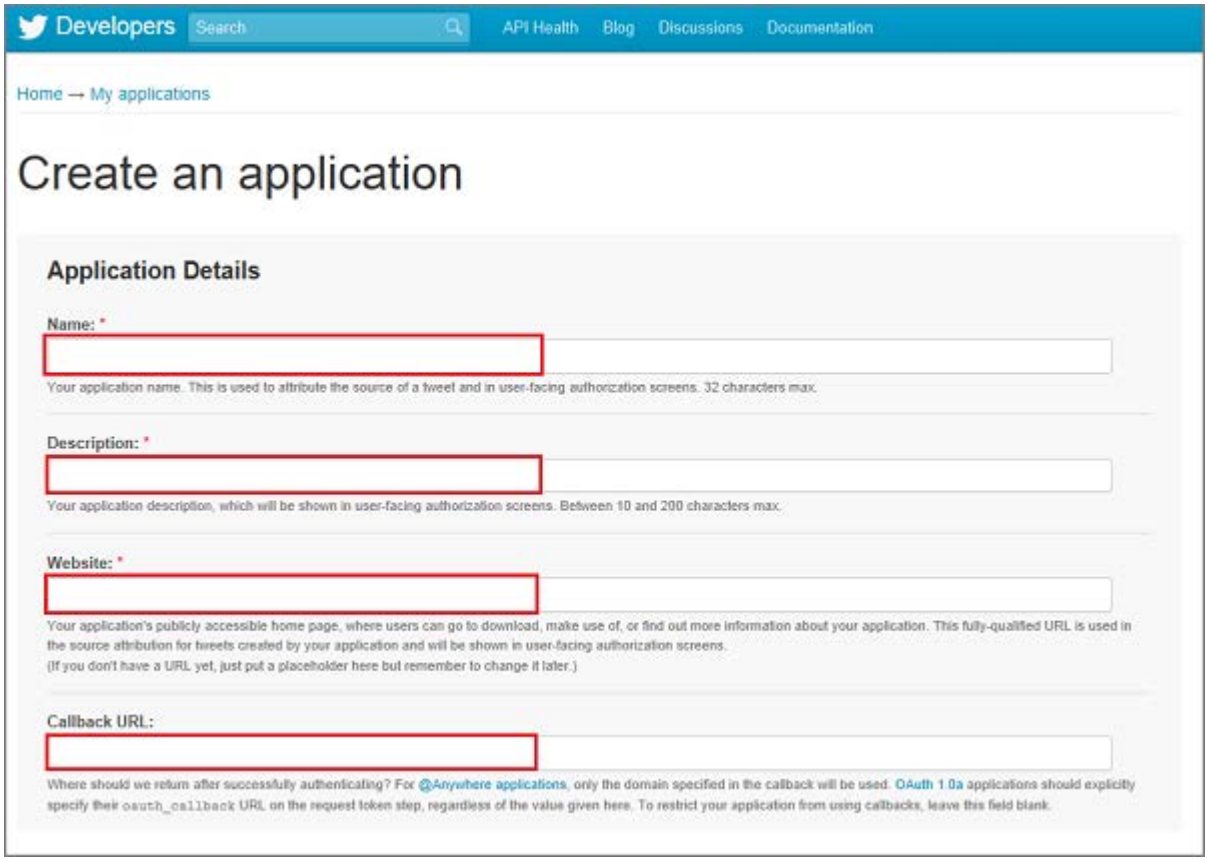
This appendix shows you how to register your apps to be able to use Twitter to authenticate with Windows Azure Mobile Services.

> **Note:** To complete the procedure in this topic, you must have a Twitter account that has a verified email address. To create a new Twitter account, go to twitter.com.

1. Navigate to the Twitter Developers web site, sign-in with your Twitter account credentials, and then click **Create an app**.

