Using Windows Azure Mobile Services to Cloud-Enable Your iOS Apps

Windows Azure Developer Center

Step-by-Step

Microsoft

Using Windows Azure Mobile Services to Cloud-Enable your iOS Apps

Windows Azure Developer Center

Summary: This topic shows you how to use Windows Azure Mobile Services to leverage data in an iOS 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

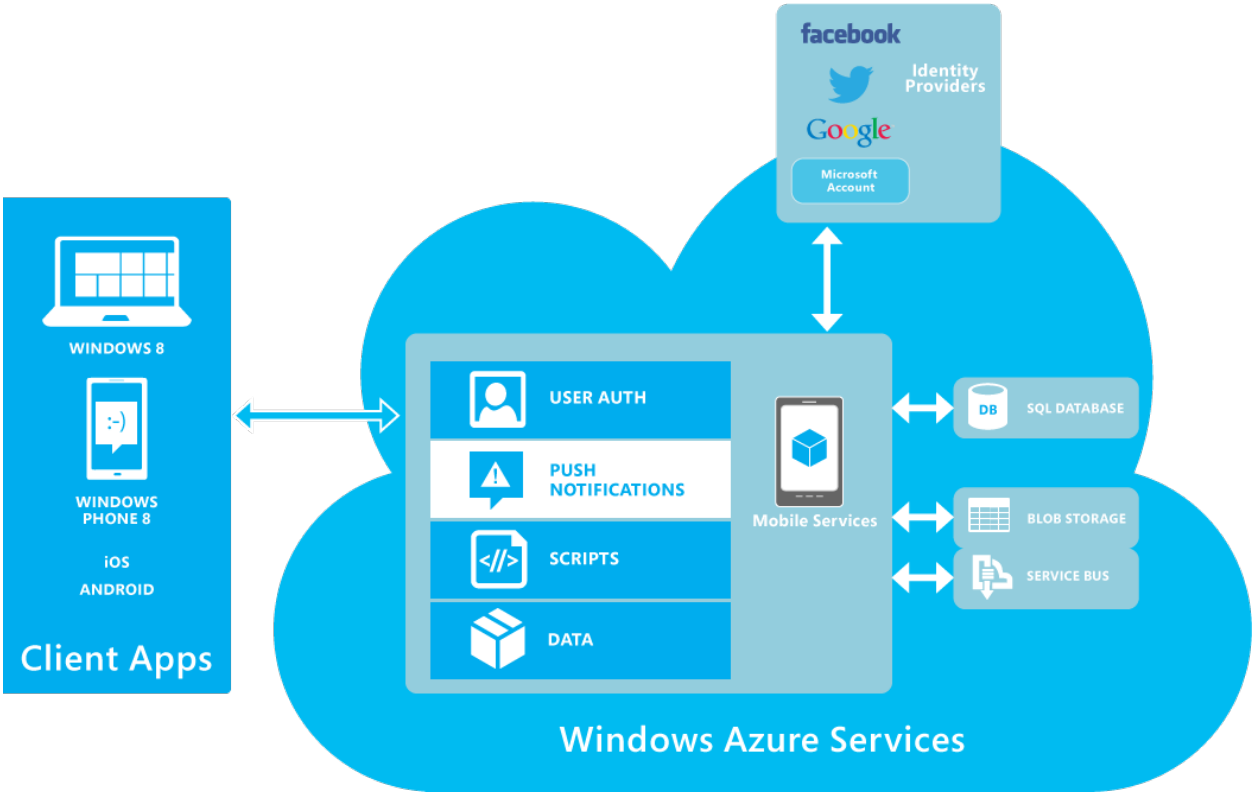
Introducing Windows Azure Mobile Services

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 topic shows you how to use Windows Azure Mobile Services to leverage data in an iOS 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 an iOS 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 iOS 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 iOS SDK and XCode 4.5 and iOS 5.0 or later versions.

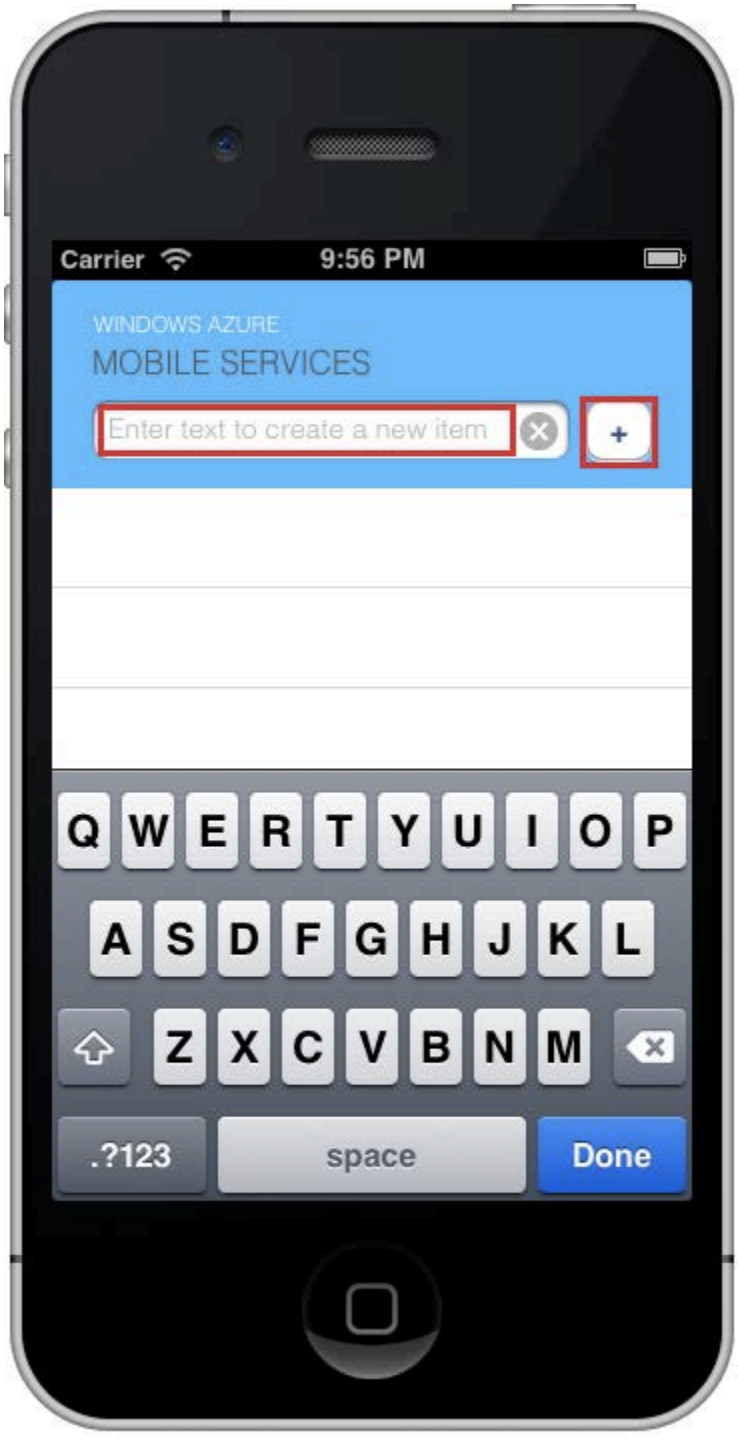
Download the projectDownload the GetStartedWithData project

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

1. Download the GetStartedWithData sample app from GitHub.
2. In Xcode, open the downloaded project and examine the TodoService.m file.

   Notice that there are eight // TODO comments that specify the steps you must take to make this app work with your mobile service.

3. Press the **Run** button (or the Command+R key) to rebuild the project and start the app.
4. In the app, type some text in the text box, then click the **+** button.



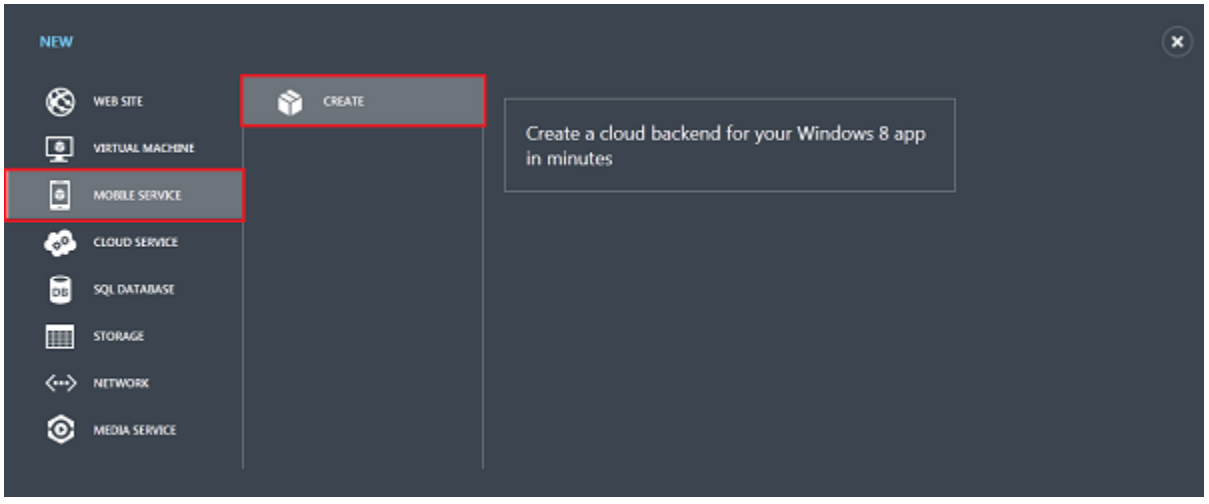
Notice that the saved text is displayed in the list below.

Create mobile serviceCreate 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.

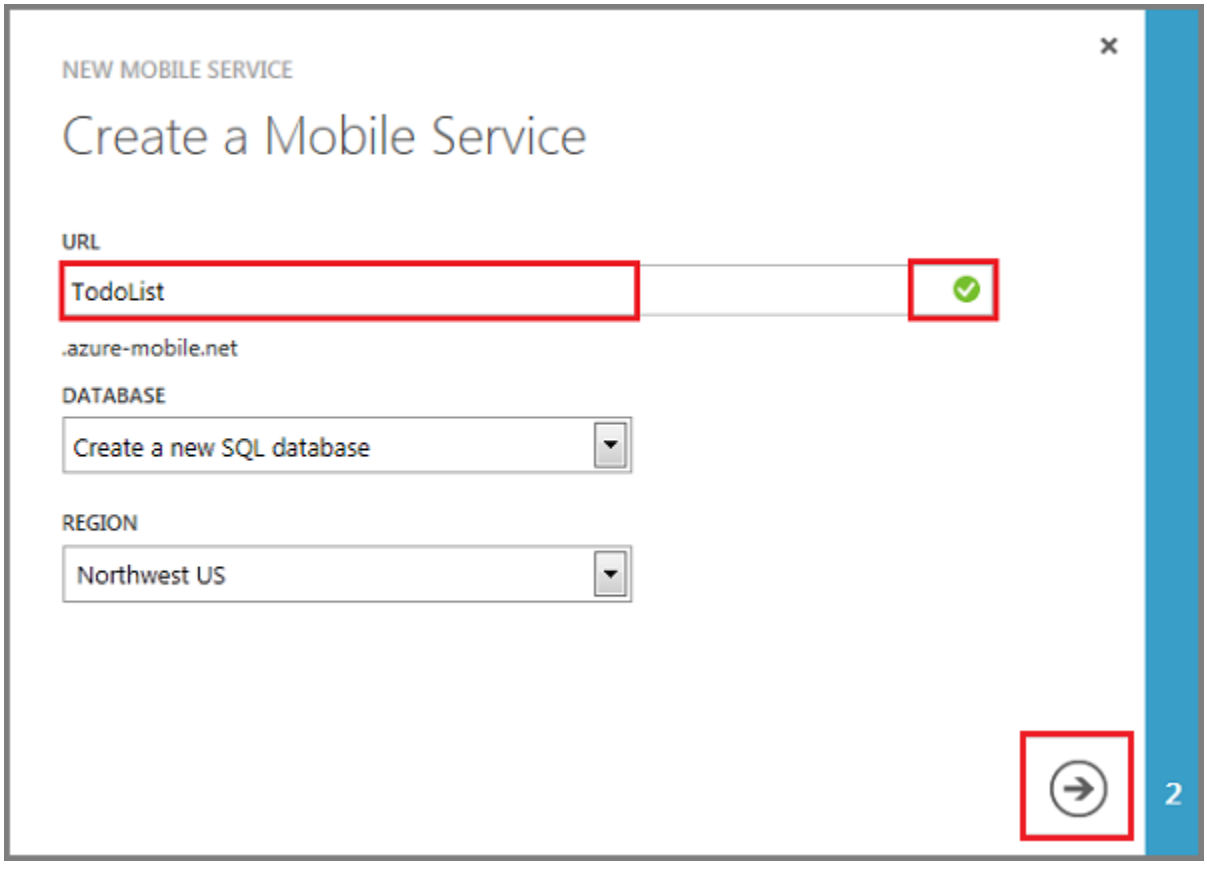
1. Log into the Windows Azure Management Portal.
2. At the bottom of the navigation pane, click **+NEW**.

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



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

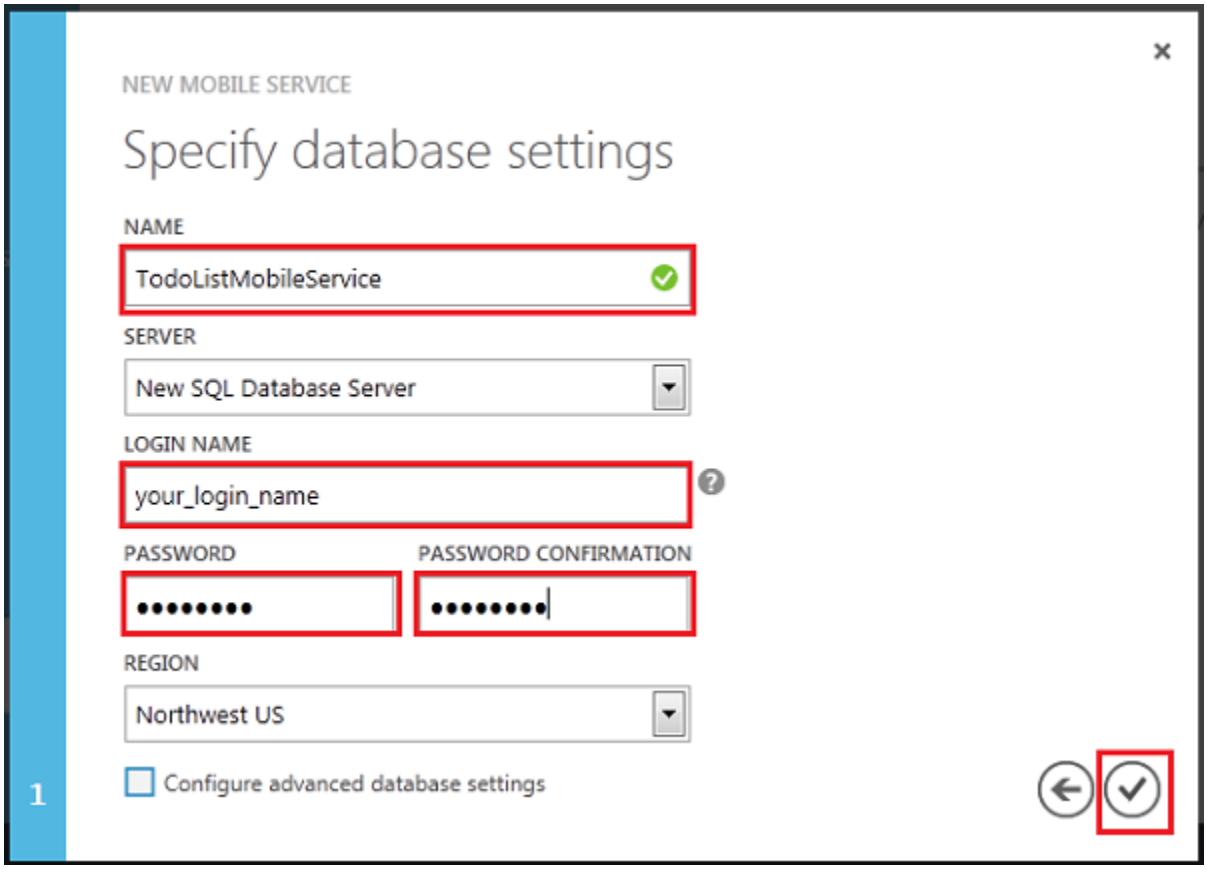
4. In the **Create a mobile service** page, type a subdomain name for the new mobile service in the **URL** textbox and wait for name verification. Once name verification completes, click the right arrow button to go to the next page.



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.

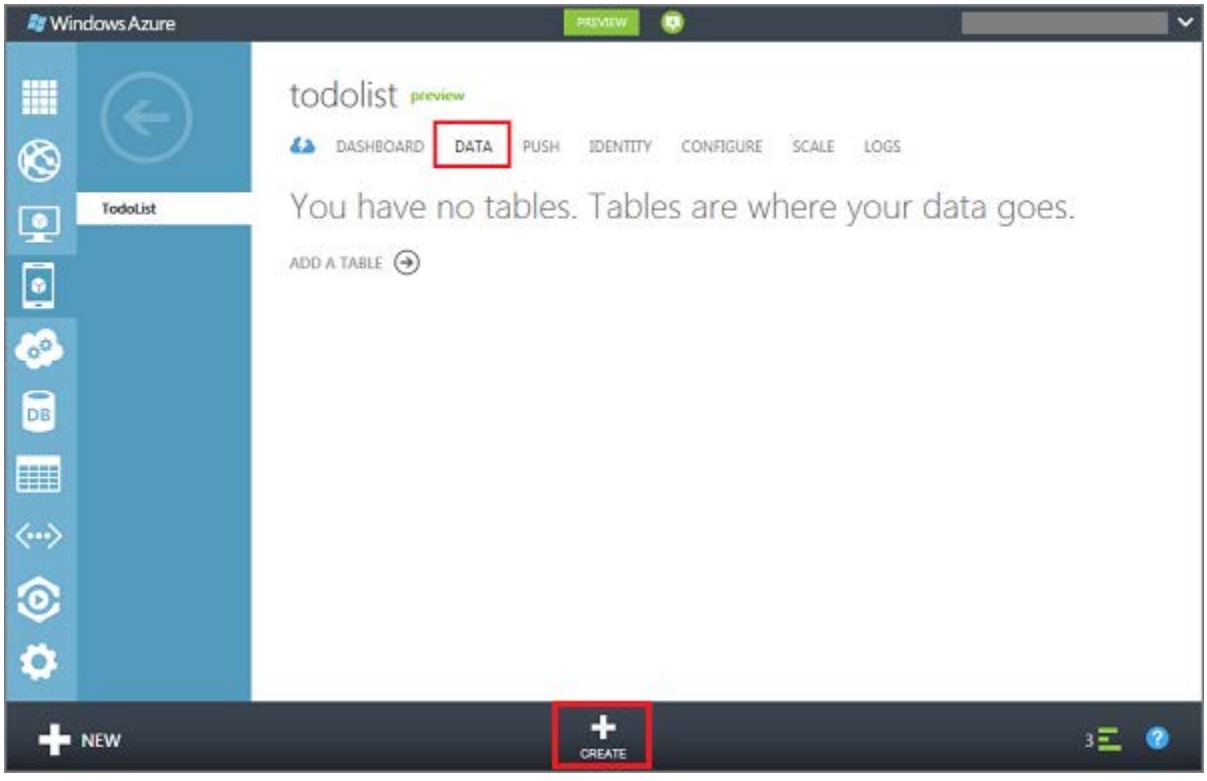
You have now created a new mobile service that can be used by your mobile apps. Next, you will add a new table in which to store app data. This table will be used by the app in place of the in-memory collection.

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.

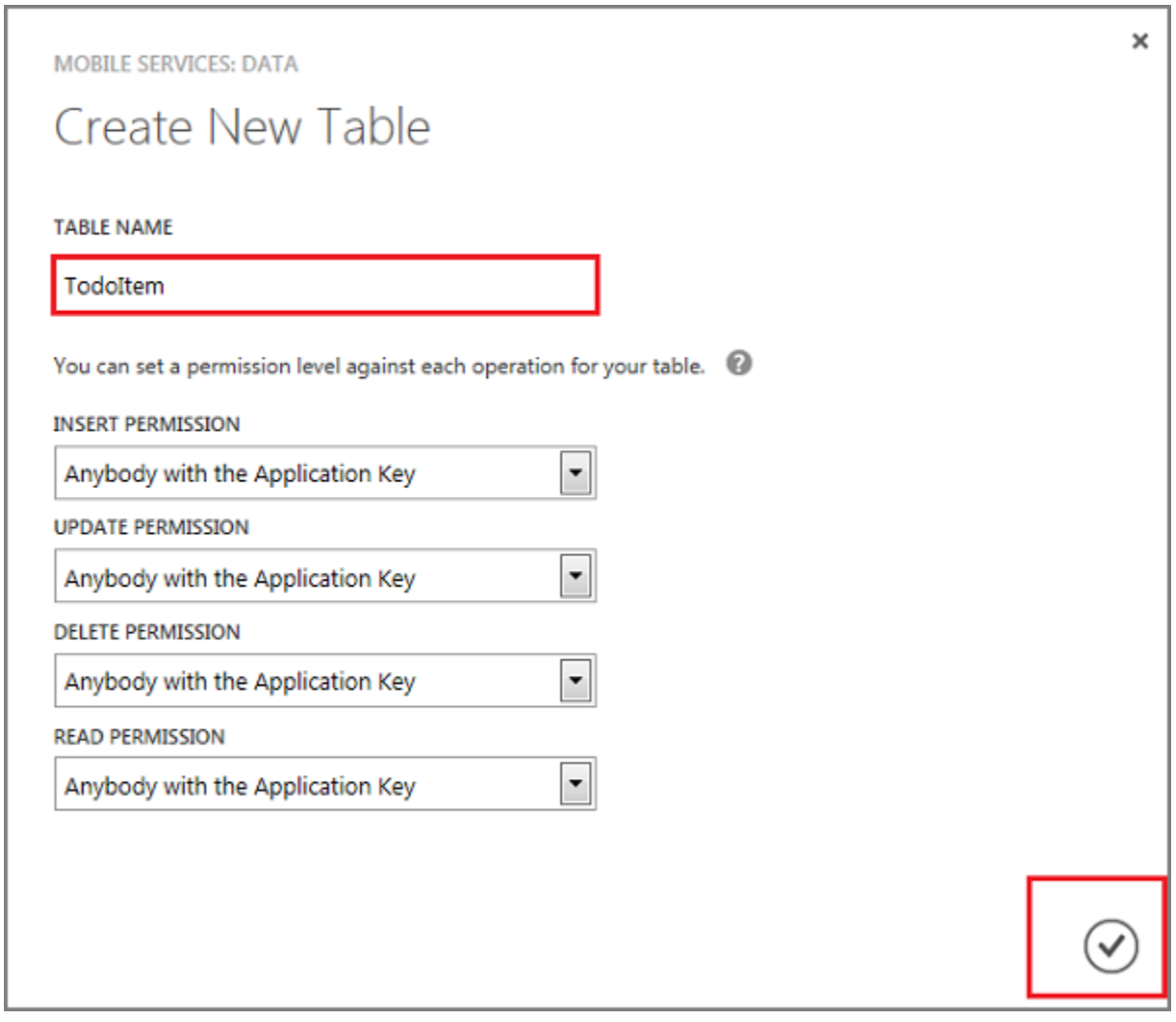
1. In the Management Portal, click **Mobile Services**, and then click the mobile service that you just created.