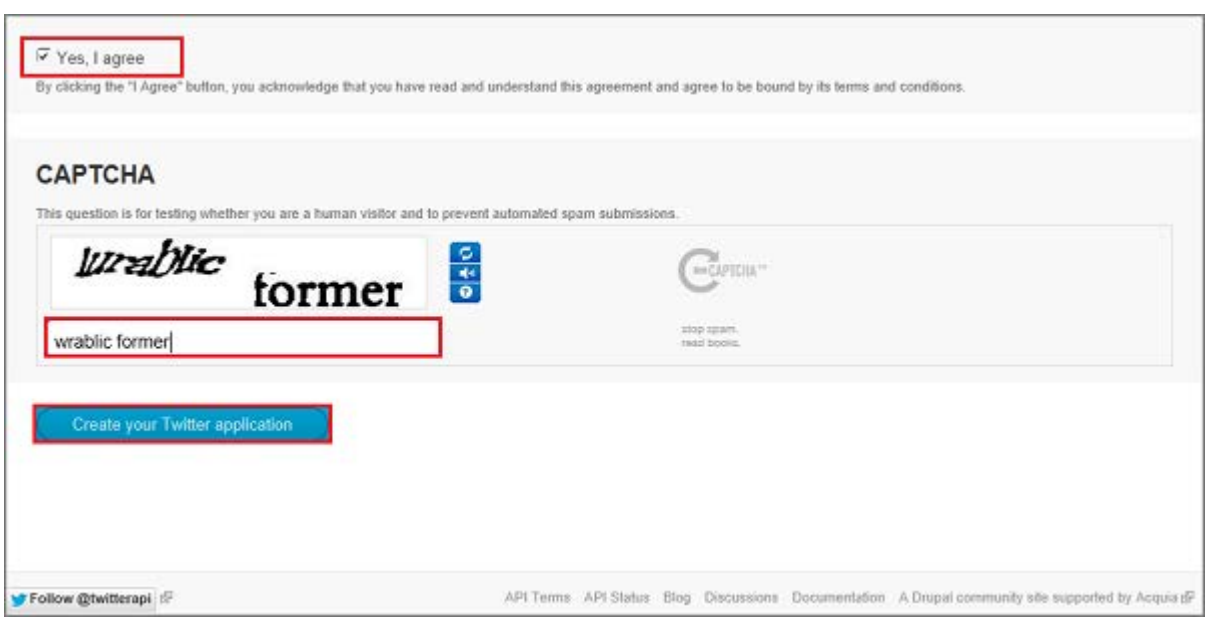


2. Type the **Name**, **Description**, and **Website** values for your app, and type the URL of the mobile service in **Callback URL**.



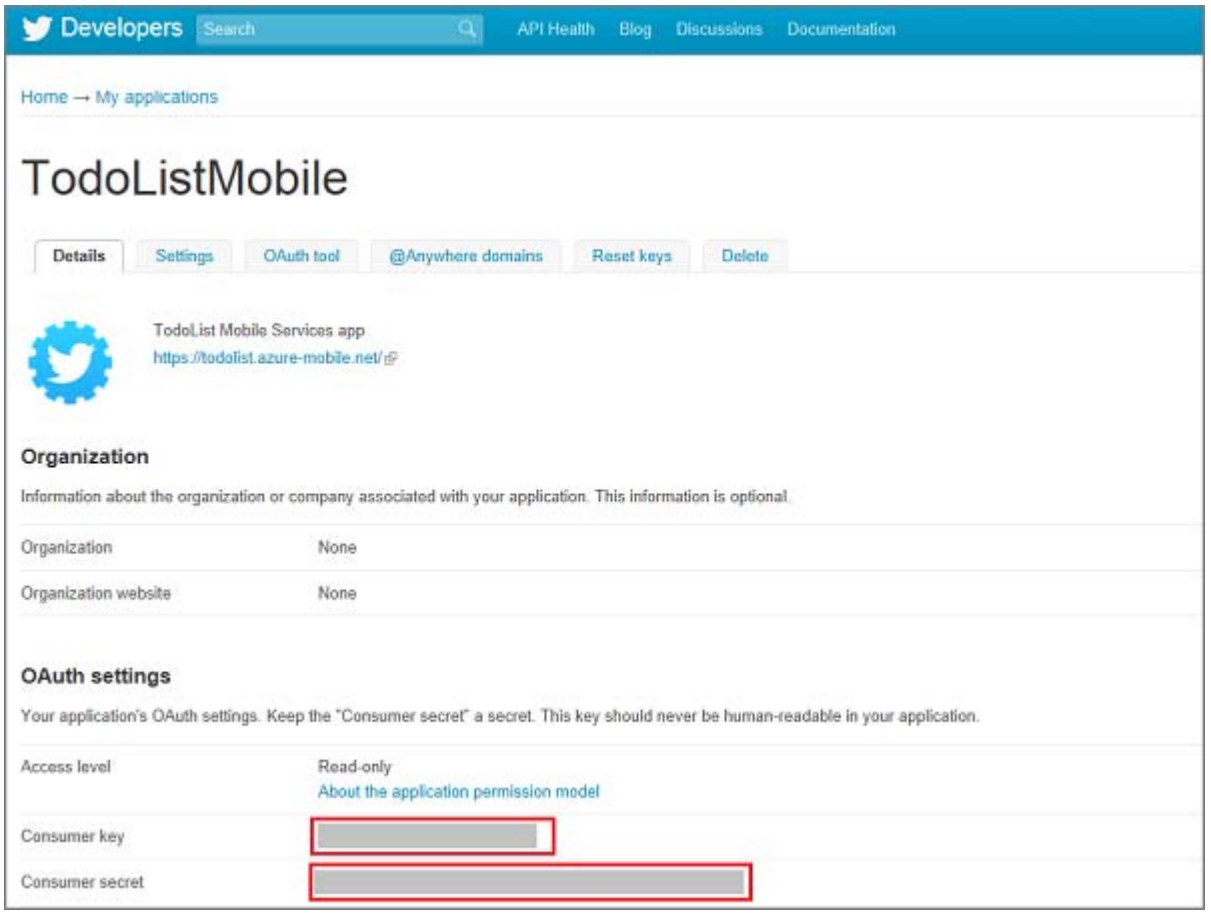
Note: The **Website** value is required but is not used.

3. At the bottom the page, read and accept the terms, type the correct CAPTCHA words, and then click **Create your Twitter application**.



This registers the app displays the application details.

4. Make a note of the values of **Consumer key** and **Consumer secret**.



Security Note: The consumer secret is an important security credential. Do not share this secret with anyone or distribute it with your app.

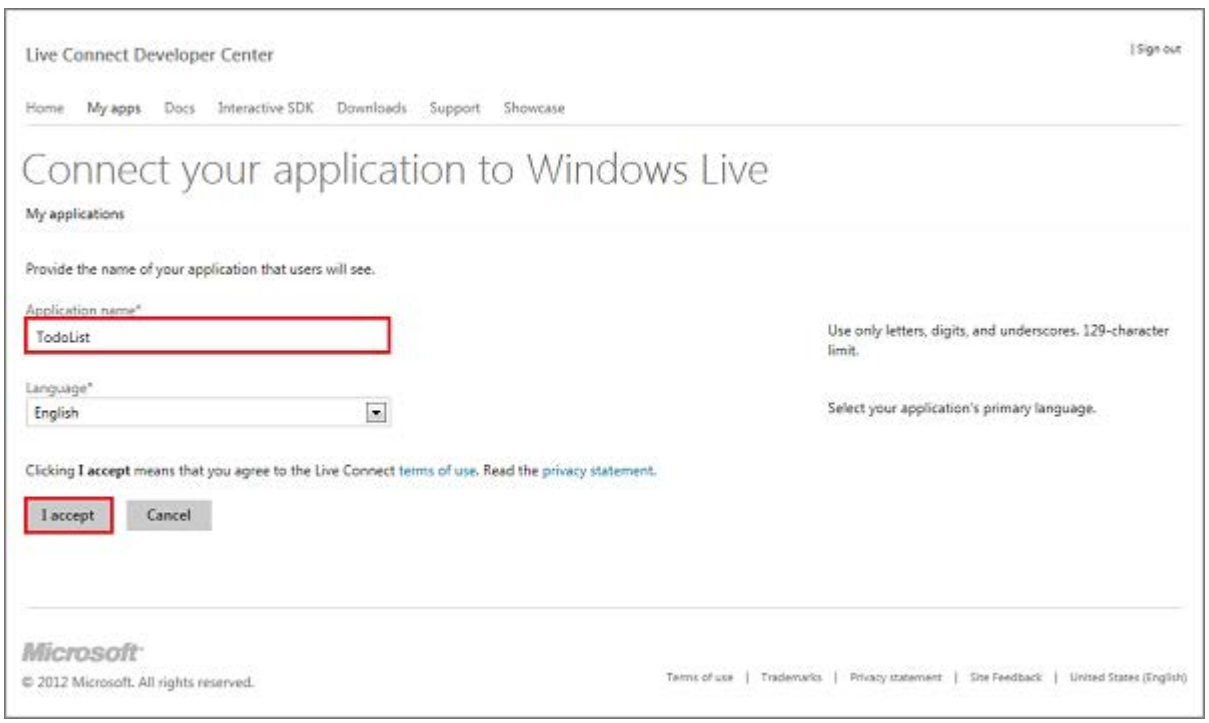
You are now ready to use a Twitter login for authentication in your app by providing the consumer key and consumer secret values to Mobile Services.