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 iOS SDK, install it now.
2. In the Project Navigator in Xcode, open both the TodoService.m and TodoService.h files located in the Quickstart folder, and add the following import statement:

```
#import <WindowsAzureMobileServices/WindowsAzureMobileServices.h>
```

3. In the ToDoService.h file, locate the following commented line of code:

```
// Create an MSClient property comment in the #interface declaration for
the TodoService.
```

After this comment, add the following line of code:

```
@property (nonatomic, strong)   MSClient *client;
```

This creates a property that represents the MSClient that connects to the service

4. In the file TodoService.m, locate the following commented line of code:

```
// Create an MSTable property for your items.
```

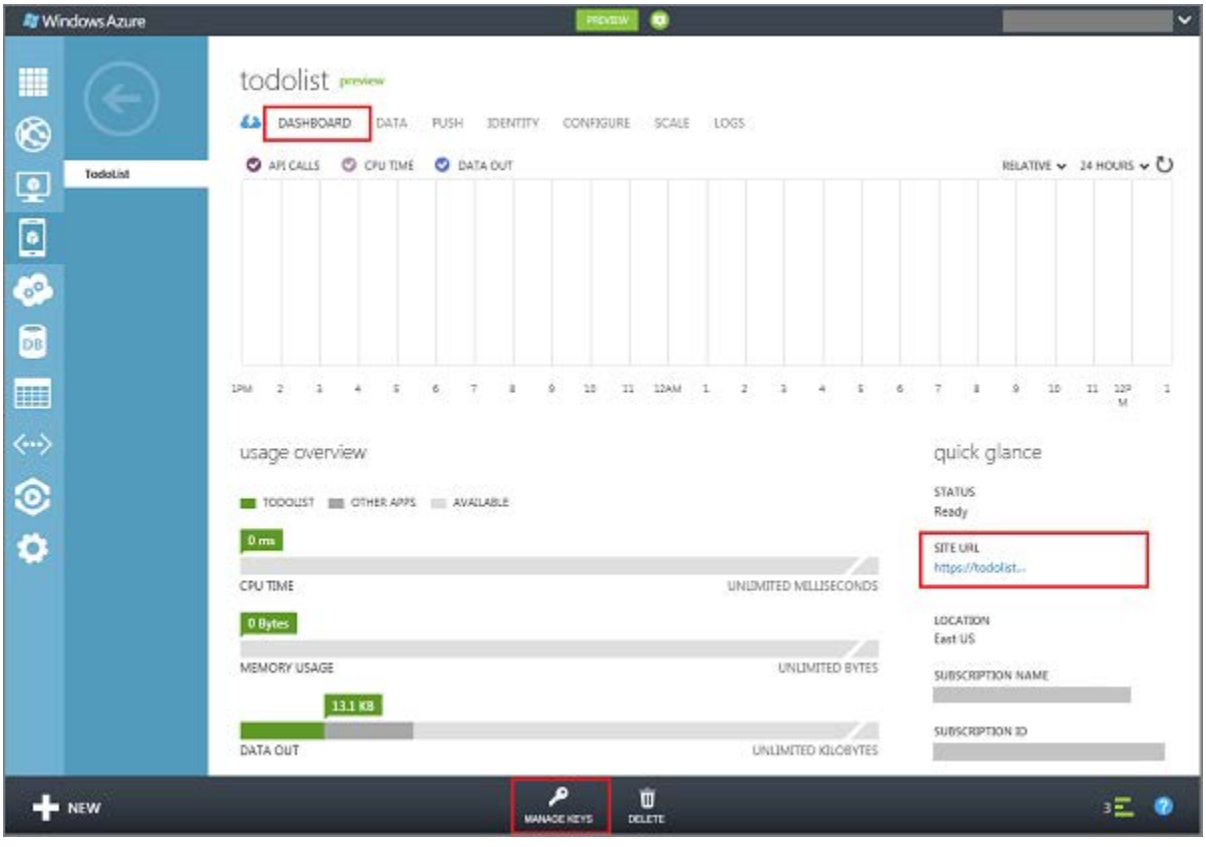
After this comment, add the following line of code inside the @interface declaration:

```
@property (nonatomic, strong)   MSTable *table;
```

This creates a property representation for your mobile services table.

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

6. 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.

7. Back in Xcode, open TodoService.m and locate the following commented line of code:

```
// Initialize the Mobile Service client with your URL and key.
```

After this comment, add the following line of code:

```
self.client = [MSClient clientWithApplicationURLString:@"APPURL"
withApplicationKey:@"APPKEY"];
```

This creates an instance of the Mobile Services client.

8. Replace the values of **APPURL** and **APPKEY** in this code with the URL and application key from the mobile service that you acquired in step 6.

9. Locate the following commented line of code:

```
// Create an MSTable instance to allow us to work with the TodoItem table.
```

After this comment, add the following line of code:

```
self.table = [self.client getTable:@"TodoItem"];
```

This creates the TodoItem table instance.

10. Locate the following commented line of code:

```
// Create a predicate that finds items where complete is false comment in
the refreshDataOnSuccess method.
```

After this comment, add the following line of code:

```
NSPredicate * predicate = [NSPredicate predicateWithFormat:@"complete ==
NO"];
```

This creates a query to return all tasks that have not yet been completed.

11. Locate the following commented line of code:

```
// Query the TodoItem table and update the items property with the results
from the service.
```

Replace that comment and the subsequent **completion** block invocation with the following code:

```
// Query the TodoItem table and update the items property with the results
from the service
[self.table readWhere:predicate completion:^(NSArray *results, NSInteger
totalCount, NSError *error)
{
   self.items = [results mutableCopy];
   completion();
}];
```

12. Locate the **addItem** method, and replace the body of the method with the following code:

```
// Insert the item into the TodoItem table and add to the items array
on completion
```

```
[self.table insert:item completion:^(NSDictionary *result, NSError
*error) {
    NSUInteger index = [items count];
    [(NSMutableArray *)items insertObject:item atIndex:index];


// Let the caller know that we finished
completion(index);


}];
```

This code sends an insert request to the mobile service.

13. Locate the **completeItem** method, and replace the body of the method with the following code:

```
// Update the item in the TodoItem table and remove from the items
array on completion
[self.table update:mutable completion:^(NSDictionary *item, NSError
*error) {


// TODO
// Get a fresh index in case the list has changed
NSUInteger index = [items indexOfObjectIdenticalTo:mutable];


[mutableItems removeObjectAtIndex:index];


// Let the caller know that we have finished
completion(index);


}];
```

This code removes TodoItems after they are marked as completed.

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 Xcode, select an emulator to deploy to (either iPhone or iPad), press the **Run** button (or the Command+R key) to rebuild the project and start the app.

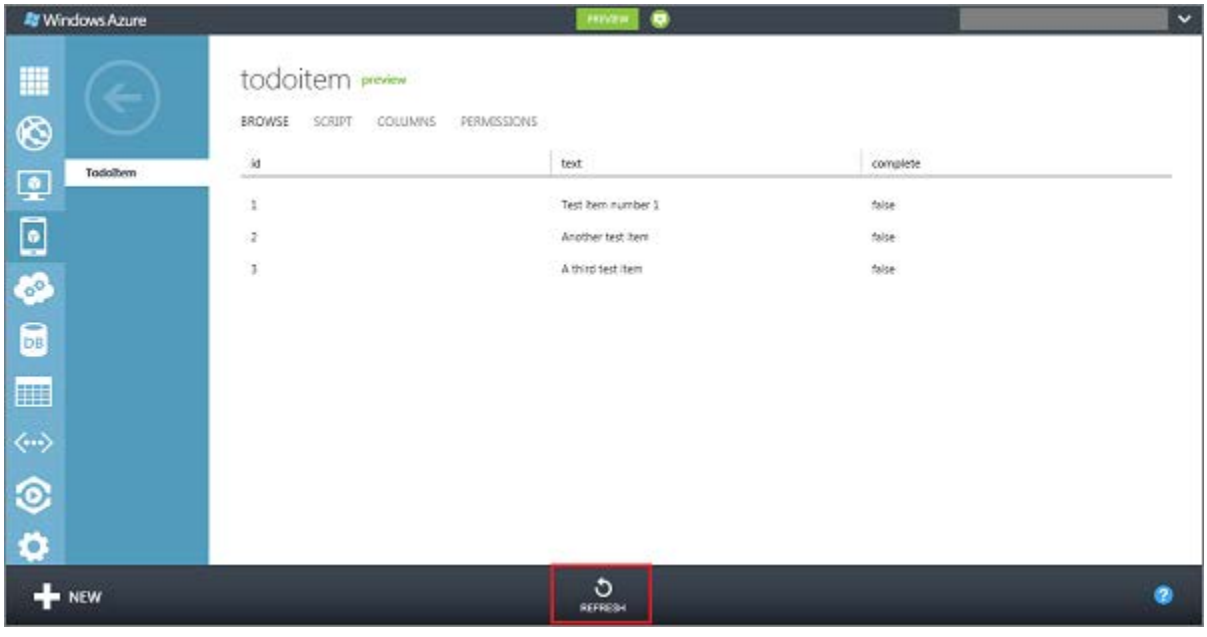
   This executes your Windows Azure Mobile Services client, built with the iOS SDK, that queries items from your mobile service.

2. As before, type text in the textbox, and then click the **+** button..

   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.

This concludes the **Get started with data** tutorial for iOS.

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 iOS app to take advantage of these new behaviors.

This tutorial walks you through these basic steps:

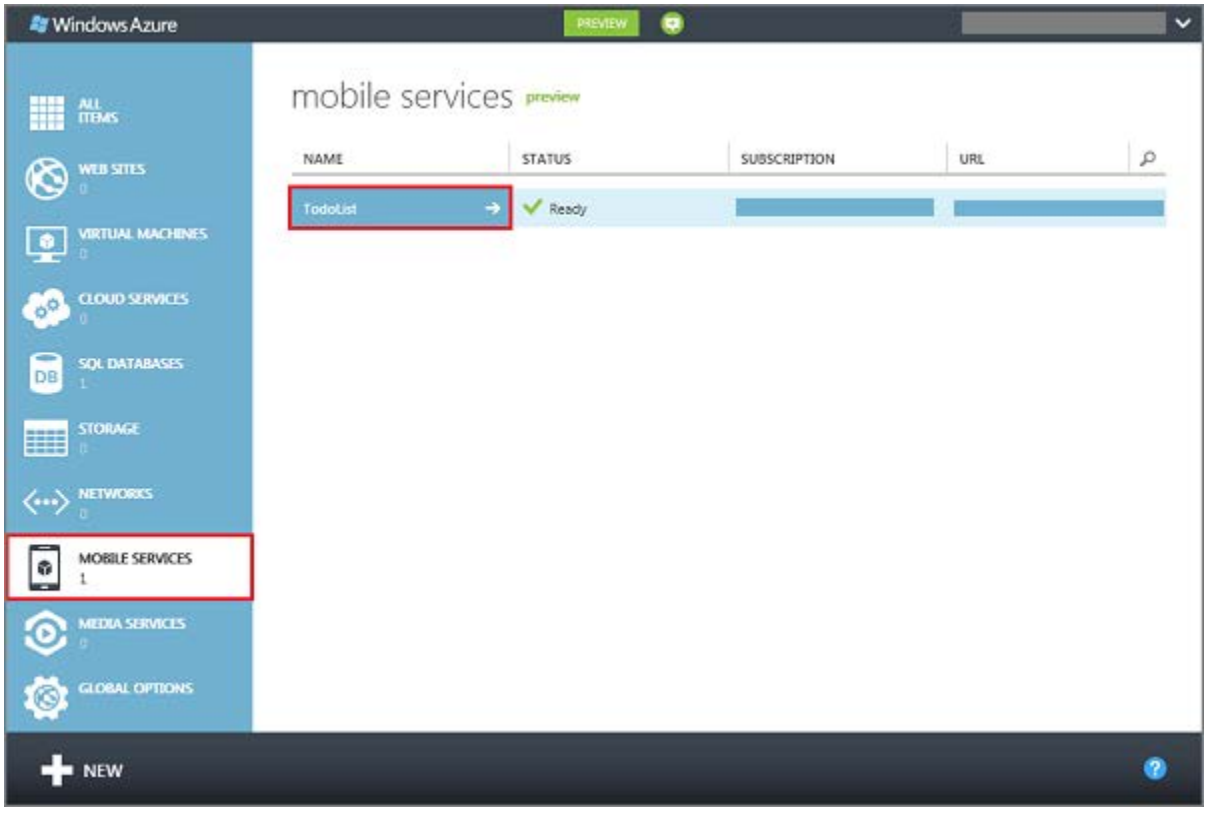
1. Add string length validation
2. Update the client to support validation

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

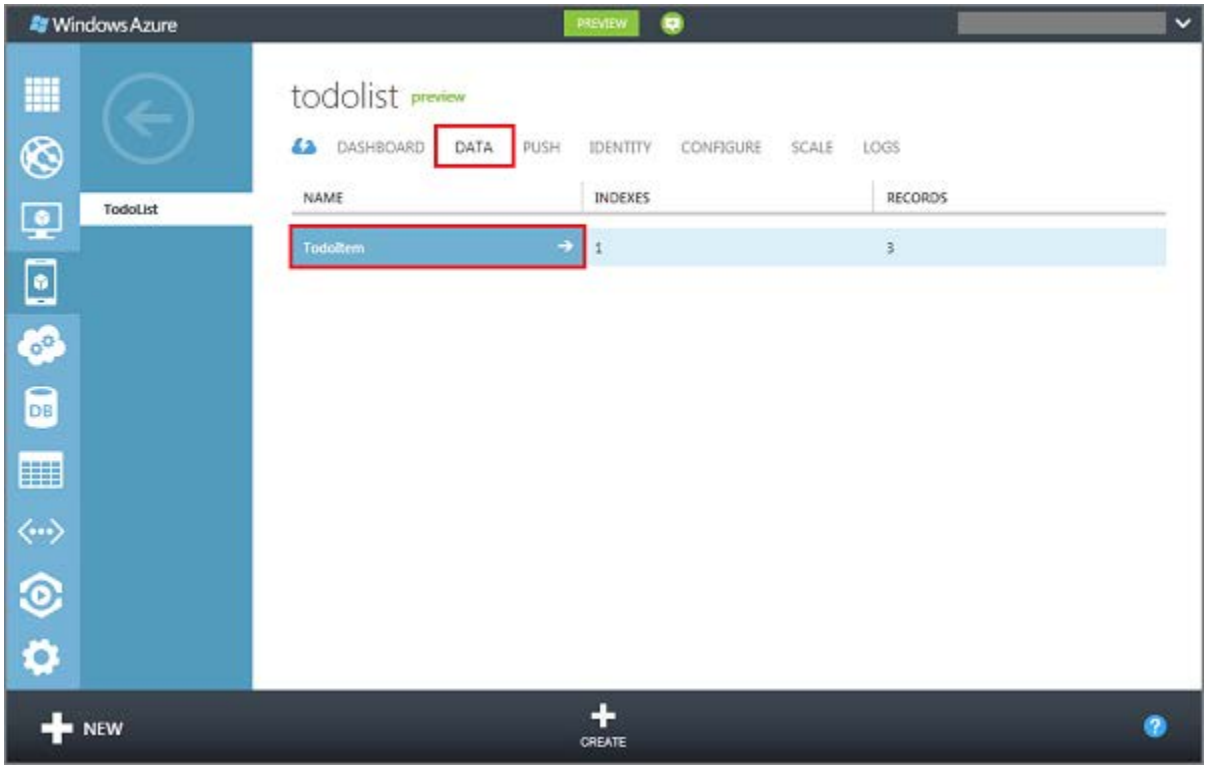
Add validation

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.

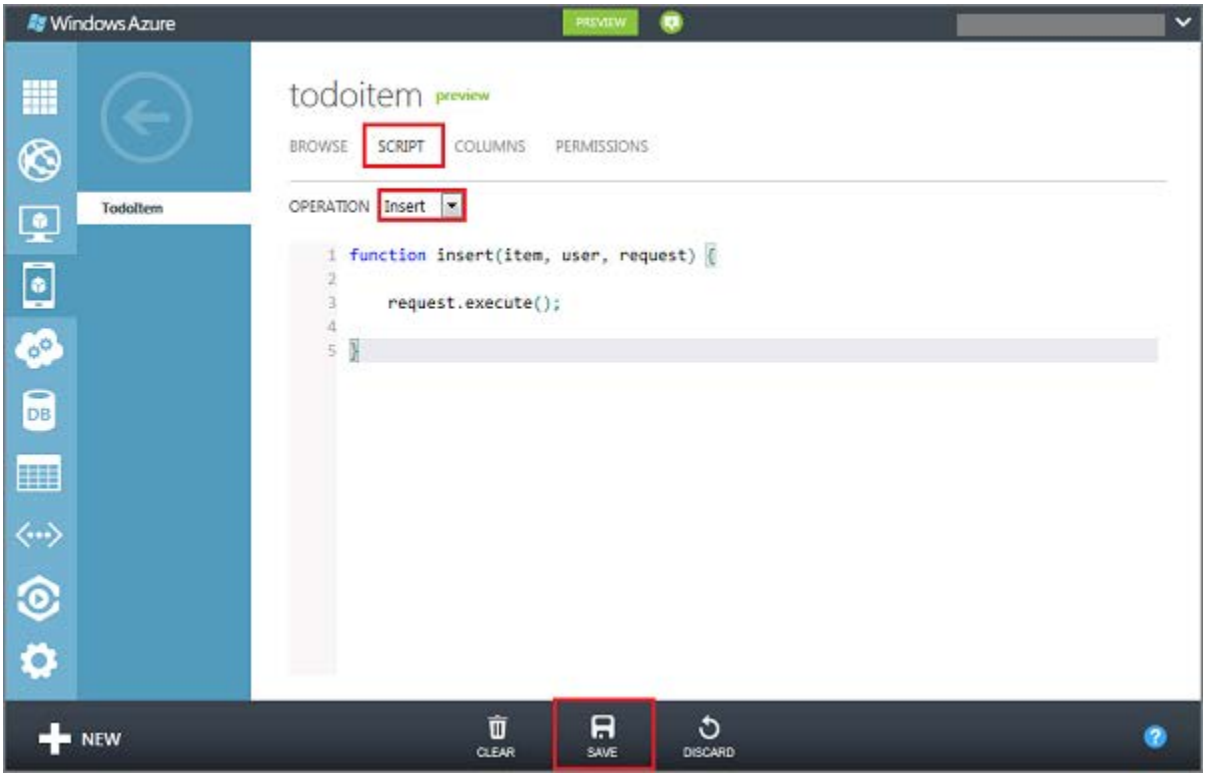
1. Log into the Windows Azure Management Portal, click **Mobile Services**, and then click your app.



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
10 characters or less.');
    } else {
        request.execute();
    }
}
```

This script checks the length of the **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 Xcode, open the project that you modified when you completed the tutorial Get started with data.

2. Press the **Run** button (Command + R) to build the project and start the app, then type text longer than 10 characters in the textbox and click the plus (+) icon.

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

3. In the TodoService.m file, locate the following line of code in the **addItem** method:

```
[self logErrorIfNotNil:error];
```

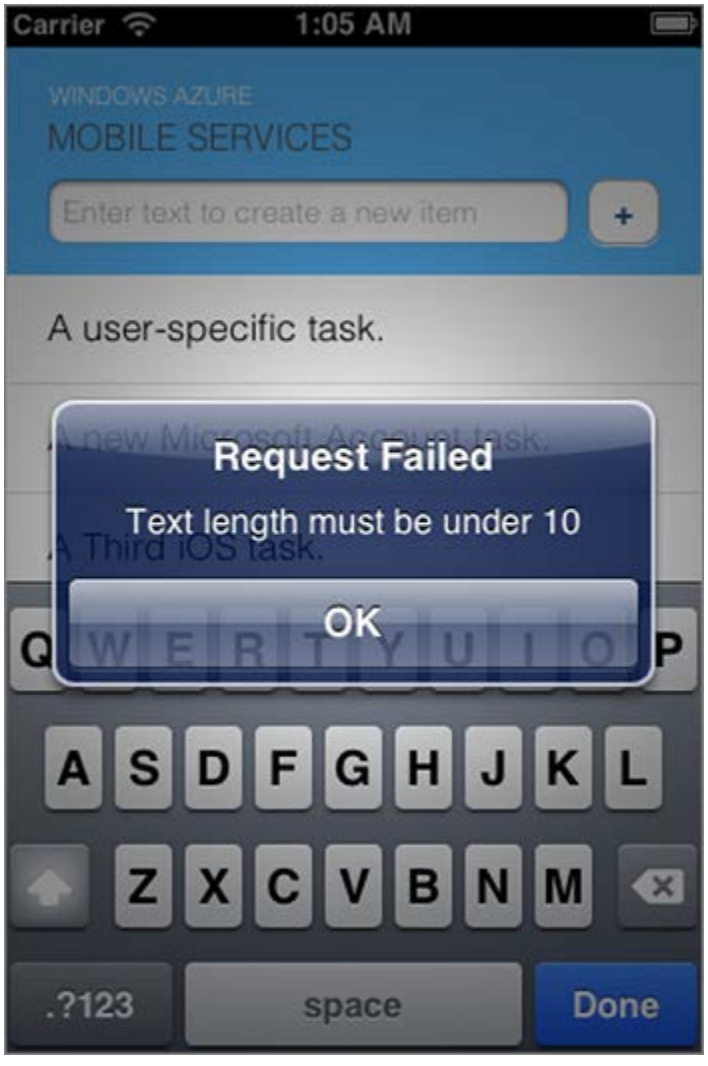
   After this line of code, replace the remainder of the completion block with the following code:

```
BOOL goodRequest = !((error) && (error.code == MSErrorMessageErrorCode));

// detect text validation error from service.
if (goodRequest) // The service responded appropriately
{
    NSUInteger index = [items count];
    [(NSMutableArray *)items insertObject:result atIndex:index];

    // Let the caller know that we finished
    completion(index);
}
else{

    // if there's an error that came from the service
    // log it, and popup up the returned string.
    if (error && error.code == MSErrorMessageErrorCode) {
        NSLog(@"ERROR %@", error);
        UIAlertView *av =
        [[UIAlertView alloc]
         initWithTitle:@"Request Failed"
         message:error.localizedDescription
         delegate:nil
         cancelButtonTitle:@"OK"
         otherButtonTitles:nil
         ];
        [av show];
    }
```

```
}
```

This logs the error to the output window and displays it to the user.

4. Rebuild and start the app.



Notice that error is handled and the error messaged is displayed to the user.

Refine Mobile Services queries with paging

This topic shows you how to use paging to manage the amount of data returned to your iOS app from Windows Azure Mobile Services. In this tutorial, you will use the **fetchLimit** and **fetchOffset** query properties 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 Xcode, open the project that you modified when you completed the tutorial Get started with data.
2. Press the **Run** button (Command + R) to build the project and start the app, then enter text into the textbox and click the plus (**+**) icon.
3. Repeat the previous step at least three times, so that you have more than three items stored in the TodoItem table.
4. Open the TodoService.m file, and locate the following method:

```
- (void) refreshDataOnSuccess:(CompletionBlock)completion
```

Replace the body of the entire method with the following code.