Appendix B: Register your Windows Store apps to use a Microsoft Account login

This topic shows you how to register your apps to be able to use Live Connect as an authentication provider for Windows Azure Mobile Services.

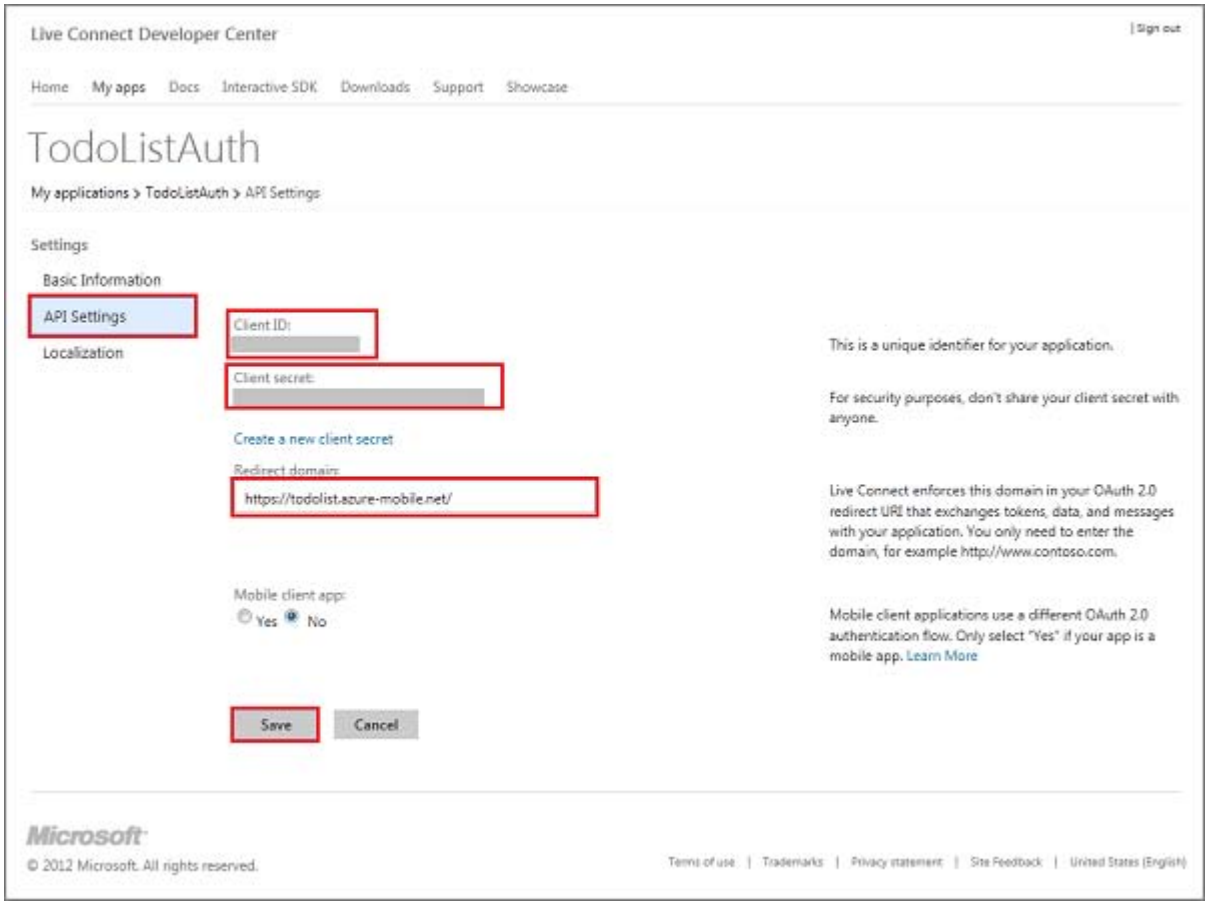
> **Note:** When you intend to also provide single sign-on or push notifications from a Windows Store app, consider also registering your app with the Windows Store. For more information, see Register your Windows Store apps for Windows Live Connect authentication.

1. Navigate to the My Applications page in the Live Connect Developer Center, and log on with your Microsoft account, if required.
2. Click **Create application**, then type an **Application name** and click **I accept**.



   This registers the application with Live Connect.

3. Click **Application settings page**, then **API Settings** and make a note of the values of the **Client ID** and **Client secret**.



Security Note: The client secret is an important security credential. Do not share the client secret with anyone or distribute it with your app.

4. In **Redirect domain**, enter the URL of your mobile service, and then click **Save**.

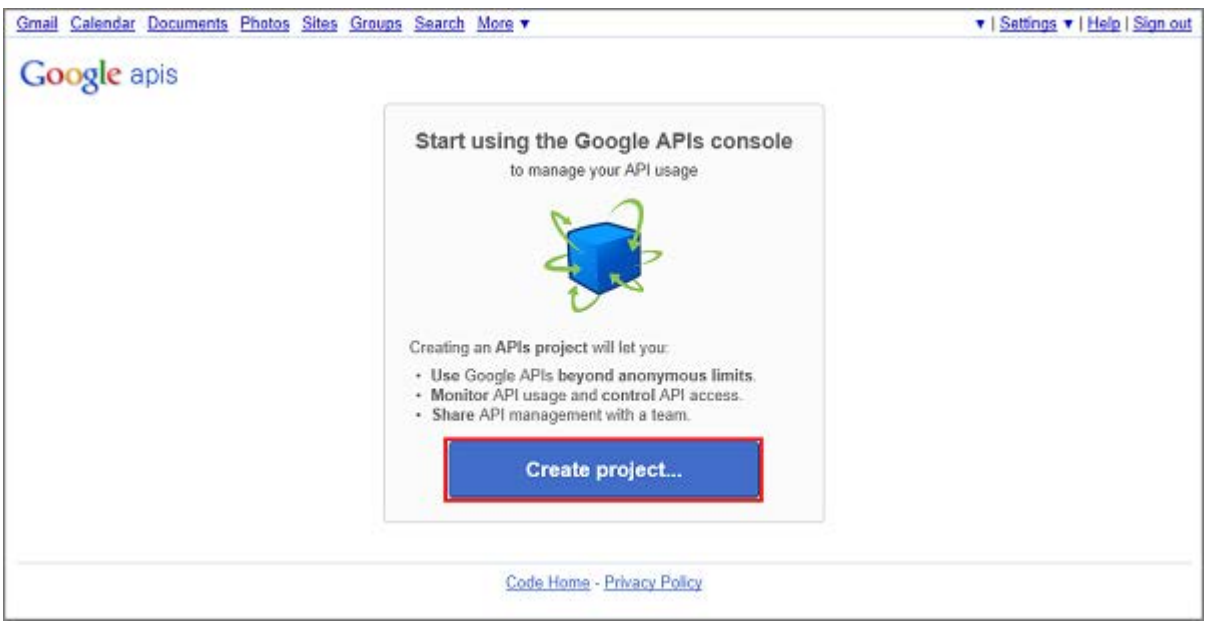
You are now ready to use a Microsoft Account for authentication in your app by providing the client ID and client secret values to Mobile Services.

Appendix C: Register your apps for Google login with Mobile Services

This topic shows you how to register your apps to be able to use Google to authenticate with Windows Azure Mobile Services.

> **Note:** To complete the procedure in this topic, you must have a Google account that has a verified email address. To create a new Google account, go to accounts.google.com.