```
// Create a predicate that finds active items in which complete is false
NSPredicate * predicate = [NSPredicate predicateWithFormat:@"complete ==
NO"];


// Retrieve the MSTable's MSQuery instance with the predicate you just
created.
MSQuery * query = [self.table queryWhere:predicate];


query.includeTotalCount = TRUE; // Request the total item count
```

```
// Start with the first item, and retrieve only three items
query.fetchOffset = 0;
query.fetchLimit = 3;


// Invoke the MSQuery instance directly, rather than using the MSTable
helper methods.
[query readWithCompletion:^(NSArray *results, NSInteger totalCount,
NSError *error) {


[self logErrorIfNotNil:error];
   if (!error)
{
    // Log total count.
    NSLog(@"Total item count: %@",[NSString stringWithFormat:@"%zd",
(ssize_t) totalCount]);
}


items = [results mutableCopy];


// Let the caller know that we finished
completion();


}];
```

This query returns the top three items that are not marked as completed.

5. Rebuild and start the app.

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

6. Update the **refreshDataOnSuccess** method once more by locating the following line of code:

```
query.fetchOffset = 0;
```

This time, set the **query.fetchOffset** value to 3.

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 setting hard-coded paging values for the **fetchOffset** and **fetchLimit** properties. 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 set **query.includeTotalCount = YES** to get the total count of all items available on the server, along with the paged data.

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.

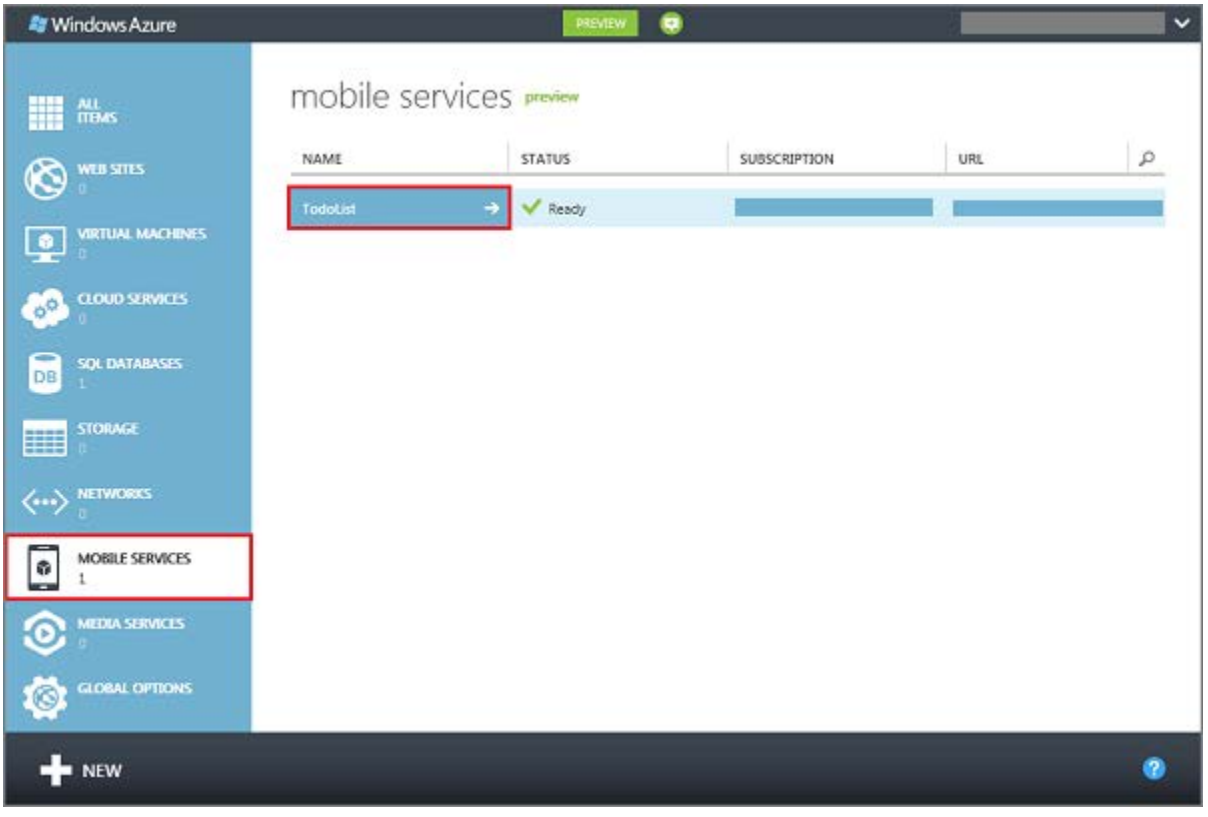
Completing this tutorial requires XCode 4.5 and iOS 5.0 or later versions.

Register your app for authentication and configure Mobile Services

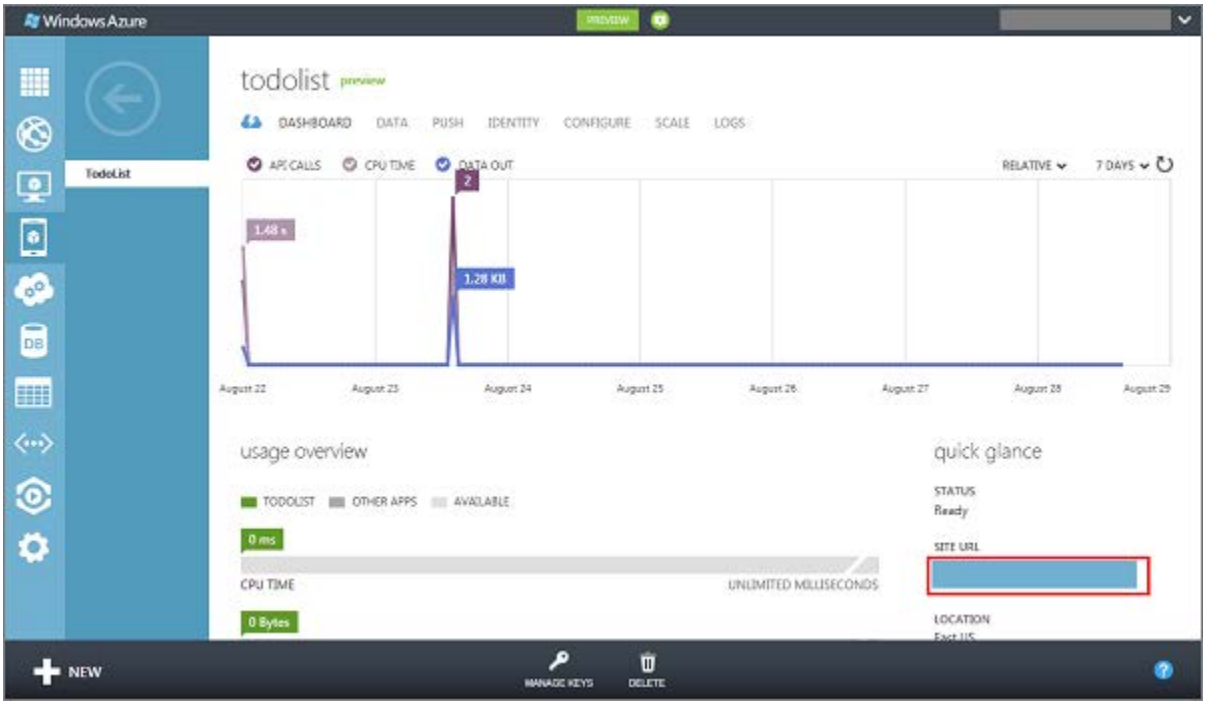
To be able to authenticate users, you must register your app with an identity provider. You must then register the provider-generated client secret with Mobile Services.

> **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.



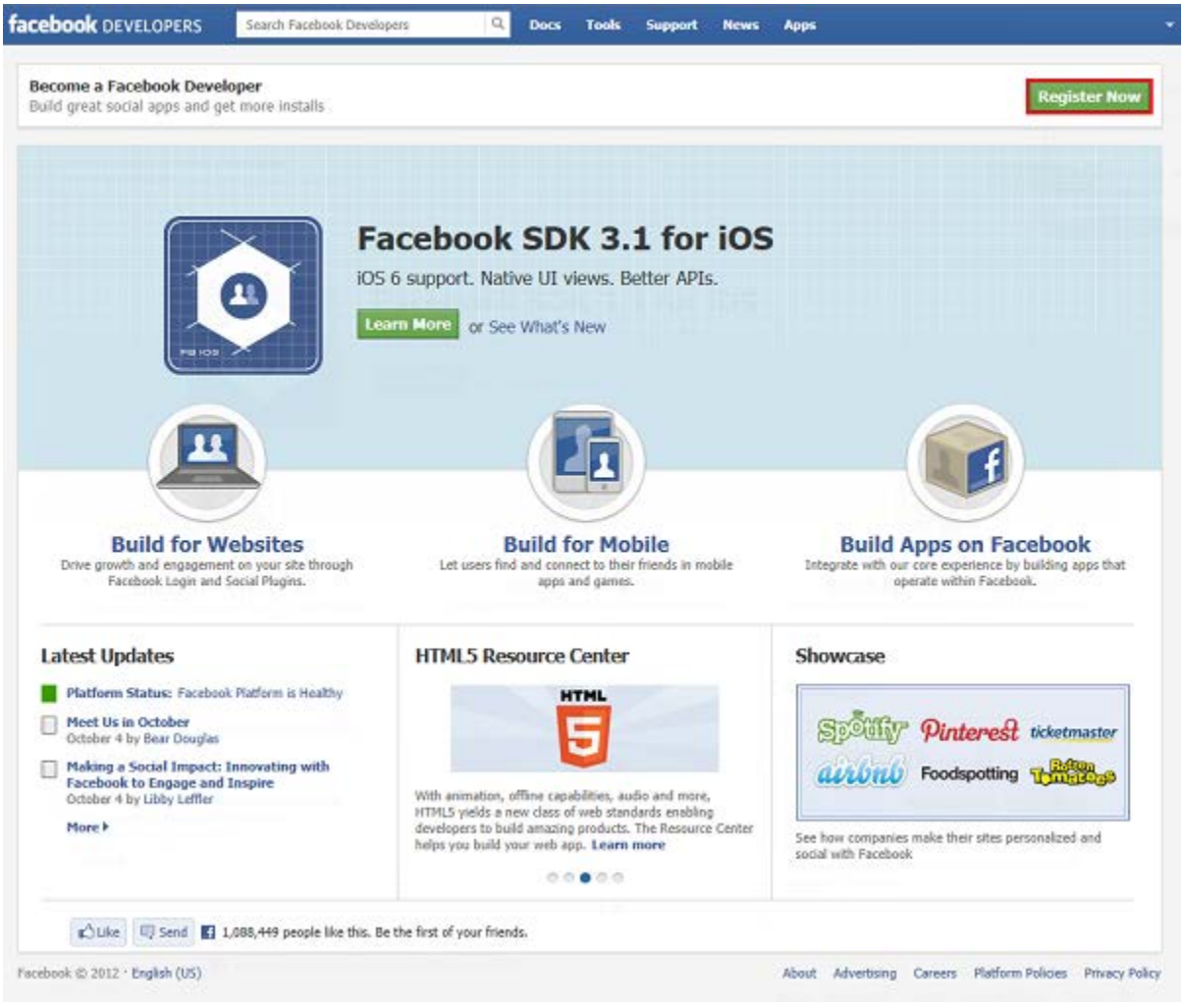
2. Click the **Dashboard** tab and make a note of the **Site URL** value.



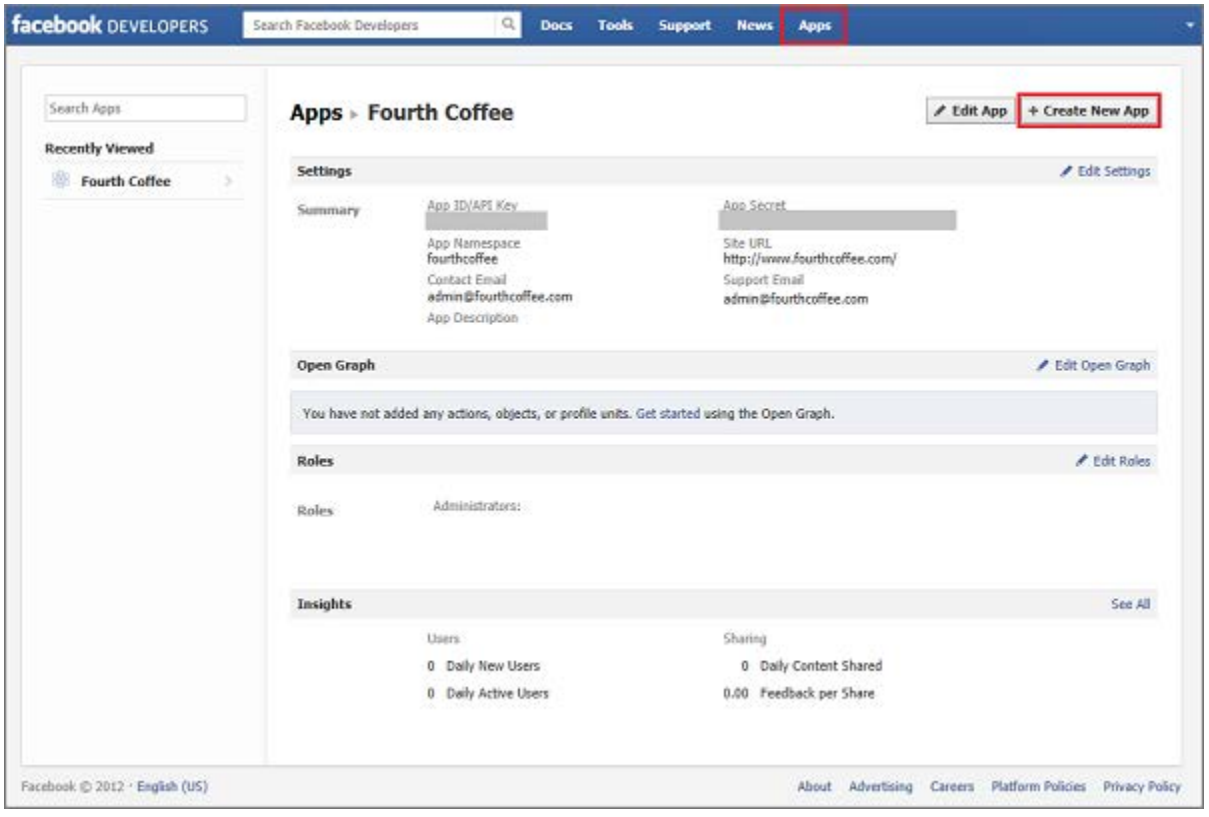
You may need to provide this value to the identity provider when you register your app.

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.

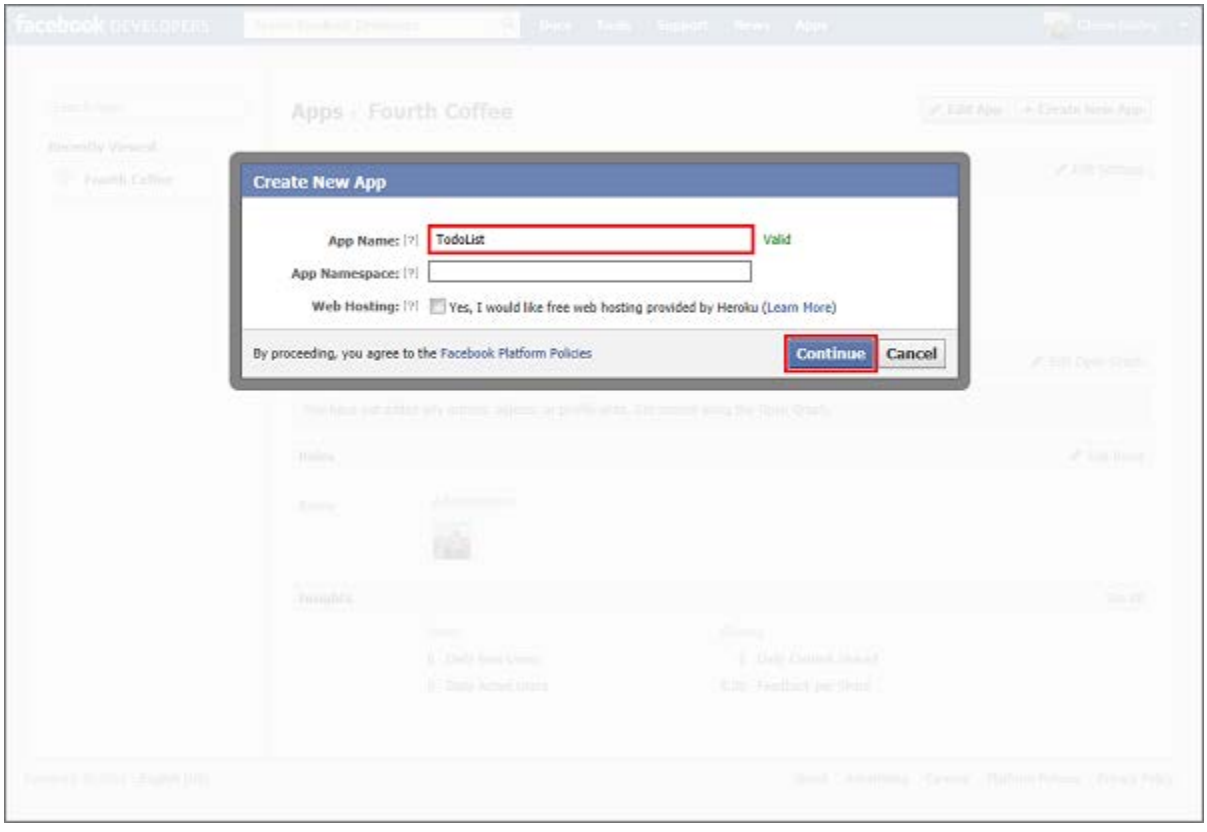
3. Navigate to the Facebook Developers web site and sign-in with your Facebook account credentials.
4. (Optional) If you have not already registered, click **Register Now** button, accept the policy, provide any and then click **Done**.



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

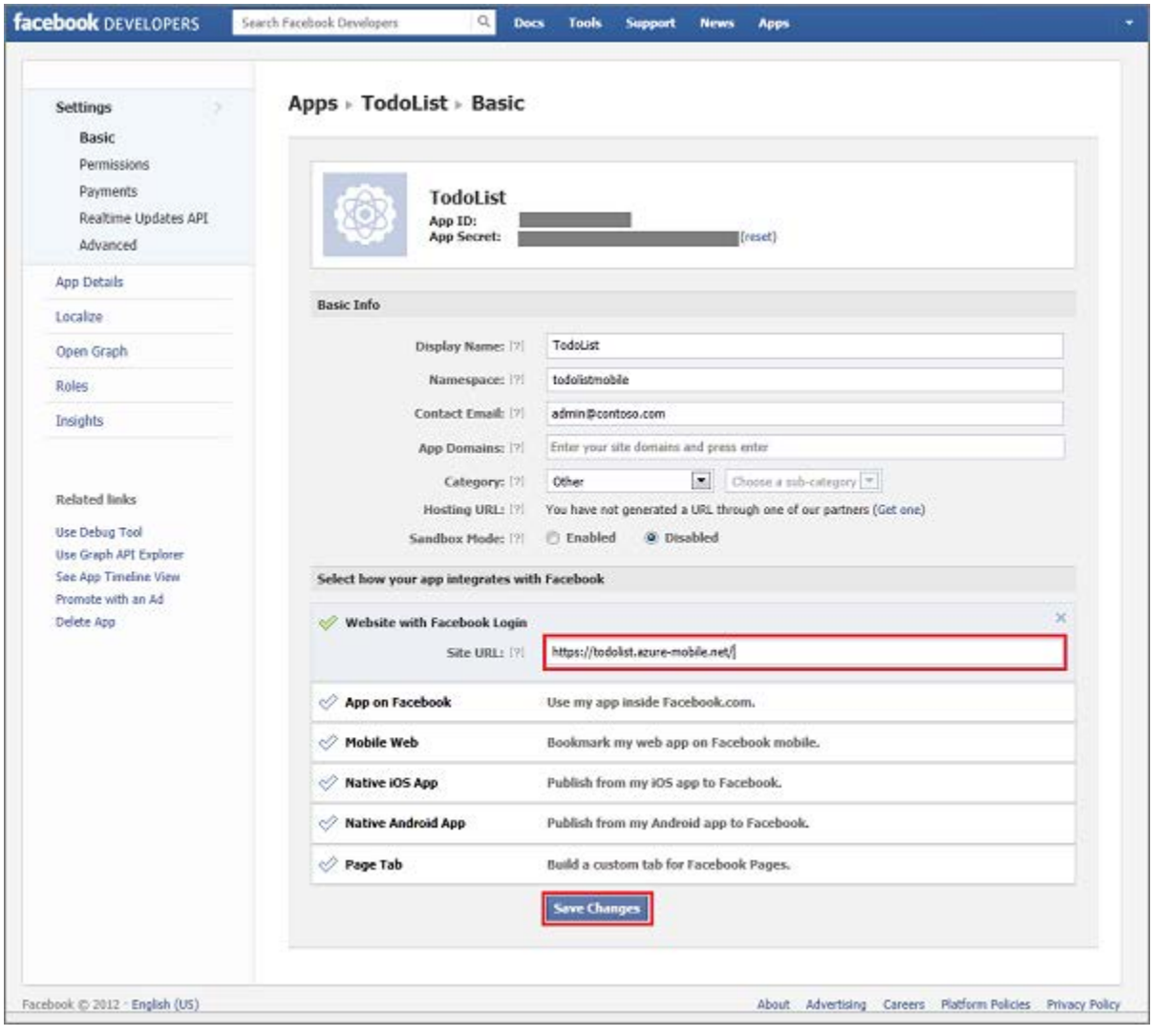


6. Choose a unique name for your app, select **OK**.

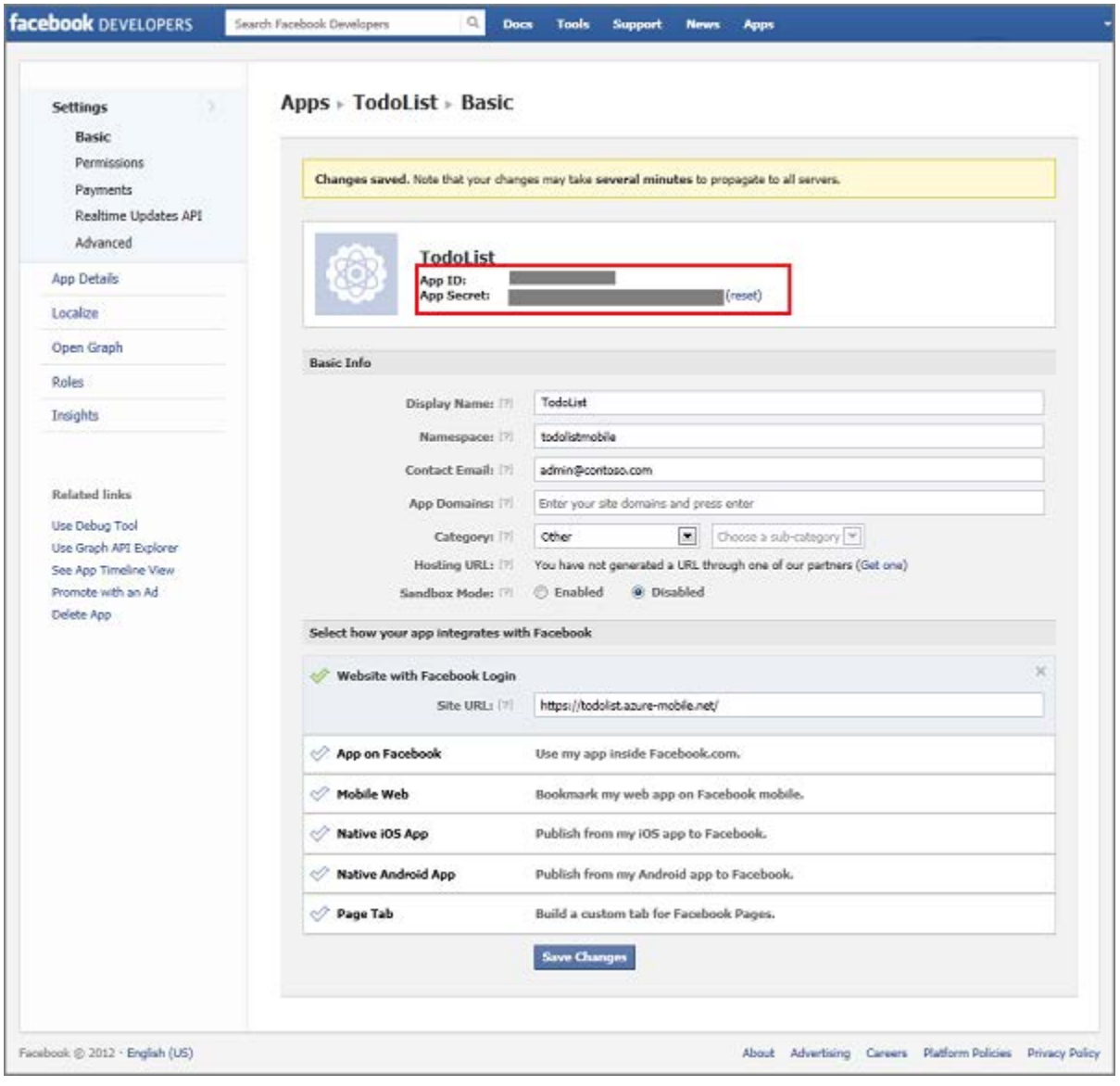


This registers the app with Facebook

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**.



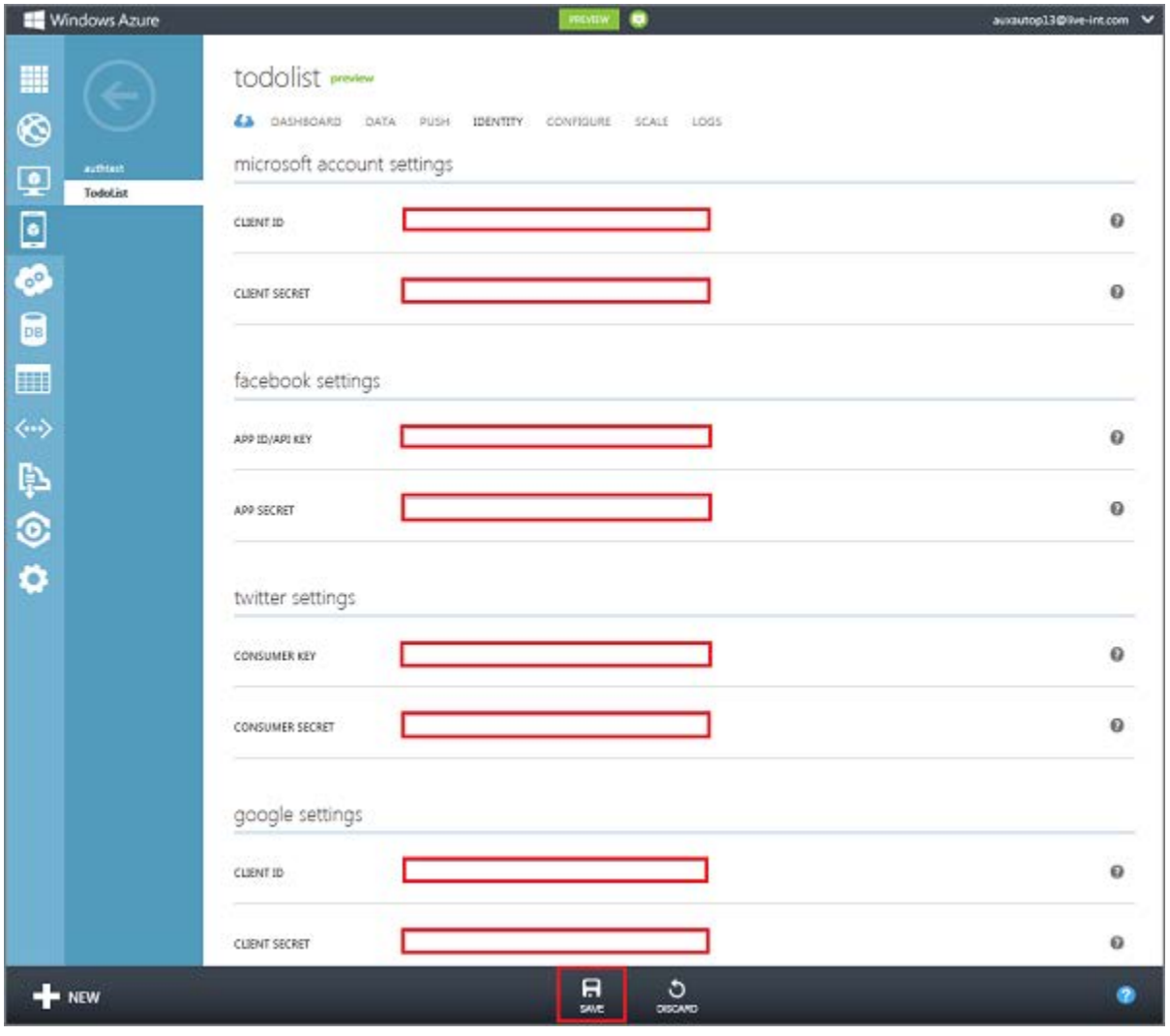
8. Make a note of the values of **App ID** and **App Secret**.



> **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.

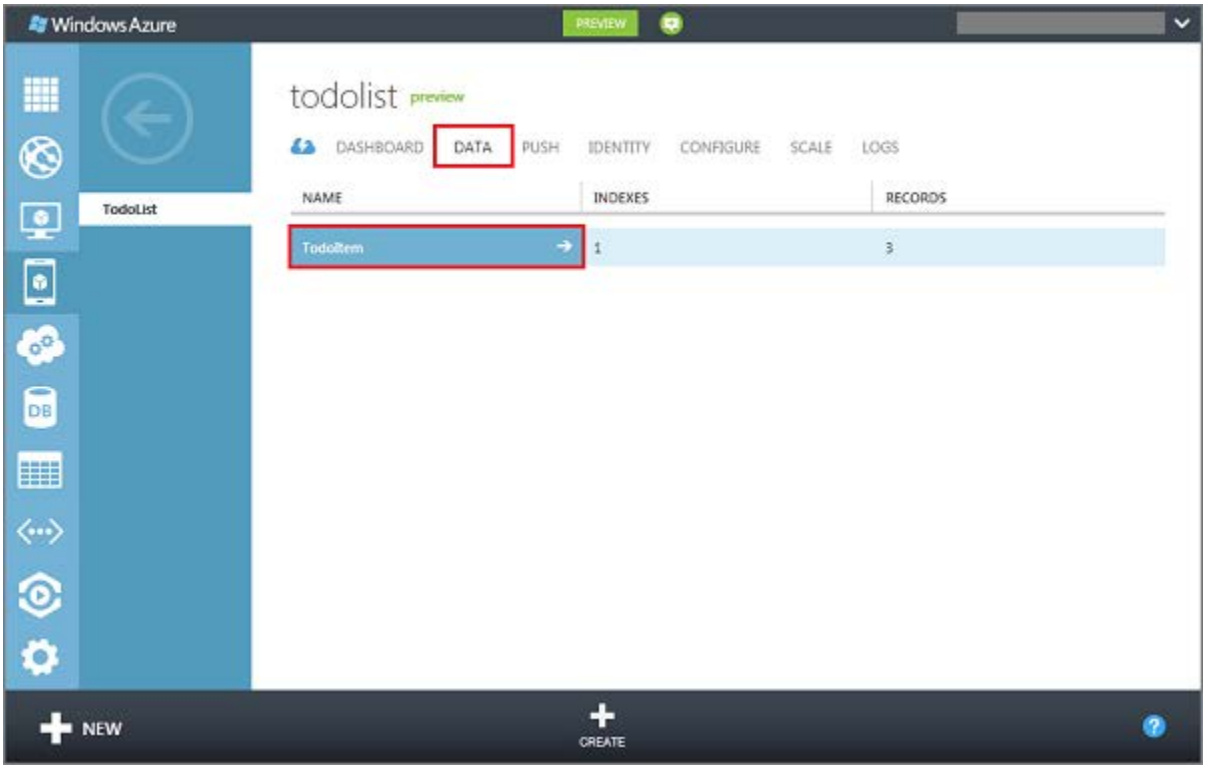
9. Back in the Management Portal, click the **Identity** tab, enter the app identifier and shared secret values obtained from your identity provider, and click **Save**.



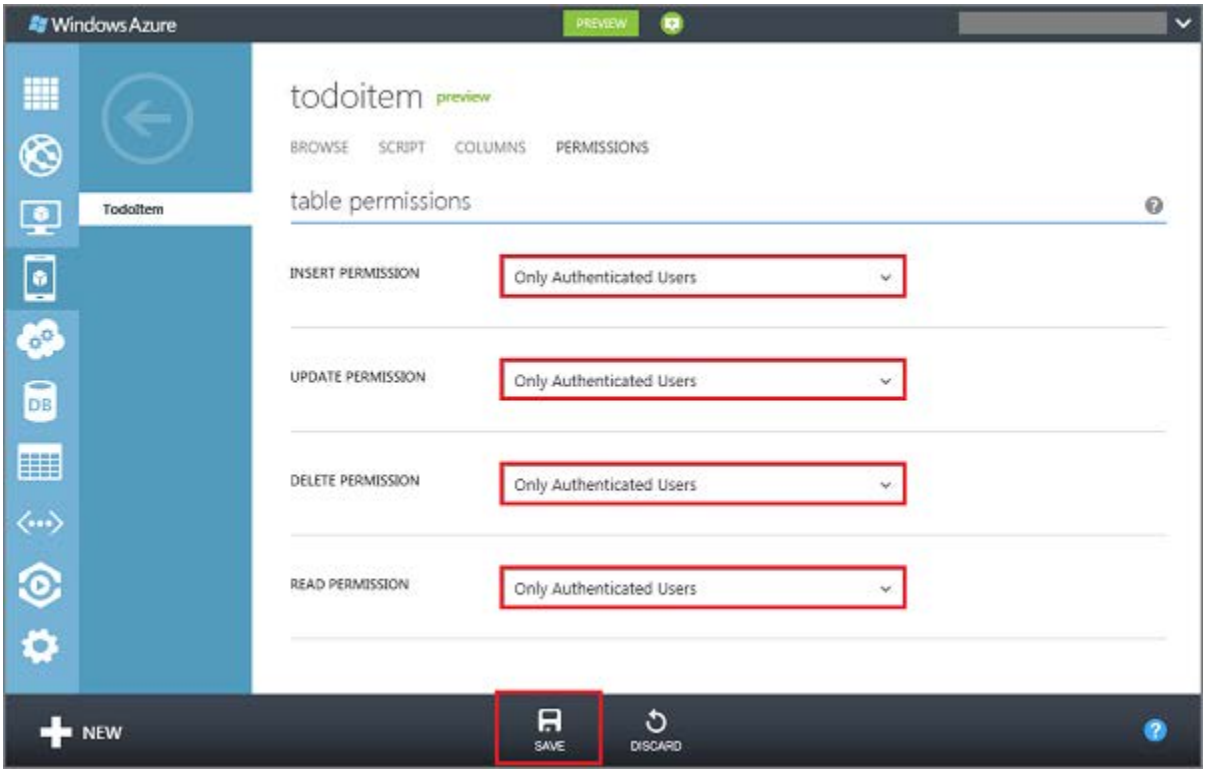
Both your mobile service and your app are now configured to work with your chosen authentication provider.

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 Xcode, open the project that you created when you completed the tutorial Get started with data in Mobile Services.

4. Press the **Run** button to build the project and start the app in the iPhone emulator; 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 TodoListController.m and in the **viewDidLoad** method, remove the following code that reloads the data into the table:

```
[todoService refreshDataOnSuccess:^{
    [self.tableView reloadData];
}];
```

2. Just after the **viewDidLoad** method, add the following code:

```
- (void)viewDidAppear:(BOOL)animated
{
    // If user is already logged in, no need to ask for auth
    if (todoService.client.currentUser == nil)
    {
        // We want the login view to be presented after the this run
loop has completed
        // Here we use a delay to ensure this.
        [self performSelector:@selector(login) withObject:self
afterDelay:0.1];
    }
}


- (void) login
{
    UINavigationController *controller =


[self.todoService.client
    loginViewControllerWithProvider:@"facebook"
    completion:^(MSUser *user, NSError *error) {


    if (error) {
            NSLog(@"Authentication Error: %@", error);
            // Note that error.code == -1503 indicates
            // that the user cancelled the dialog
    } else {
        // No error, so load the data
        [self.todoService refreshDataOnSuccess:^{
            [self.tableView reloadData];
        }];
    }


    [self dismissViewControllerAnimated:YES completion:nil];
}];
```

```
[self presentViewController:controller animated:YES completion:nil];


}
```

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:** If you are using an identity provider other than Facebook, change the value passed to **loginViewControllerWithProvider** above to one of the following: microsoftaccount, facebook, twitter, or google.

3. Press the **Run** button to build the project, start the app in the iPhone emulator, then log-on 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

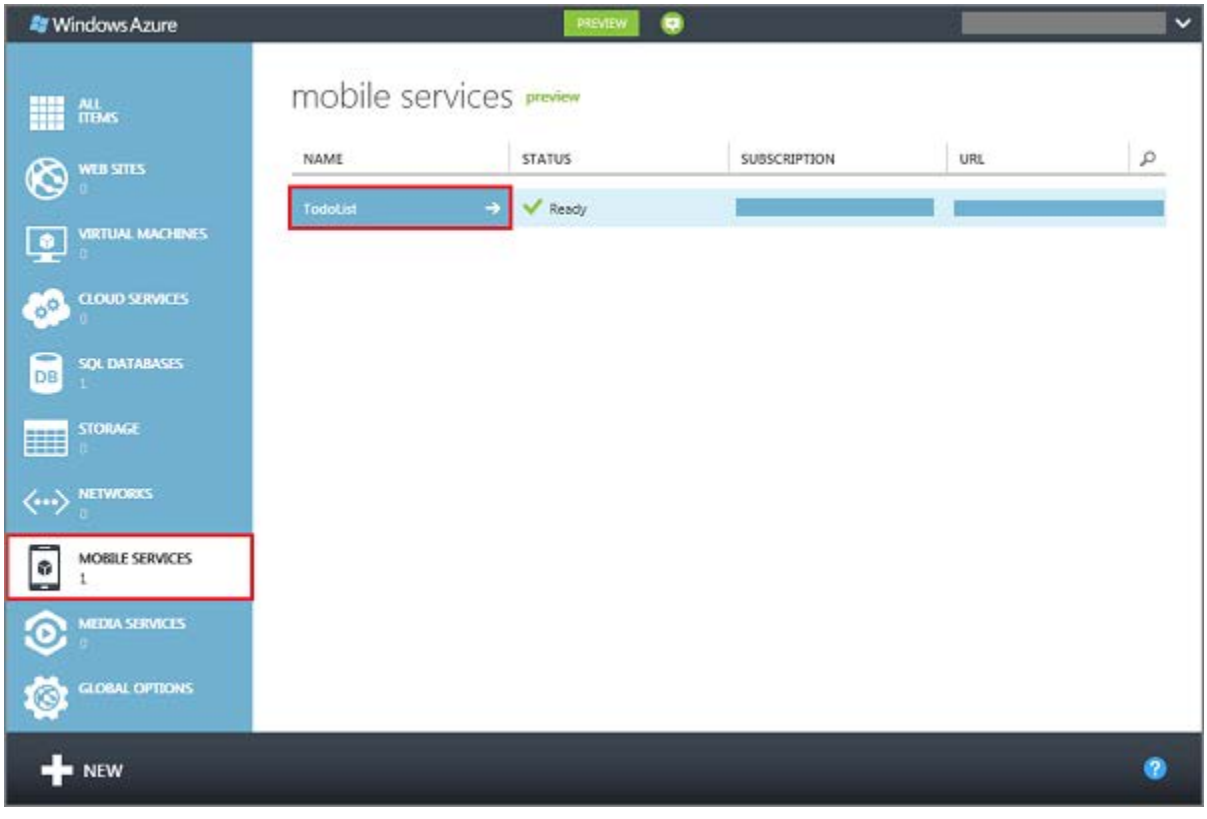
This section shows you how to use server scripts to authorize authenticated users for accessing data in Windows Azure Mobile Services from an iOS app. In this tutorial you register scripts with Mobile Services to filter queries based on the userId of an authenticated user, ensuring that each user can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

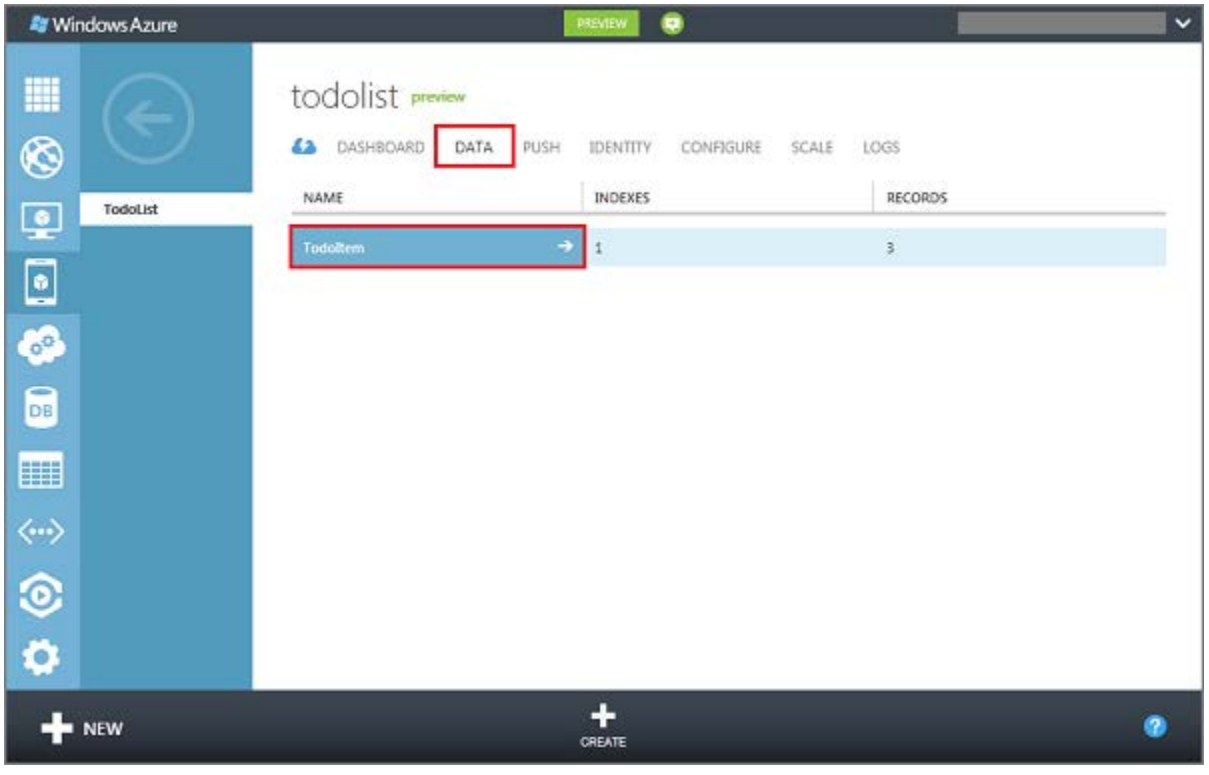
Register scripts

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

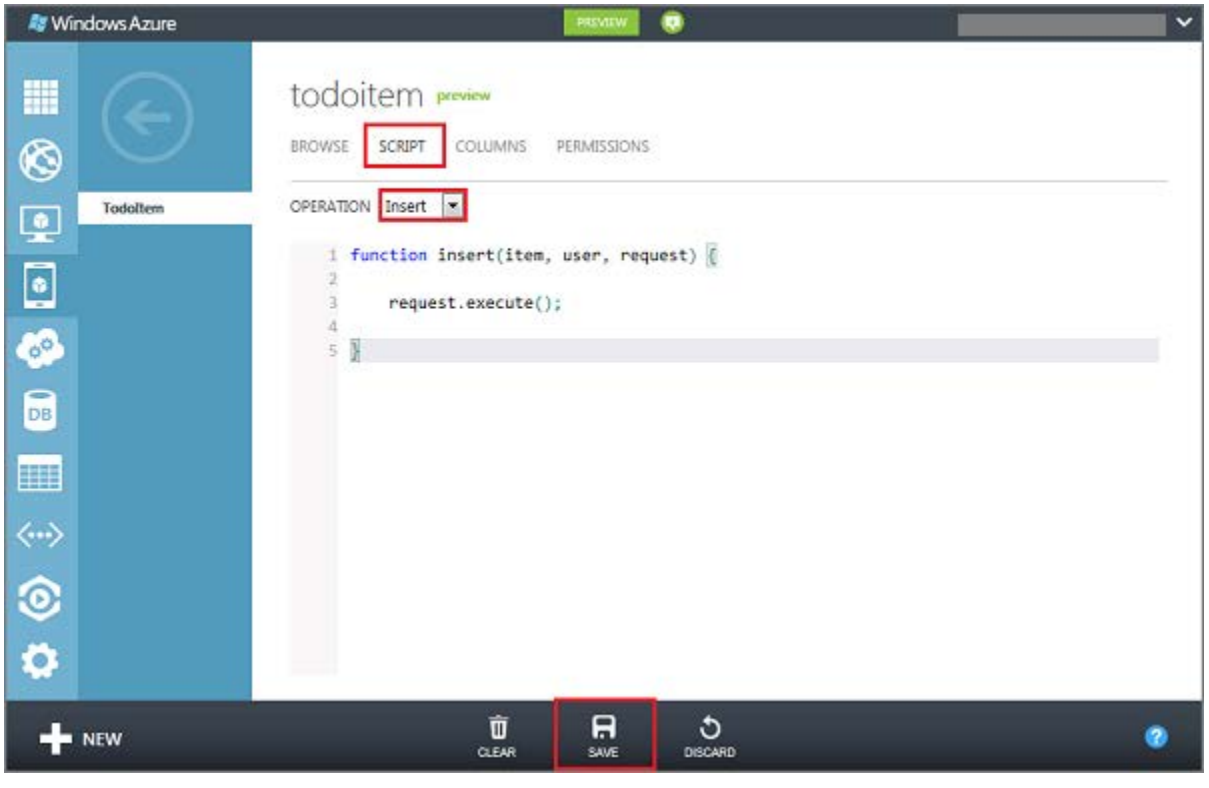
1. Log on to the Windows Azure Management Portal, click **Mobile Services**, and then click your app.



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.

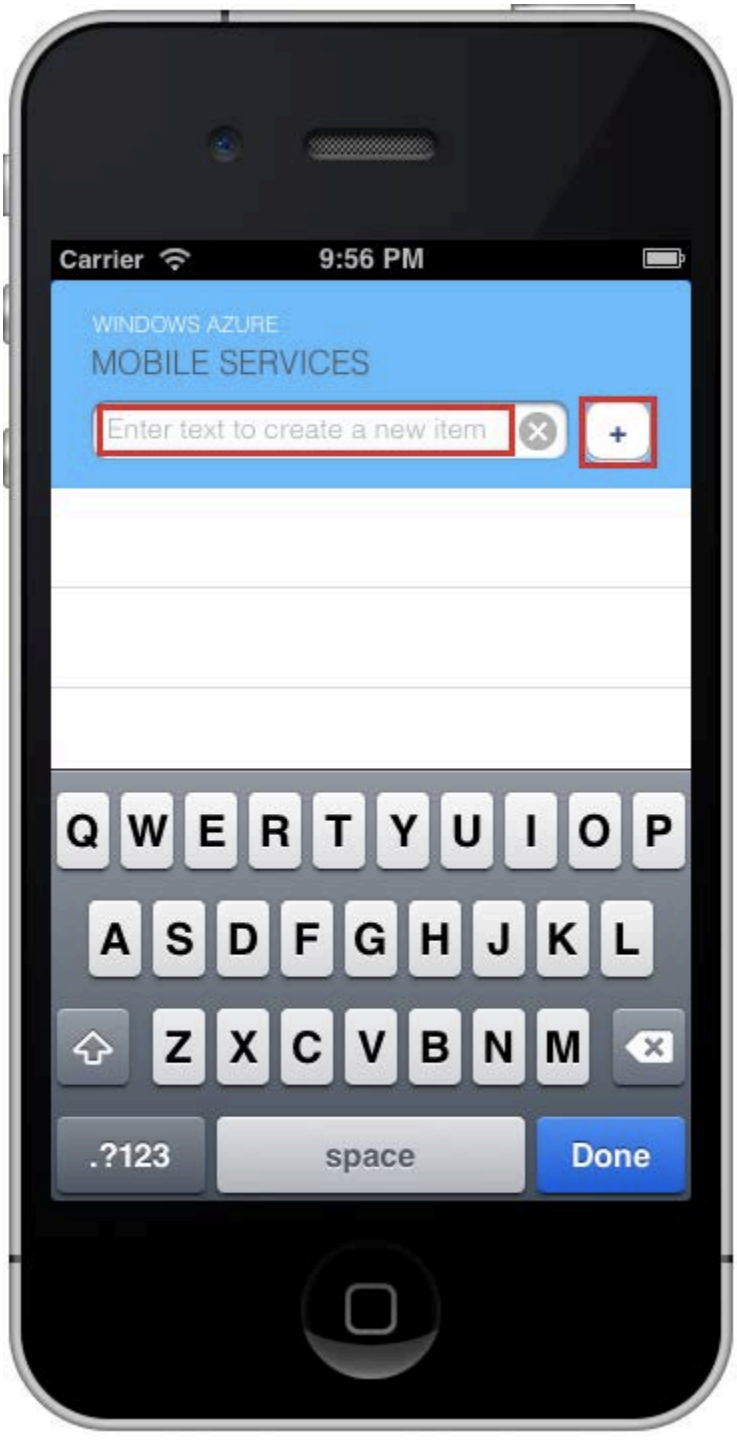
Test the app

1. In Xcode, open the project that you modified when you completed the tutorial Get started with authentication.

2. Press the **Run** button to build the project, start the app in the iPhone emulator, 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 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.

Get started with push notifications in Mobile Services

This section shows you how to use Windows Azure Mobile Services to send push notifications to an iOS app. In this tutorial you add push notifications using the Apple Push Notification service (APNS) 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 device token 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. Generate the certificate signing request
2. Register your app and enable push notifications
3. Create a provisioning profile for the app
4. Configure Mobile Services
5. Add push notifications to the app
6. Update scripts to send push notifications
7. Insert data to receive notifications

This tutorial requires the following:

- Mobile Services iOS SDK
- XCode 4.5
- An iOS 5.0 (or later version) capable device
- iOS Developer Program membership

> **Note:** Because of push notification configuration requirements, you must deploy and test push notifications on an iOS capable device (iPhone or iPad) instead of in the emulator.

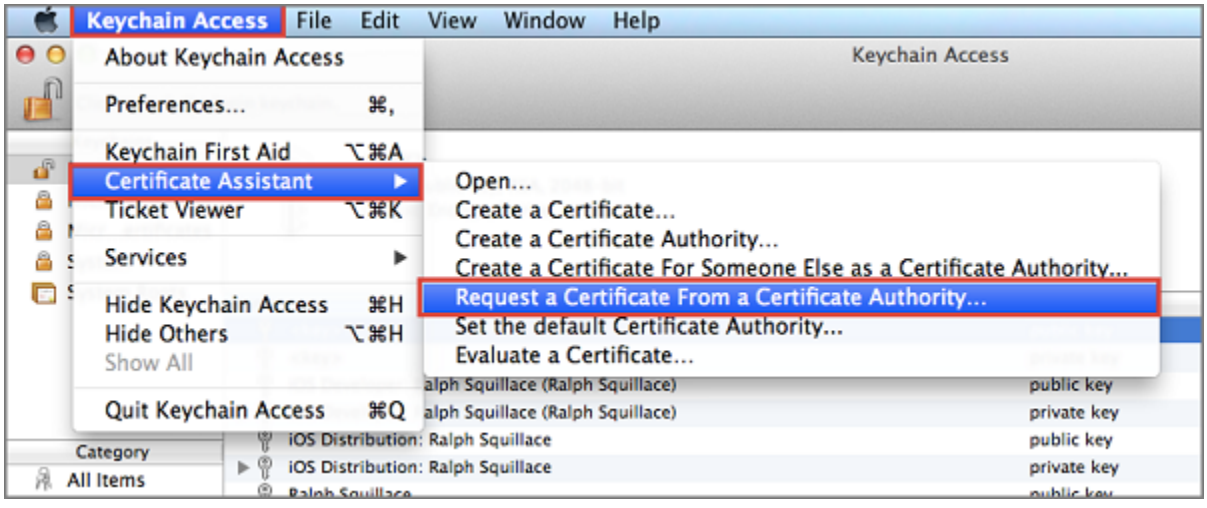
This tutorial is based on the Mobile Services quickstart. Before you start this tutorial, you must first complete Get started with data with Mobile Services.

The Apple Push Notification Service (APNS) uses certificates to authenticate your mobile service. Follow these instructions to create the necessary certificates and upload it to your Mobile Service. For the official APNS feature documentation, see Apple Push Notification Service.

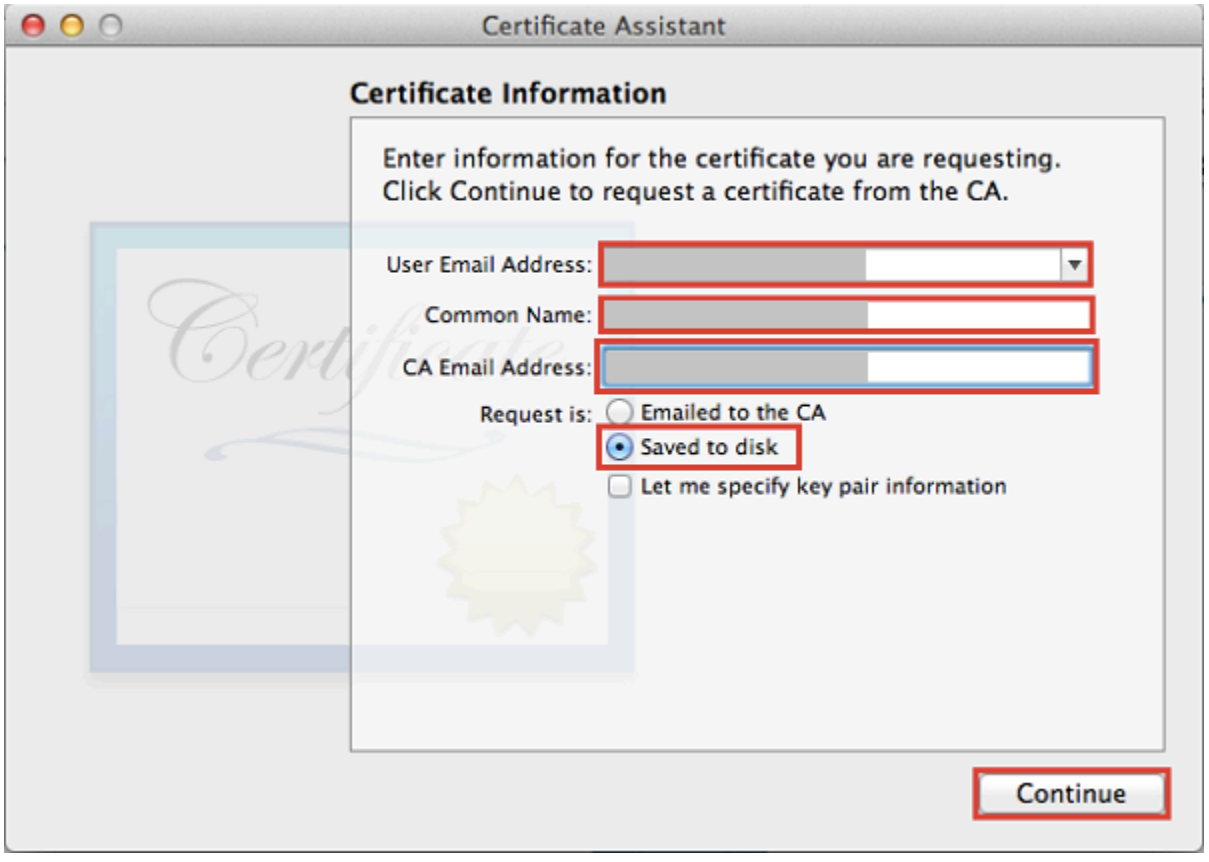
Generate the Certificate Signing Request file

First you must generate the Certificate Signing Request (CSR) file, which is used by Apple to generate a signed certificate.

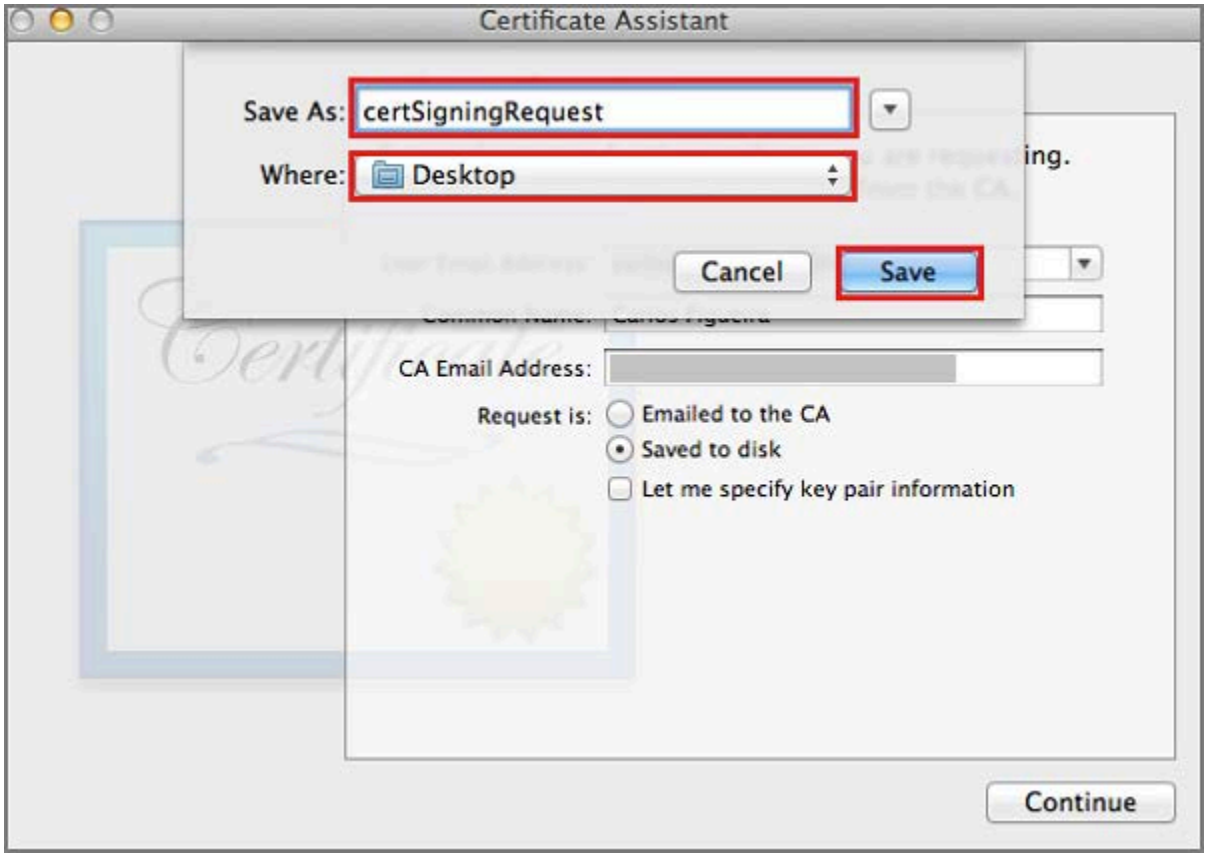
1. From the Utilities folder, run the Keychain Access tool.
2. Click **Keychain Access**, expand **Certificate Assistant**, then click **Request a Certificate from a Certificate Authority...**.



3. Select your **User Email Address**, type **Common Name** and **CA Email Address** values, make sure that **Saved to disk** is selected, and then click **Continue**.



4. Type a name for the Certificate Signing Request (CSR) file in **Save As**, select the location in **Where**, then click **Save**.



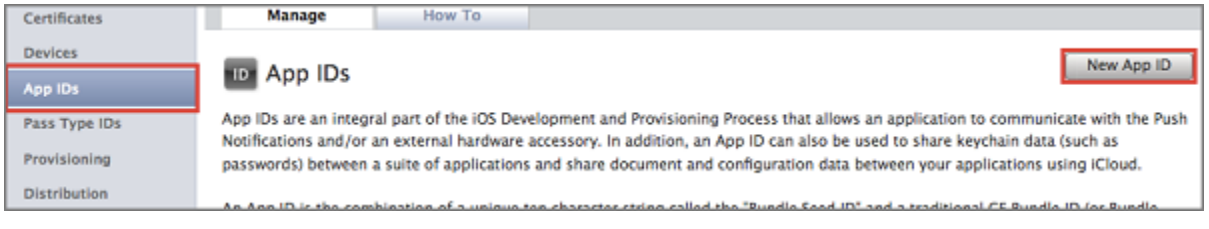
This saves the CSR file in the selected location; the default location is in the Desktop. Remember the location chosen for this file.

Next, you will register your app with Apple, enable push notifications, and upload this exported CSR to create a push certificate.

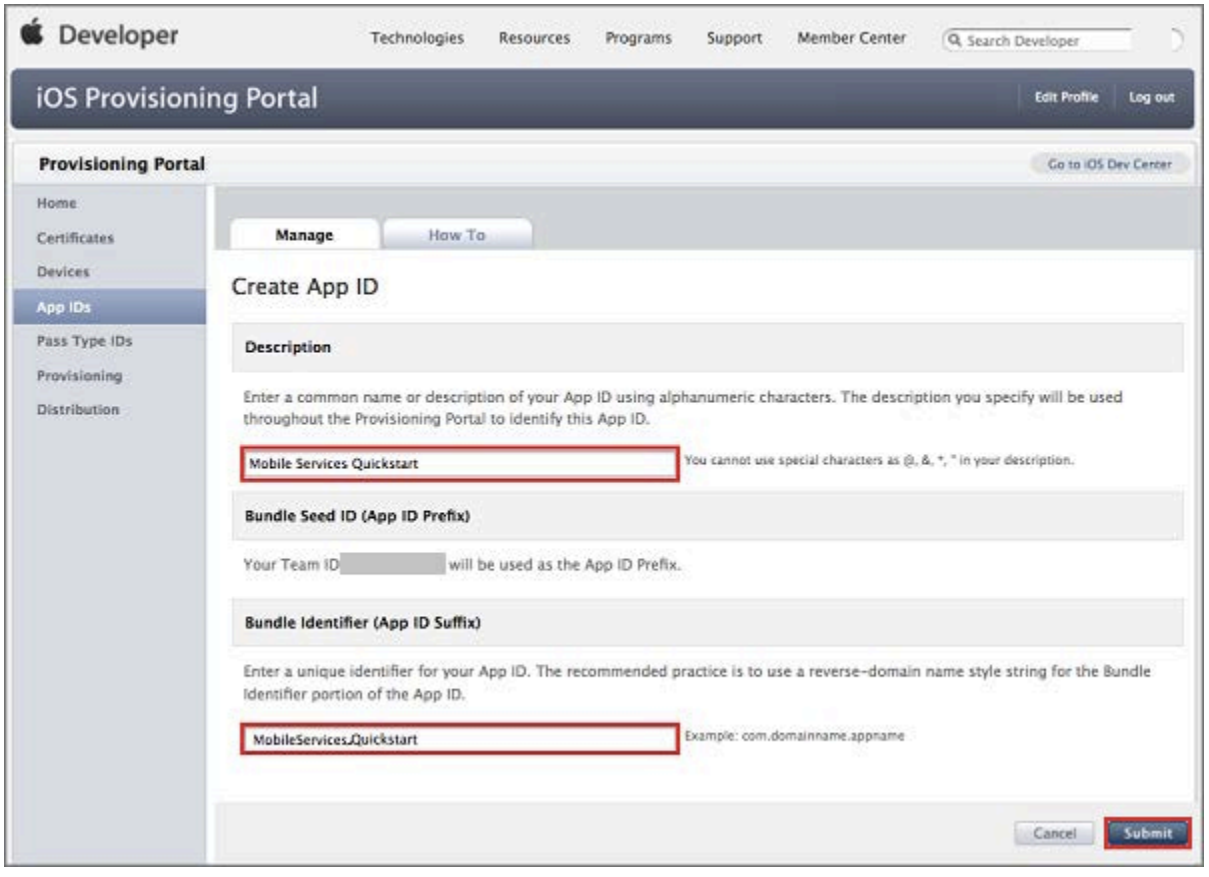
Register your app for push notifications

To be able to send push notifications to an iOS app from mobile services, you must register your application with Apple and also register for push notifications.

1. If you have not already registered your app, navigate to the iOS Provisioning Portal at the Apple Developer Center, log on with your Apple ID, click **App IDs**, then click **New App ID**.



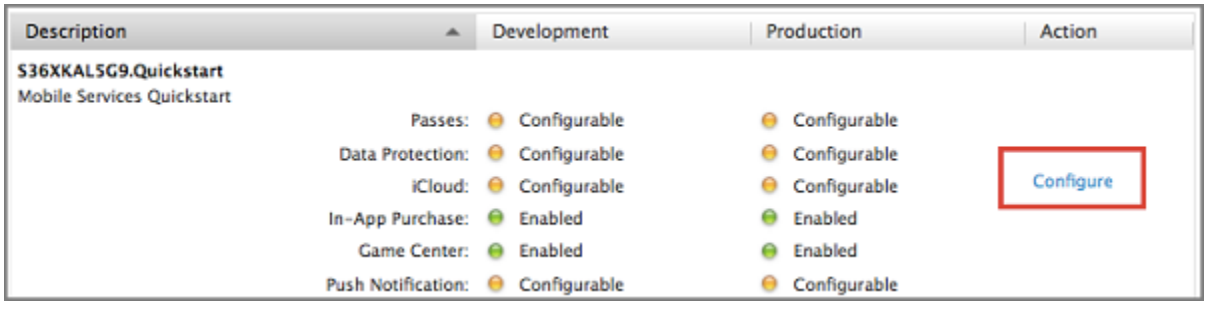
2. Type a name for your app in **Description**, enter the value *MobileServices.Quickstart* in **Bundle Identifier**, then click **Submit**.



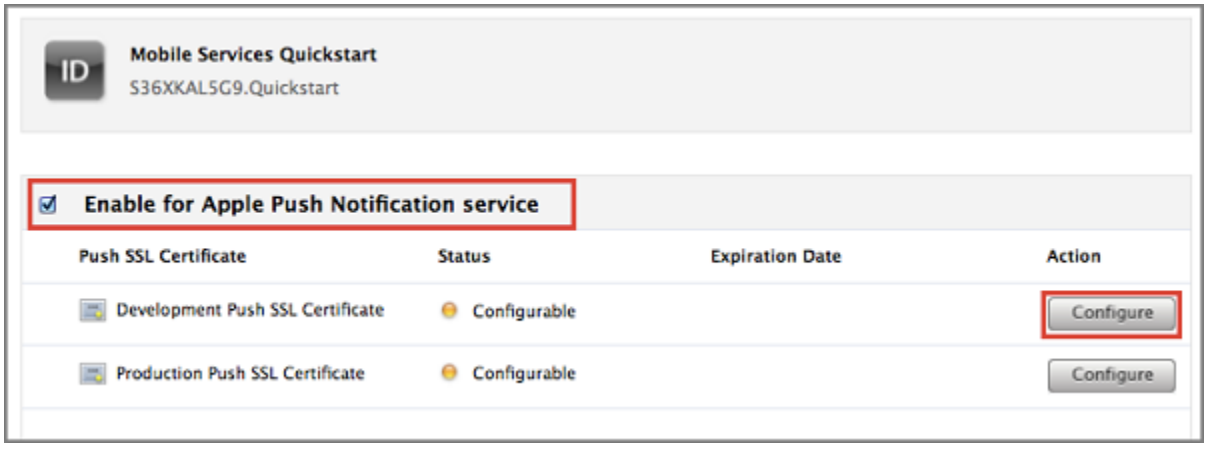
This generates your app ID.

> **Note:** If you choose to supply a **Bundle Identifier** value other thanMobileServices.Quickstart, you must also update the bundle identifier value in your Xcode project.

3. Locate the app ID that you just created, then click **Configure**.



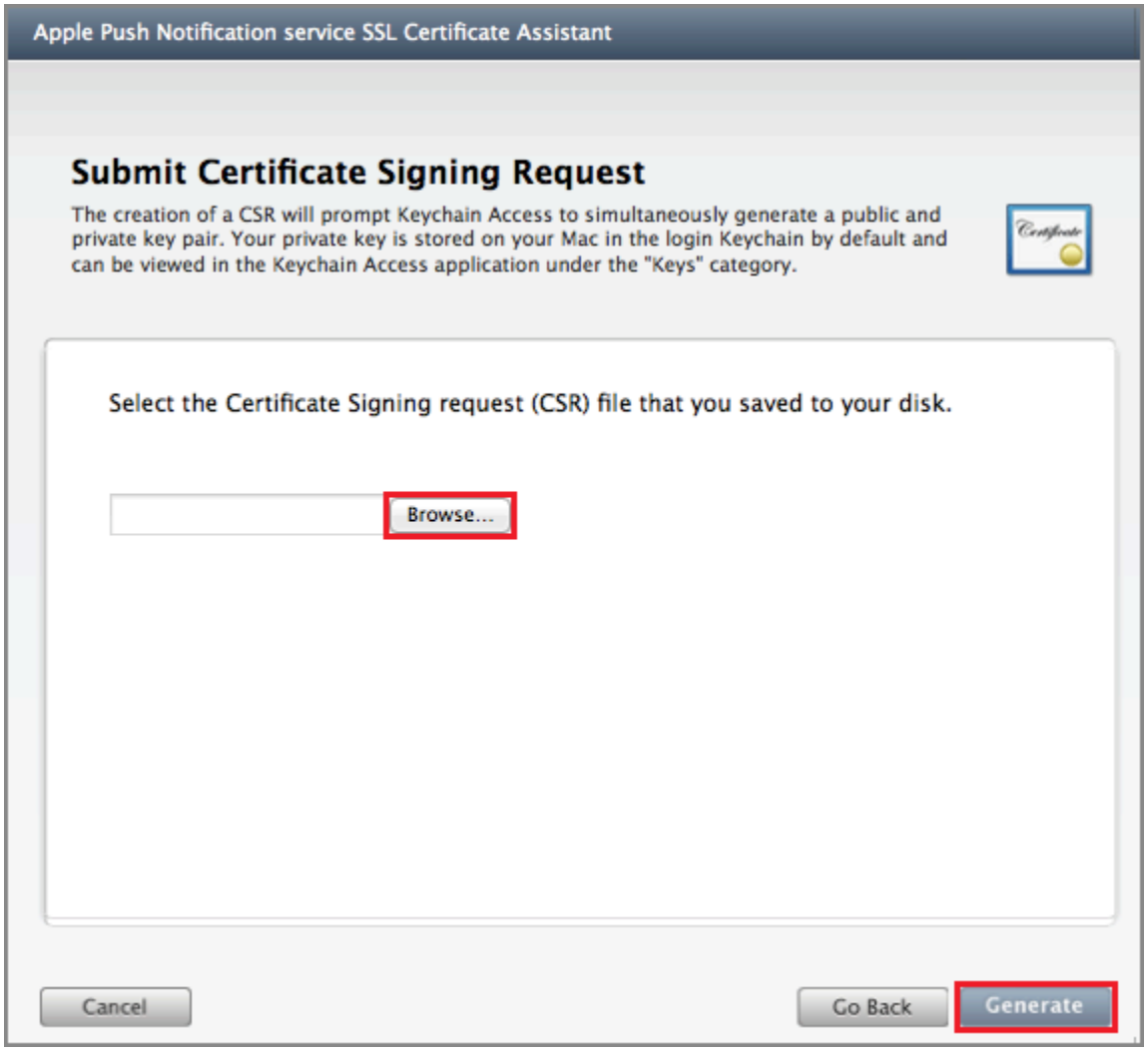
4. Check the **Enable for Apple Push Notification service** check box, then click the **Continue** button for the **Development Push SSL Certificate**.



This displays the Apple Push Notification service SSL Certificate Assistant.

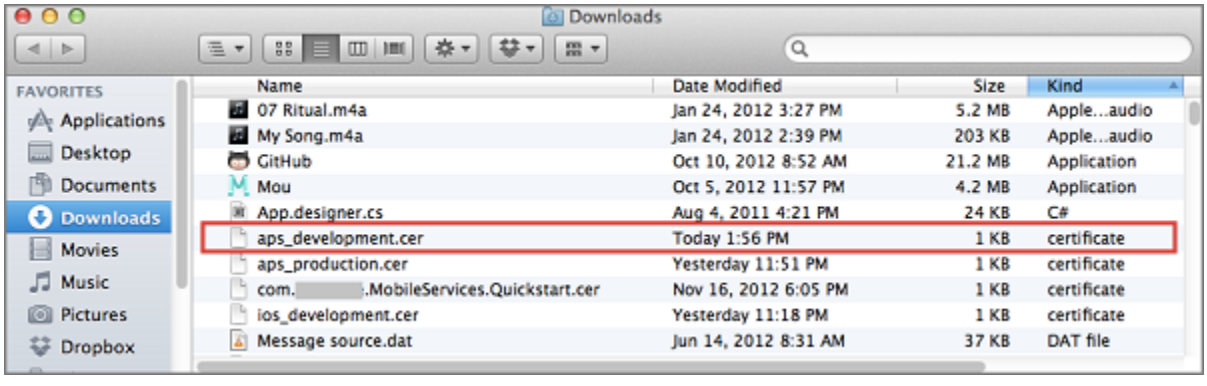
> **Note:** This tutorial uses a development certificate. The same process is used when registering a production certificate. Just make sure that you set the same certificate type when you upload the certificate to Mobile Services.