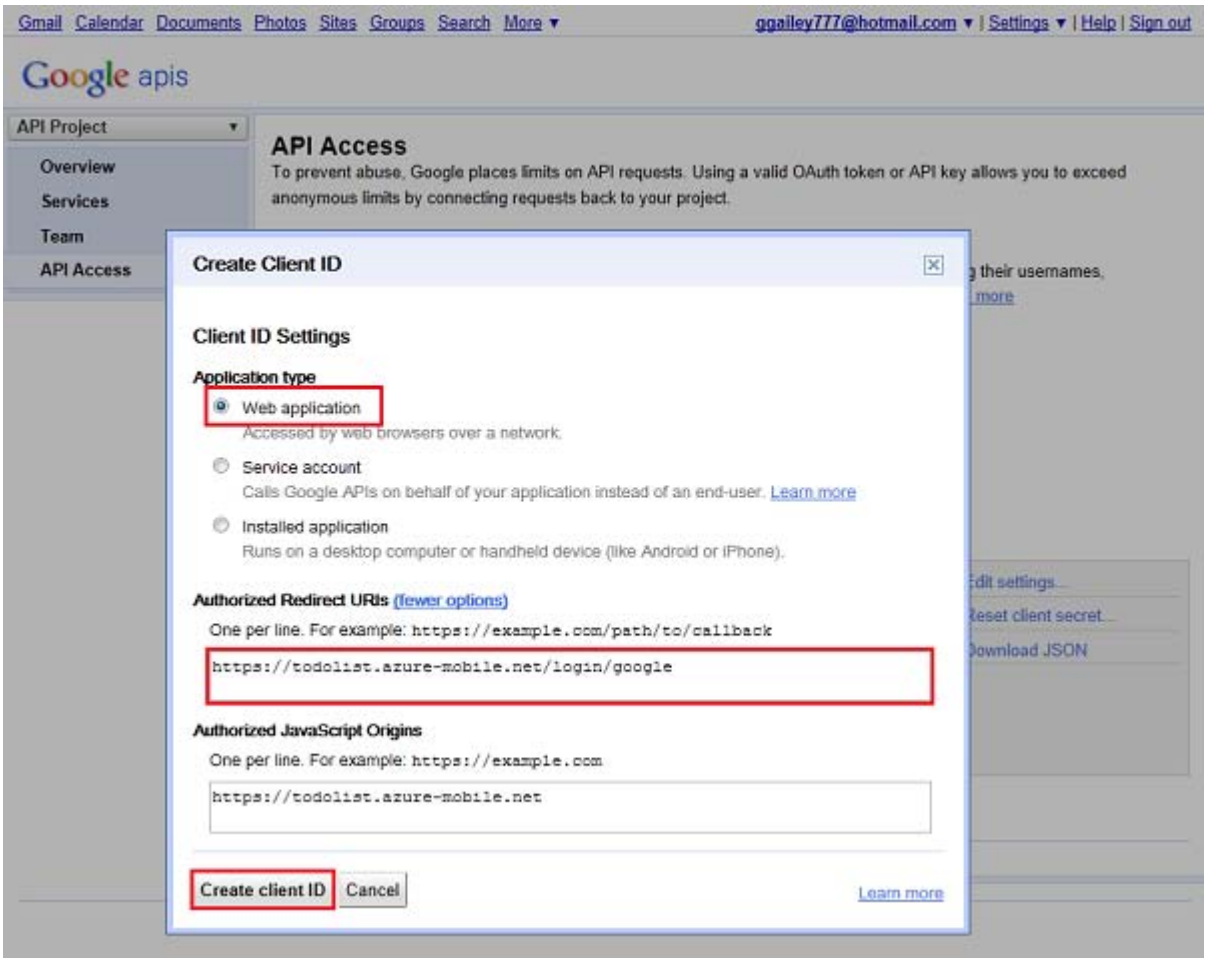
1. Navigate to the Google apis web site, sign-in with your Google account credentials, and then click **Create project...**.



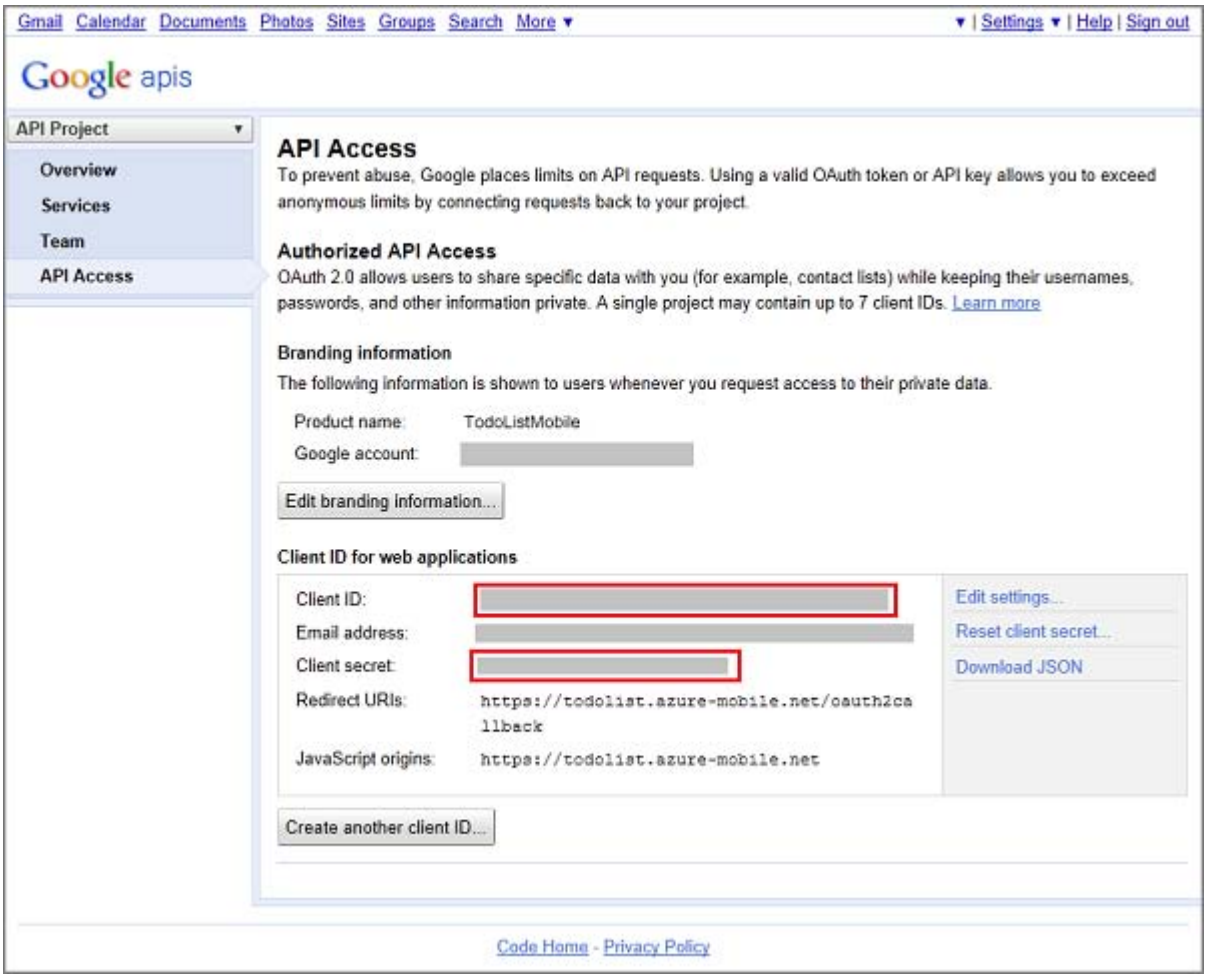
2. Click **API Access** and then click **Create an OAuth 2.0 client ID...**.

3. Under **Branding Information**, type your **Product name**, then click **Next**.

4. Under **Client ID Settings**, select **Web application**, type your mobile service URL in **Your site or hostname**, click **more options**, replace the generated URL in **Authorized Redirect URIs** with the URL of your mobile service appended with the path */login/google*, and then click **Create client ID**.



5. Under **Client ID for web applications**, make a note of the values of **Client ID** and **Client secret**.



Security Note: The client secret is an important security credential. Do not share this secret with anyone or distribute it with your app.

You are now ready to use a Google login for authentication in your app by providing the client ID and client secret values to Mobile Services.