5. Click **Browse**, browse to the location where you saved the CSR file that you created in the first task, then click **Generate**.



6. After the certificate is created by the portal, click **Continue** and on the next screen click **Download**.

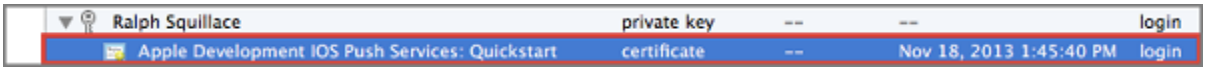
   This downloads the signing certificate and saves it to your computer in your Downloads folder.



Note: By default, the downloaded file a development certificate is named **aps_development.cer**.

7. Double-click the downloaded push certificate **aps_development.cer**.

   This installs the new certificate in the Keychain, as shown below:



   Note: The name in your certificate might be different, but it will be prefixed with **Apple Development iOS Push Notification Services:**.

   Later, you will use this certificate to generate a .p12 file and upload it to Mobile Services to enable authentication with APNS.

Create a provisioning profile for the app

1. Back in the iOS Provisioning Portal, select **Provisioning**, then click **New Profile**.

2. Enter a **Profile Name**, select the **Certificates** and **Devices** to use for testing, select the **App ID**, then click **Submit**.

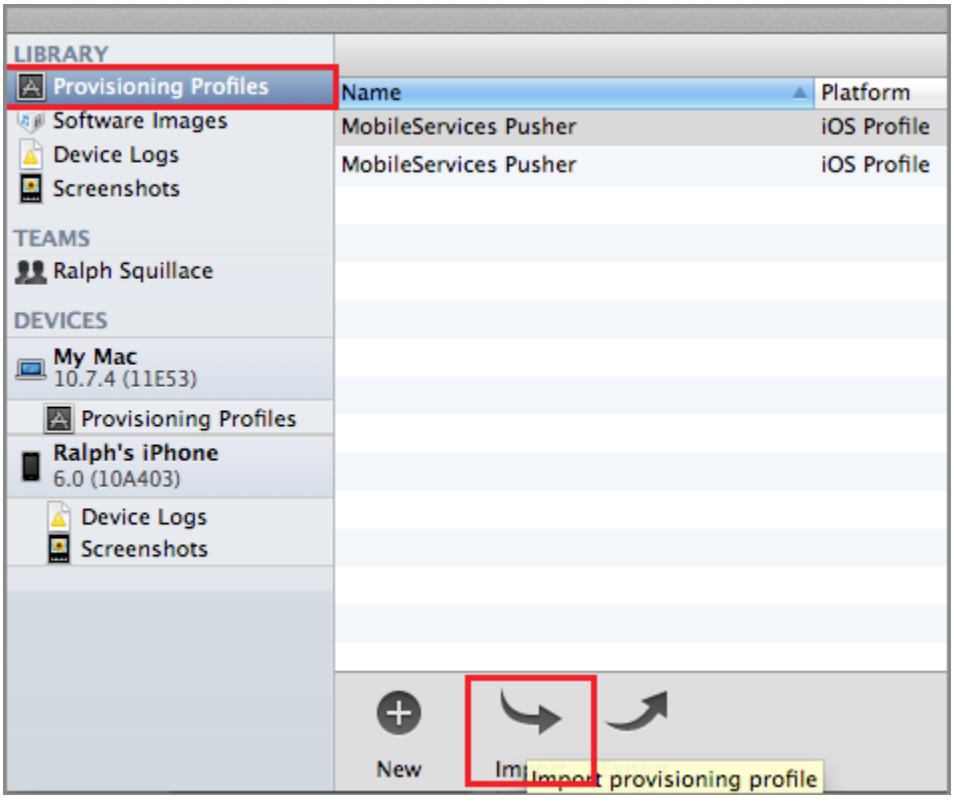
This creates a new provisioning profile.

3. From the list of provisioning profiles, click the **Download** button for this new profile.

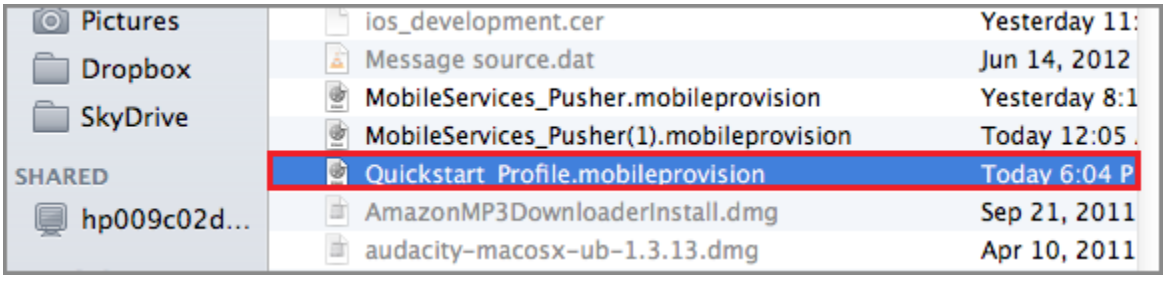
This downloads the profile to the local computer.

> **Note:** You may need to refresh the page to see the new profile.

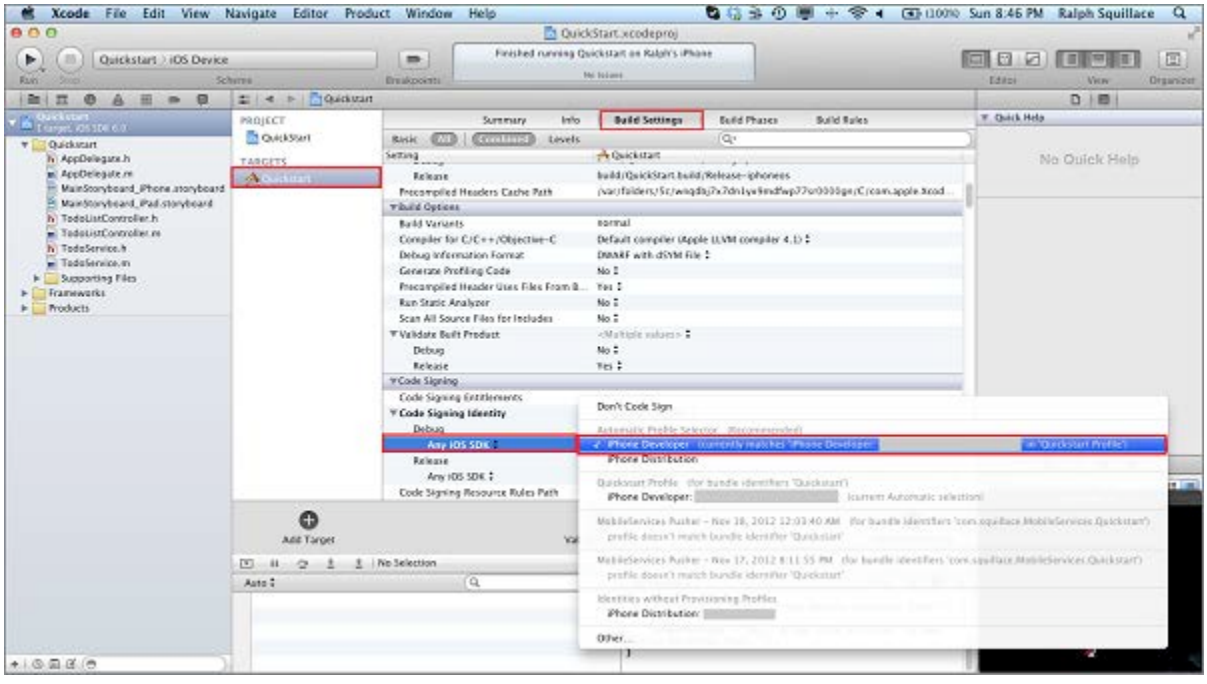
4. In Xcode, open the Organizer select the Devices view, select **Provisioning Profiles** in the **Library** section in the left pane, and then click the **Import** button at the very bottom of the middle pane.



5. Locate the downloaded provisioning profile and click **Open**.



6. Under **Targets**, click **Quickstart**, expand **Code Signing Identity**, then under **Debug** select the new profile.

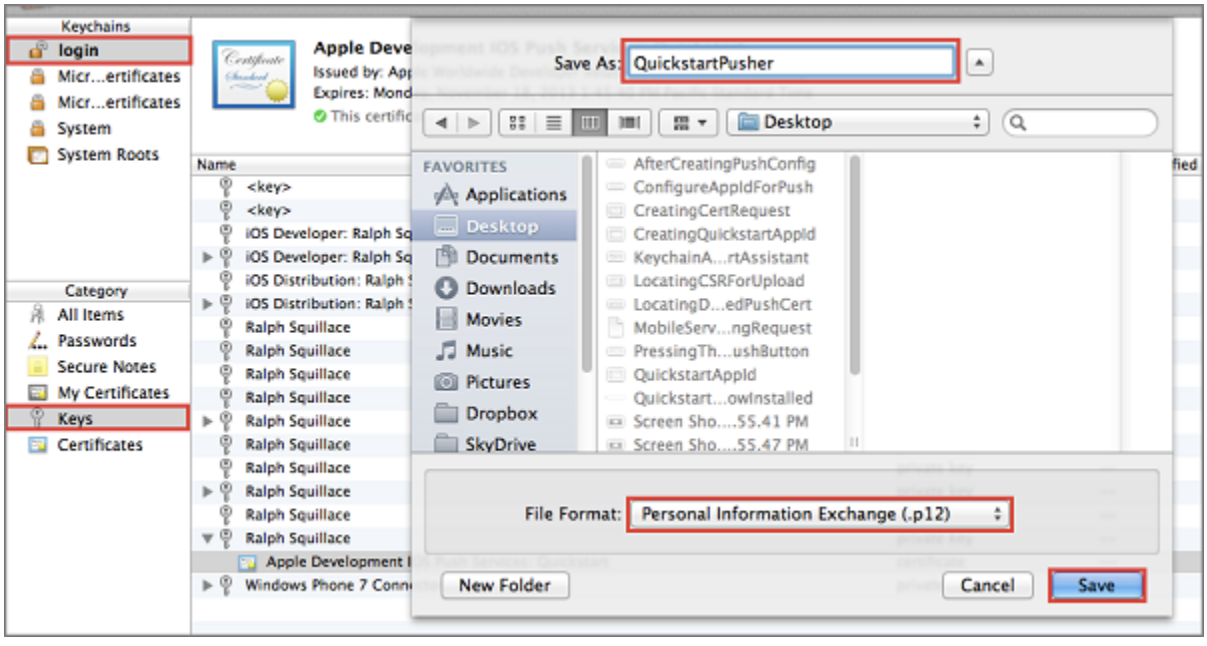


This ensures that the Xcode project uses the new profile for code signing. Next, you must upload the certificate to Mobile Services.

Configure Mobile Services to send push requests

After you have registered your app with APNS and configured your project, you must next configure your mobile service to integrate with APNS.

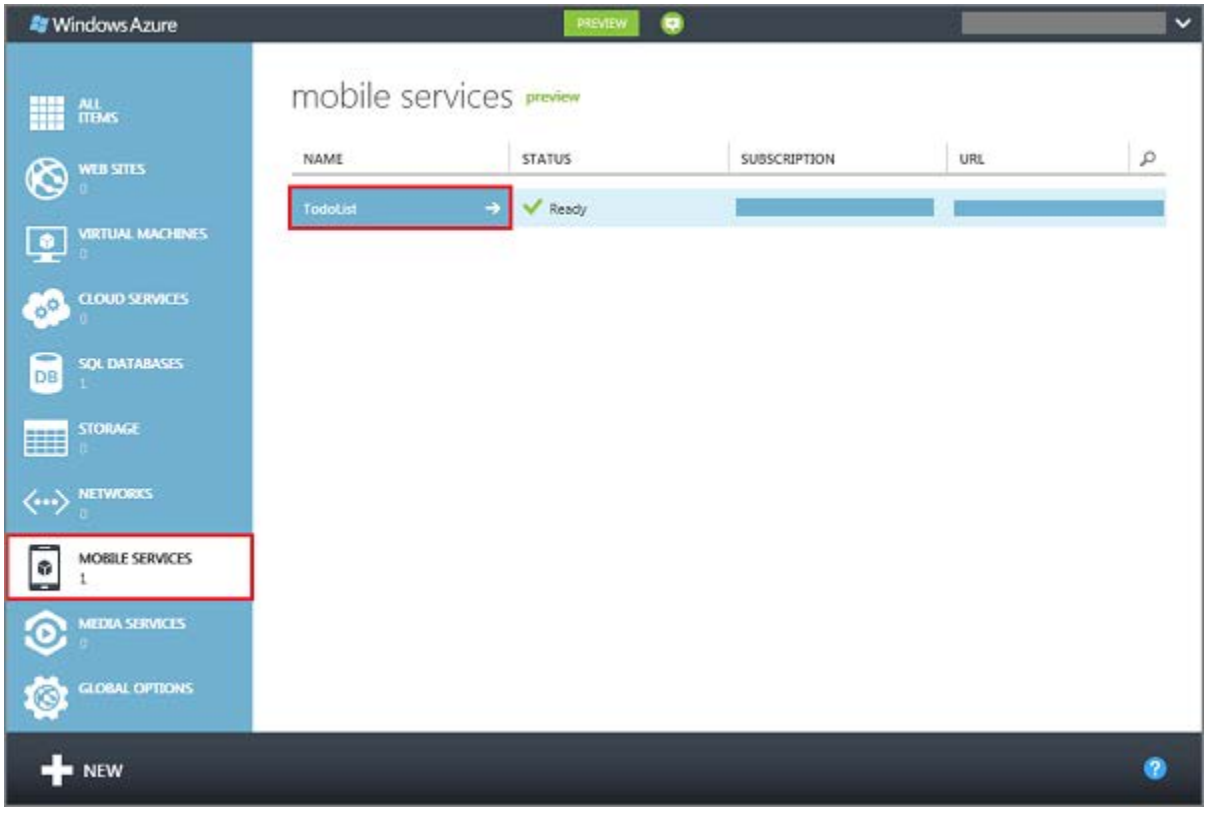
1. In Keychain Access, right-click the new certificate, click **Export**, name your file QuickstartPusher, select the **.p12** format, then click **Save**.



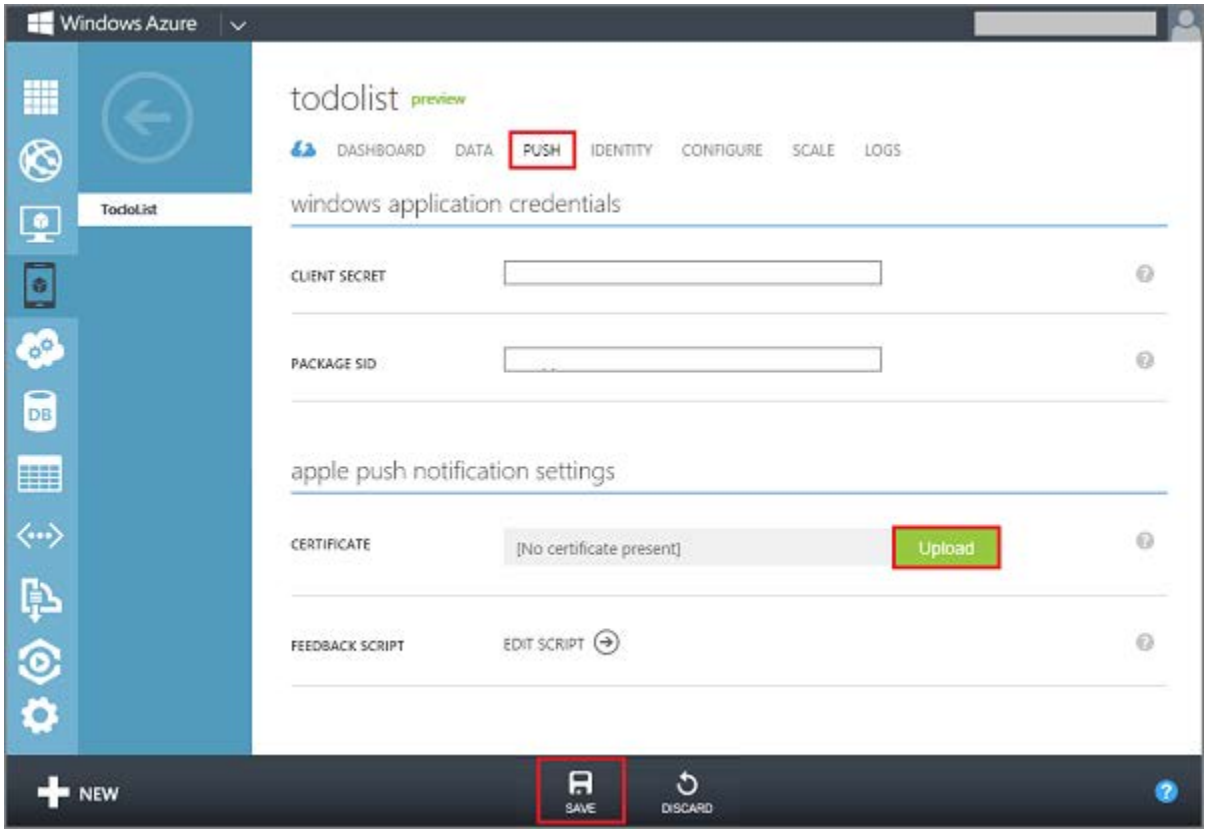
Make a note of the file name and location of the exported certificate.

> **Note:** This tutorial creates a QuickstartPusher.p12 file. Your file name and location might be different.

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

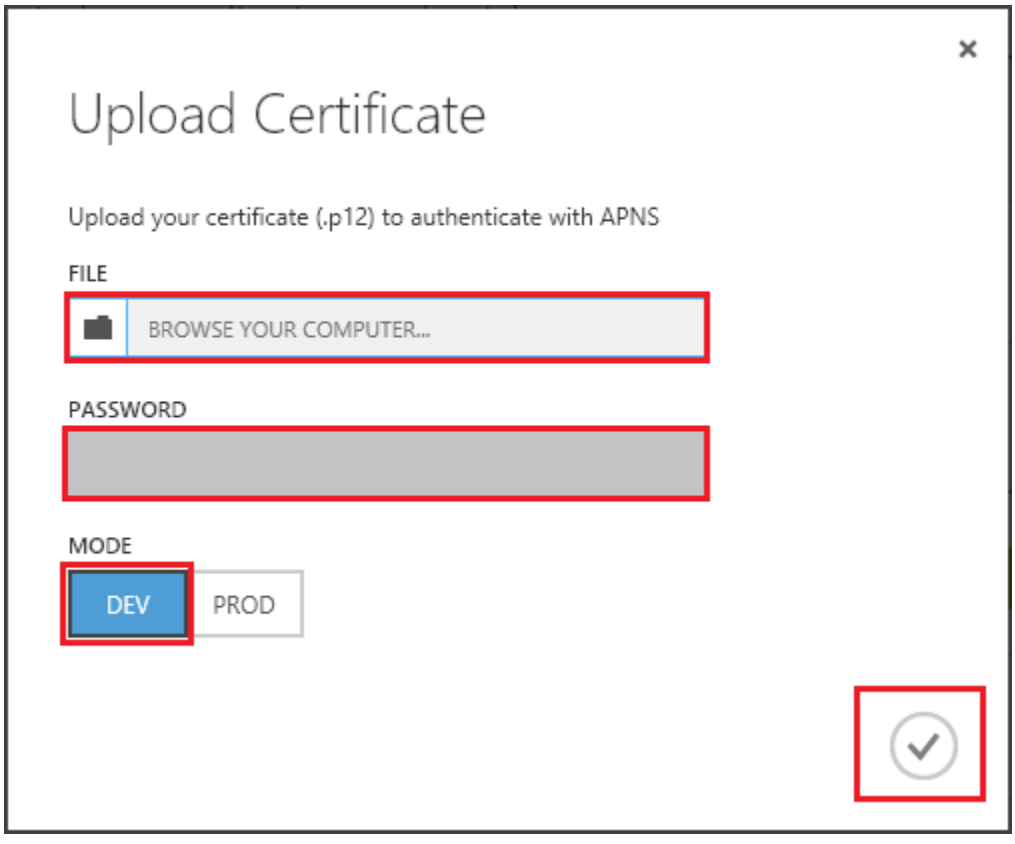


3. Click the **Push** tab and click **Upload**.



This displays the Upload Certificate dialog.

4. Click **File**, select the exported certificate QuickstartPusher.p12 file, enter the **Password**, make sure that the correct **Mode** is selected, click the check icon, then click **Save**.



Note: This tutorial uses developement certificates.

Both your mobile service is now configured to work with APNS.

Add push notificationsAdd push notifications to your app

1. In Xcode, open the AppDelegate.h file and add the following property below the ***window** property:

```
@property (strong, nonatomic) NSString *deviceToken;
```

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

2. In AppDelegate.m, replace the following handler method inside the implementation:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{
    // Register for remote notifications
```

```
    [[UIApplication sharedApplication]
registerForRemoteNotificationTypes:
    UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge |
UIRemoteNotificationTypeSound];
    return YES;
}
```

3. In AppDelegate.m, add the following handler method inside the implementation:

```
// We are registered, so now store the device token (as a string) on
the AppDelegate instance
// taking care to remove the angle brackets first.
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:
(NSData *)deviceToken {
    NSCharacterSet *angleBrackets = [NSCharacterSet
characterSetWithCharactersInString:@"<>"];
    self.deviceToken = [[deviceToken description]
stringByTrimmingCharactersInSet:angleBrackets];
}
```

4. In AppDelegate.m, add the following handler method inside the implementation:

```
// Handle any failure to register. In this case we set the deviceToken
to an empty
// string to prevent the insert from failing.
-(void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:
(NSError *)error {
NSLog(@"Failed to register for remote notifications: %@", error);
self.deviceToken = @"";
}
```

5. In AppDelegate.m, add the following handler method inside the implementation:

```
// Because toast alerts don't work when the app is running, the app
handles them.
// This uses the userInfo in the payload to display a UIAlertView.
- (void)application:(UIApplication *)application
didReceiveRemoteNotification:
(NSDictionary *)userInfo {
```

```
    NSLog(@"%@", userInfo);
    UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Notification" message:
    [userInfo objectForKey:@"inAppMessage"] delegate:nil
cancelButtonTitle:
    @"OK" otherButtonTitles:nil, nil];
    [alert show];
}
```

6. In TodoListController.m, import the AppDelegate.h file so that you can use the delegate to obtain the device token:

```
#import "AppDelegate.h"
```

7. In TodoListController.m, modify the **(IBAction)onAdd** action by locating the following line:

```
NSDictionary *item = @{ @"text" : itemText.text, @"complete" : @(NO) };
```

Replace this with the following code:

```
// Get a reference to the AppDelegate to easily retrieve the deviceToken
AppDelegate *delegate = [[UIApplication sharedApplication] delegate];


NSDictionary *item = @{
    @"text" : itemText.text,
    @"complete" : @(NO),
    // add the device token property to our todo item payload
    @"deviceToken" : delegate.deviceToken
};
```

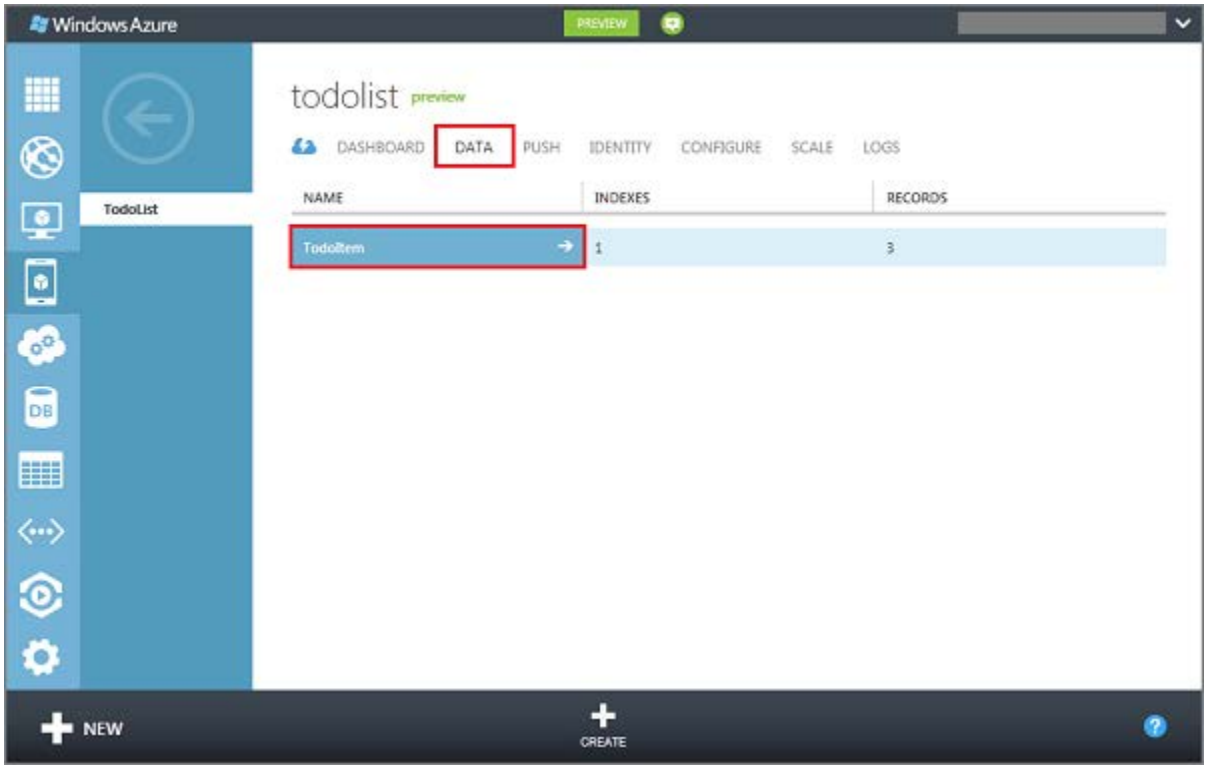
This adds a reference to the **AppDelegate** to obtain the device token and then modifies the request payload to include that device token.

> **Note:** You must add this code before to the call to the **addItem** method.

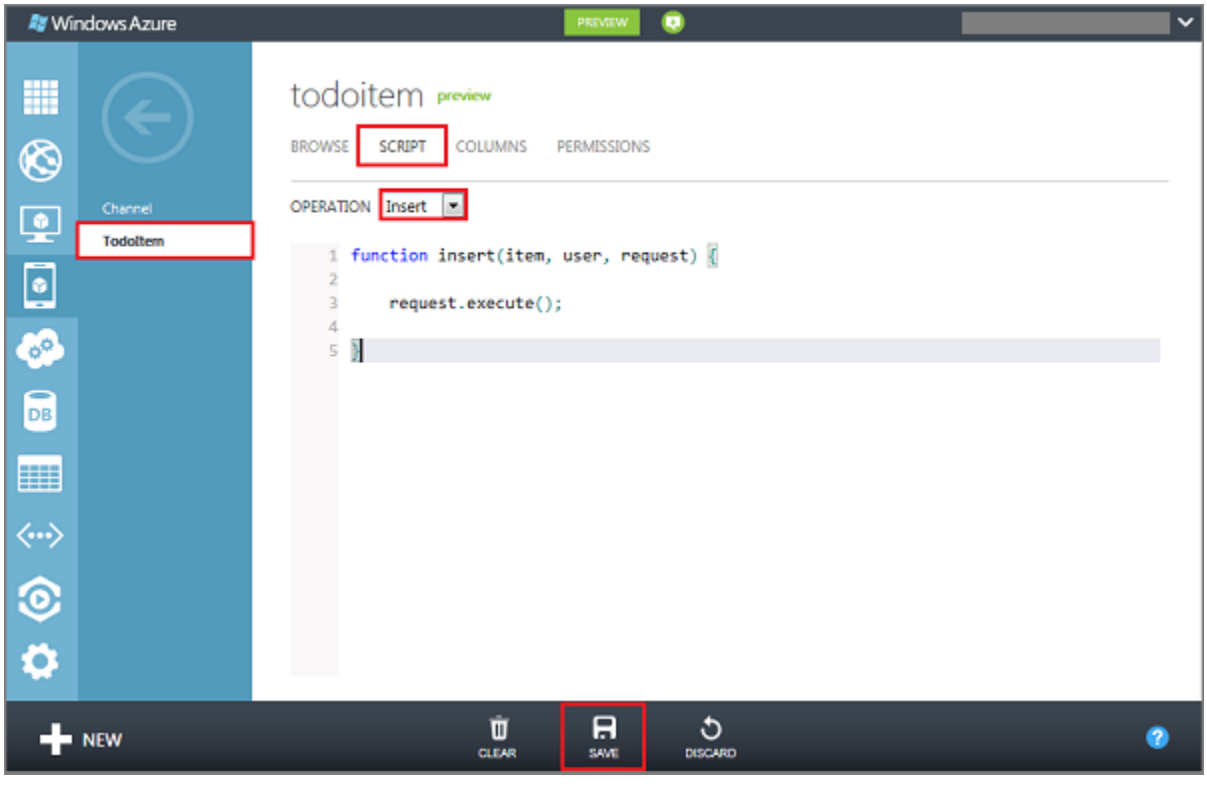
Your app is now updated to support push notifications.

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**:

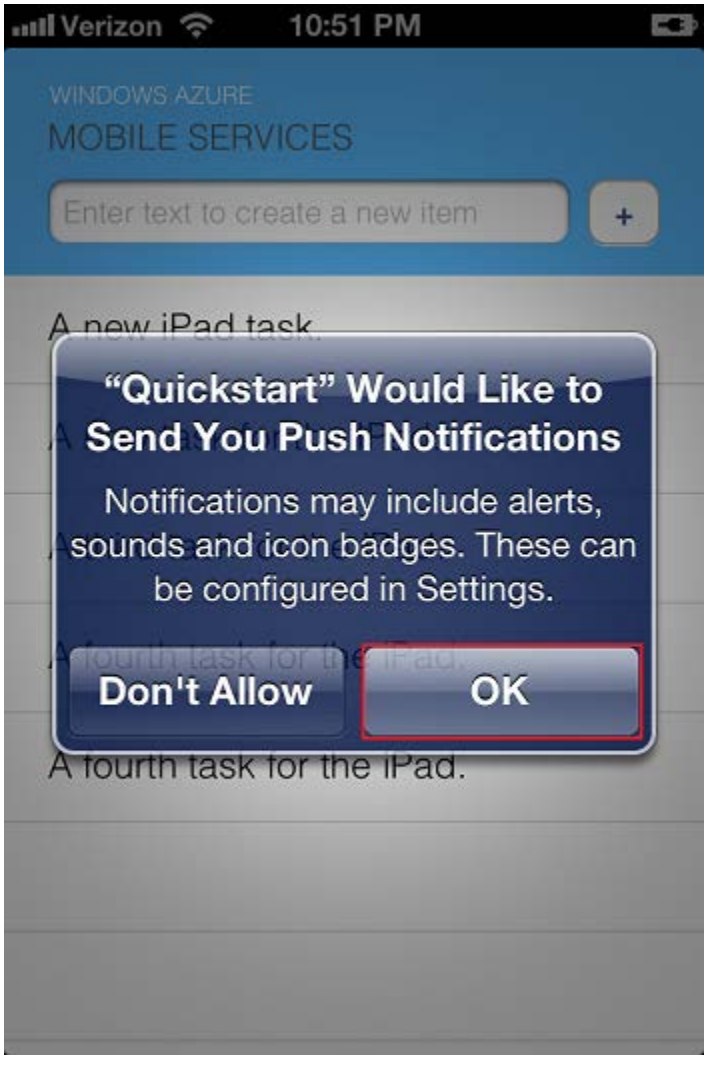
```
function insert(item, user, request) {
    request.execute();
    // Set timeout to delay the notification, to provide time for the
    // app to be closed on the device to demonstrate toast
notifications
    setTimeout(function() {
        push.apns.send(item.deviceToken, {
            alert: "Toast: " + item.text,
            payload: {
                inAppMessage: "Hey, a new item arrived: '" + item.text
+ "'"
            }
        });
    }, 2500);
}
```

This registers a new insert script, which uses the apns object to send a push notification (the inserted text) to the device provided in the insert request.

> **Note:** This script delays sending the notification to give you time to close the app to receive a toast notification.

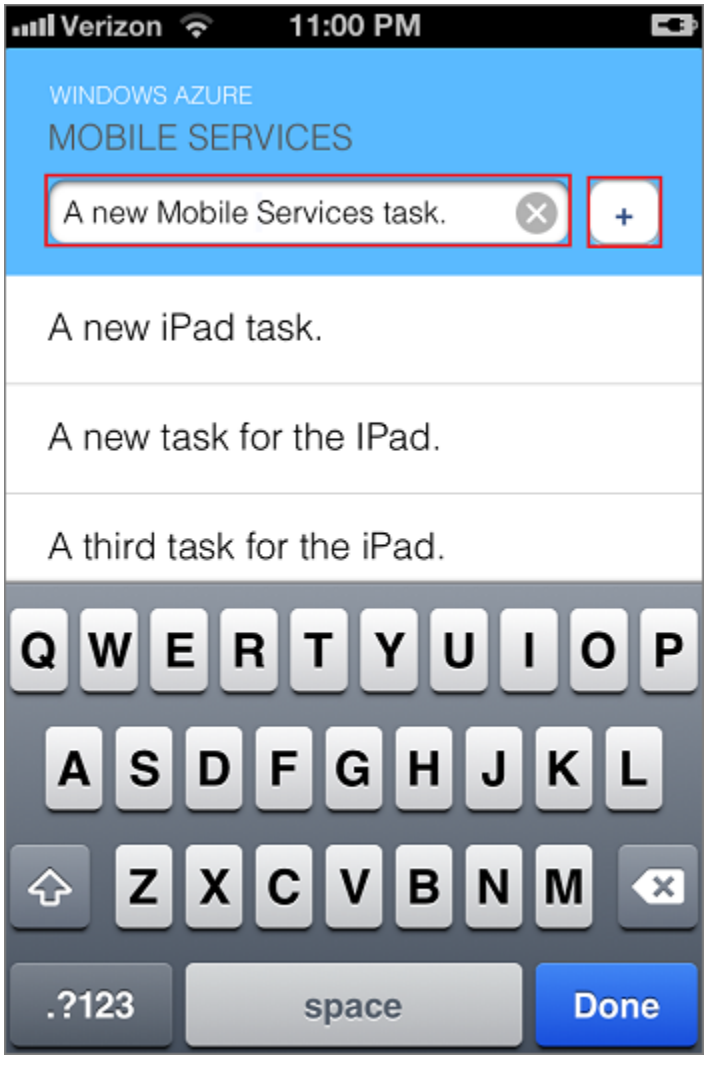
Test push notifications in your app

1. Press the **Run** button to build the project and start the app in an iOS capable device, then click **OK** to accept push notifications



> **Note:** You must explicitly accept push notifications from your app. This request only occurs the first time that the app runs.

2. In the app, type meaningful text, such as *A new Mobile Services task* and then click the plus (**+**) icon.



3. Verify that a notification is received, then click **OK** to dismiss the notification.

4. Repeat step 2 and immediately close the app, then verify that the following toast is shown.

You have successfully completed this tutorial. In this simple example a user receives a push notification with the data that was just inserted. The device token used by APNS is supplied to the mobile service by the client in the request. In the next tutorial, you will create a separate Devices table in which to store device tokens and then send a push notification out to all stored tokens 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 Apple Push Notification Service (APNS) tokens. These tokens can then be used to send push notifications to users of the iPhone or iPad app.

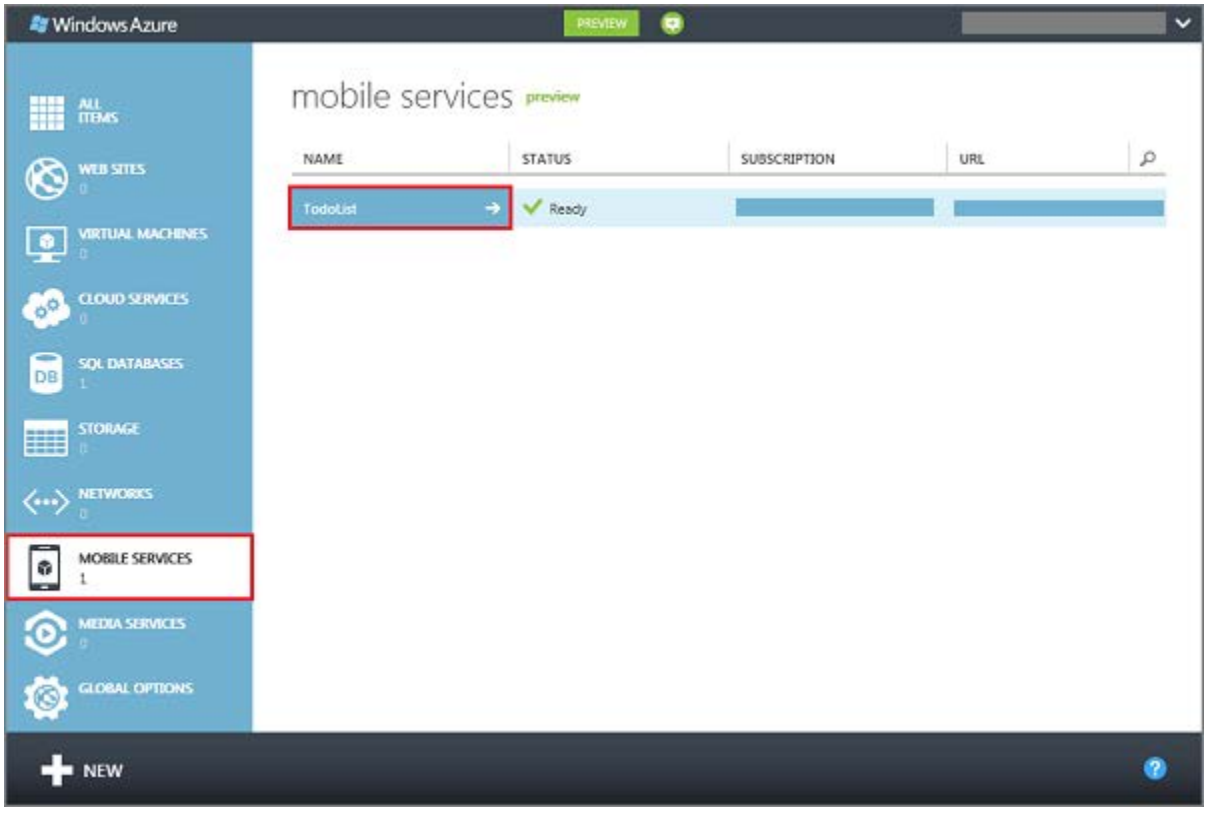
This tutorial walks you through these steps to update push notifications in your app:

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

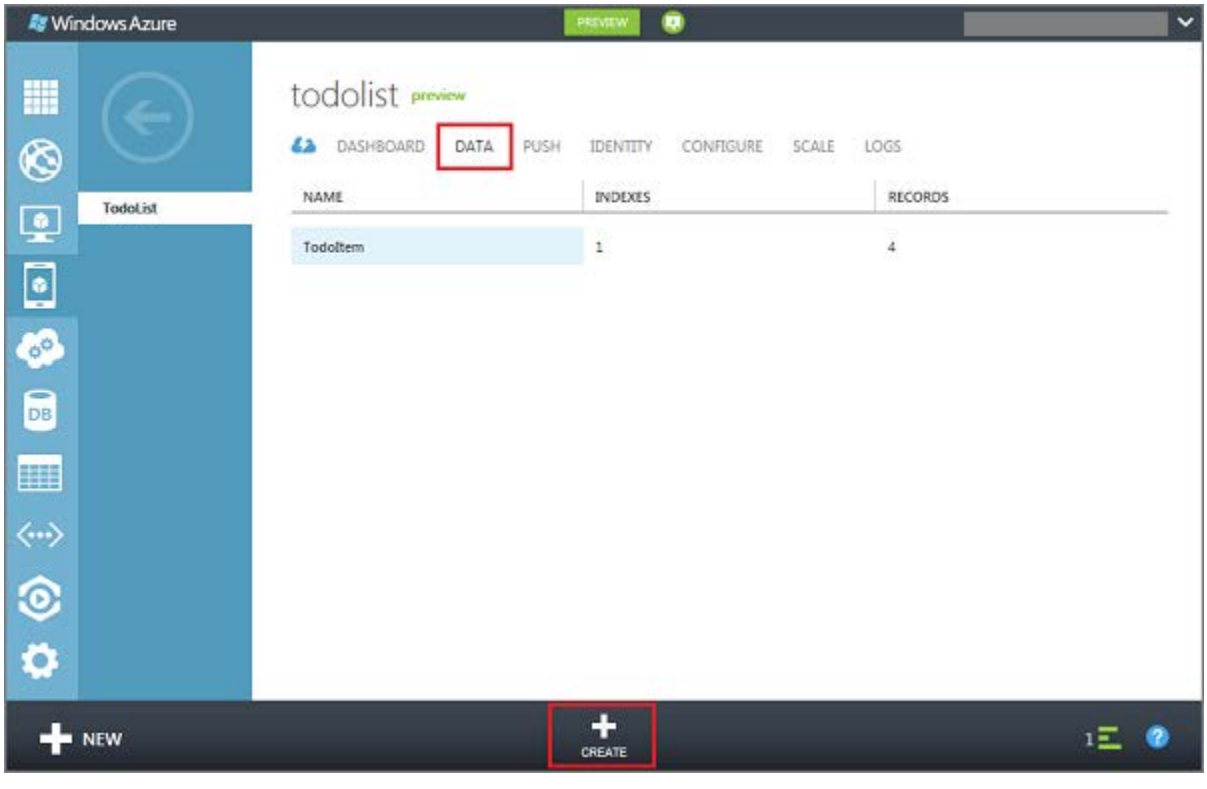
This tutorial is based on the Mobile Services quickstart and builds on the previous tutorial Get started with push notifications. Before you start this tutorial, you must first complete Get started with push notifications.

Create the new Devices table

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

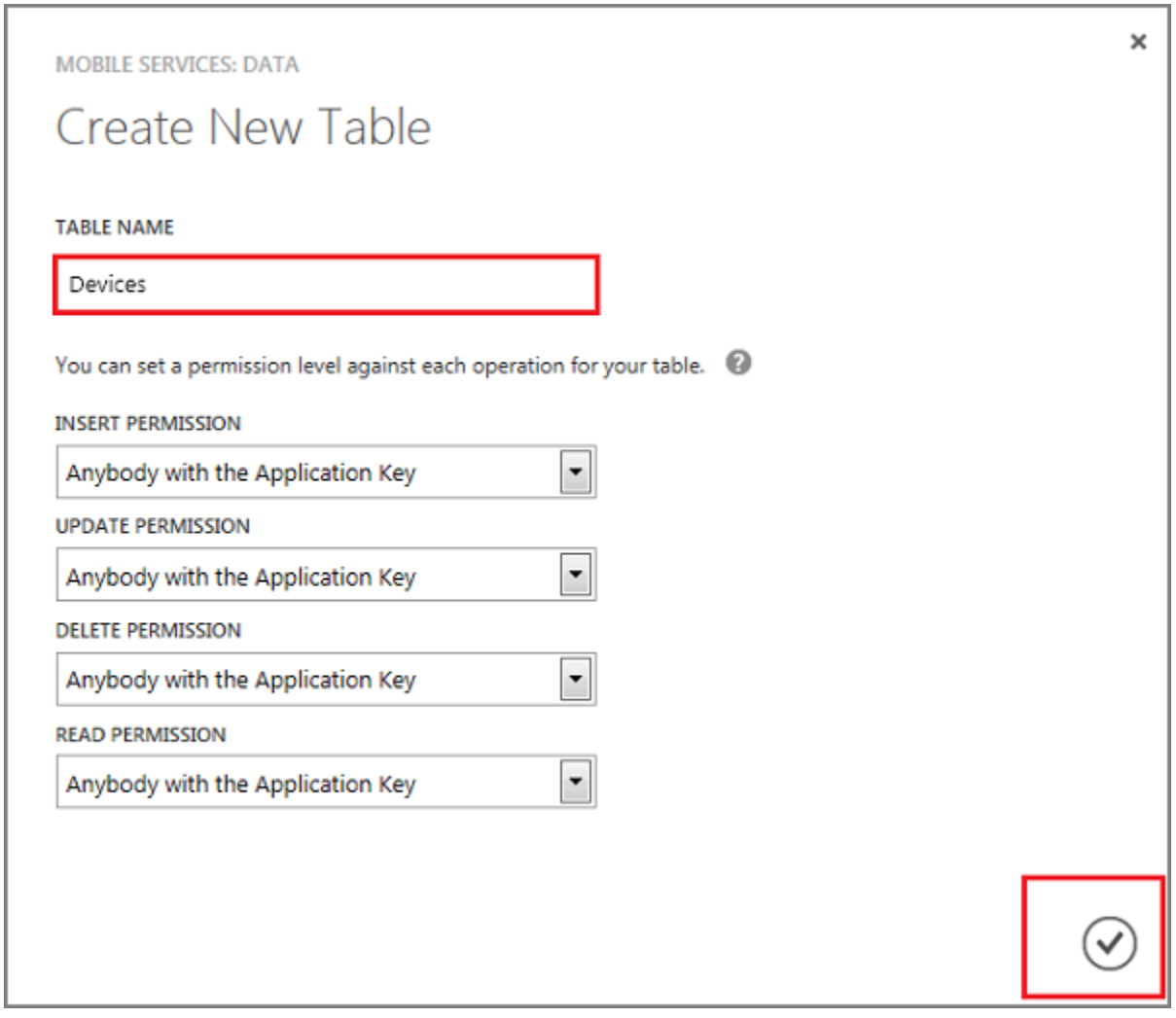


2. Click the **Data** tab, and then click **Create**.



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

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



This creates the **Devices** table, which stores the device tokens 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 Xcode, open the TodoService.h file and add the following method declarations:

```
// Declare the singleton instance for other users
```

```
+ (TodoService *) getCurrent;


// Declare method to register device token for other users
-(void) registerDeviceToken:(NSString *)deviceToken;
```

This enables other callers to get an instance of the TodoService and register a deviceToken with the Mobile Service.

2. In TodoService.m, add the following variable and static method inside the @implementation of the TodoService:

```
// Add a variable to support Singleton creation.
TodoService *instance;


// Add static method to return TodoService instance.
+ (TodoService *)getCurrent
{
    if (instance == nil) {
        instance = [[TodoService alloc] init];
    }
    return instance;
}
```

This enables the singleton pattern for the TodoService class.

3. In TodoService.m, underneath the preceding code, add the following instance method:

```
// Instance method to register deviceToken in Devices table.
// Called in AppDelegate.m when APNS registration succeeds.
- (void)registerDeviceToken:(NSString *)deviceToken
{
    MSTable* devicesTable = [self.client getTable:@"Devices"];
    NSDictionary *device = @{ @"deviceToken" : deviceToken };


// Insert the item into the devices table and add to the items array on
completion
[devicesTable insert:device completion:^(NSDictionary *result, NSError
*error) {
```

```
    if (error) {
        NSLog(@"ERROR %@", error);
    }
}];


}
```

This allows other callers to register the device token with Mobile Services.

4. In the AppDelegate.m file, add the following import statement:

```
#import "TodoService.h"
```

This code makes the AppDelegate aware of the TodoService implementation.

5. In AppDelegate.m, replace the **didRegisterForRemoteNotificationsWithDeviceToken** method with the following code:

```
// We have registered, so now store the device token (as a string) on
the AppDelegate instance
// taking care to remove the angle brackets first.
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:
(NSData *)deviceToken {


   // Register the APNS deviceToken with the Mobile Service Devices
table.
   NSCharacterSet *angleBrackets = [NSCharacterSet
characterSetWithCharactersInString:@"<>"];
   NSString *token = [[deviceToken description]
stringByTrimmingCharactersInSet:angleBrackets];


   TodoService *instance = [TodoService getCurrent];
   [instance registerDeviceToken:token];
}
```

6. In the TodoListController.m file, in the **(void)viewDidLoad** method, locate the following line of code:

```
self.todoService = [[TodoService alloc]init];
```

Replace this with the following code:

```
// Create the todoService.
self.todoService = [TodoService getCurrent];
```

This creates the Mobile Service client inside the wrapped service using the new singleton.

7. In TodoListController.m, locate the **(IBAction)onAdd** method and remove the following code:

```
// Get a reference to the AppDelegate to easily retrieve the
deviceToken
AppDelegate *delegate = [[UIApplication sharedApplication] delegate];


NSDictionary *item = @{
    @"text" : itemText.text,
    @"complete" : @(NO),
    // add the device token property to our todo item payload
    @"deviceToken" : delegate.deviceToken
};
```

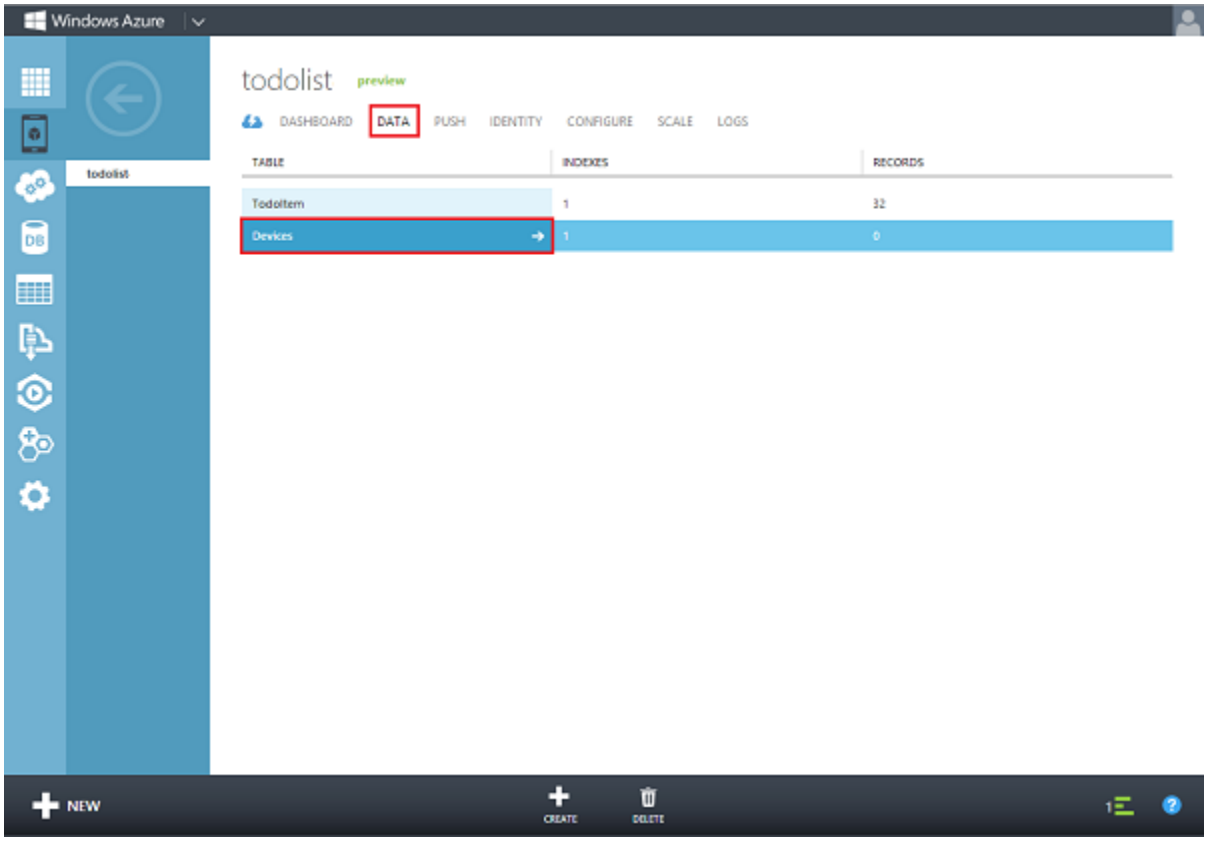
Replace this with the following code:

```
// We removed the delegate; this application no longer passes the
deviceToken here.
// Remove the device token from the payload
NSDictionary *item = @{ @"text" : itemText.text, @"complete" : @(NO) };
```

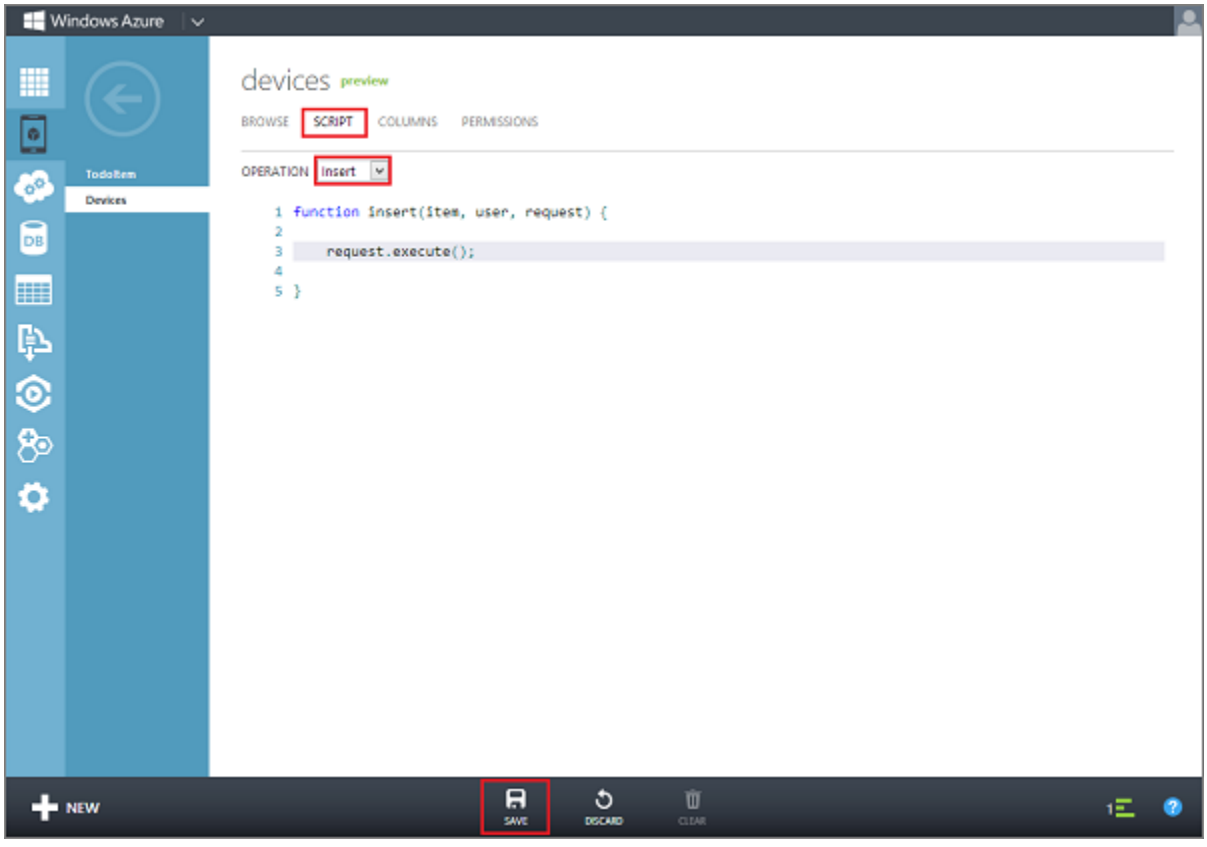
Your app has now been updated to use the new Devices table to store device tokens that are used to send push notifications back to the device.

Update server scripts

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



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



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

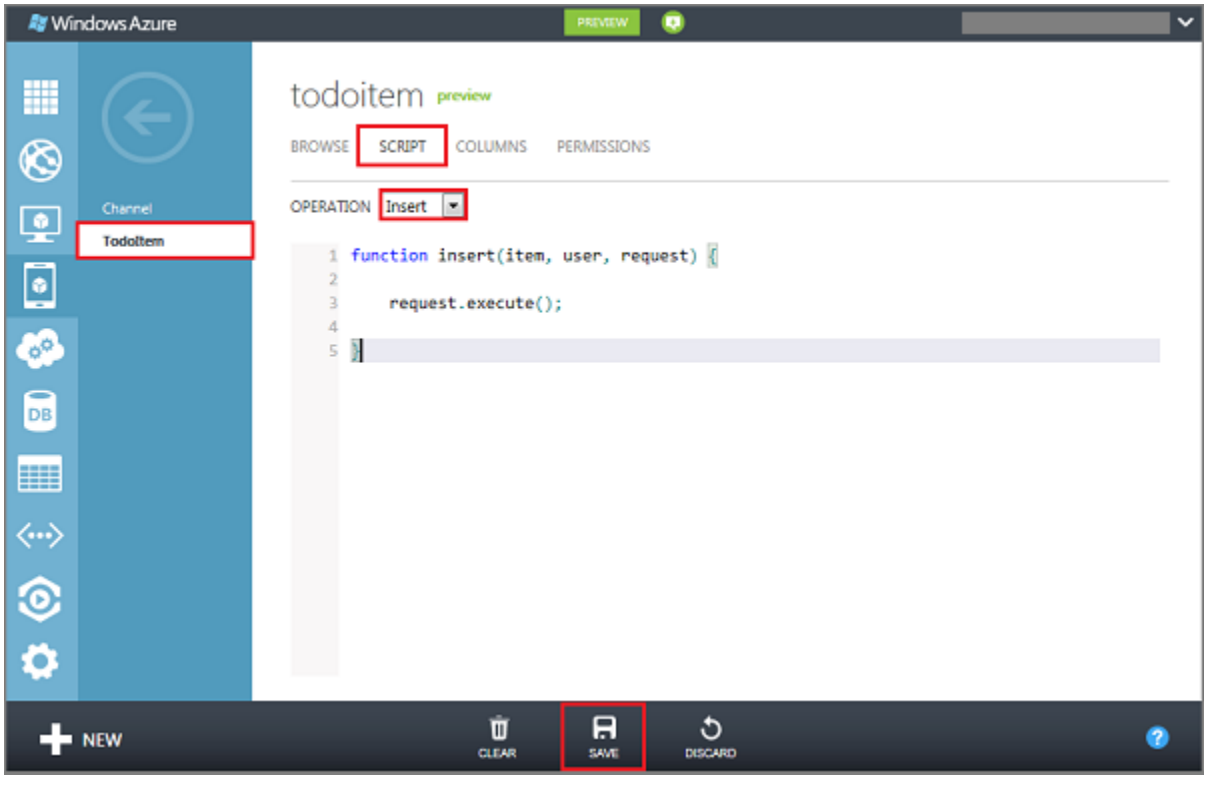
3. Replace the insert function with the following code, and then click **Save**:

```
function insert(item, user, request) {
    var devicesTable = tables.getTable('Devices');
    devicesTable.where({
        token: item.token
    }).read({
        success: insertTokenIfNotFound
    });


    function insertTokenIfNotFound(existingTokens) {
        if (existingTokens.length > 0) {
            request.respond(200, existingTokens[0]);
        } else {
            request.execute();
        }
    }
```

```
}
```

This script checks the **Devices** table for an existing device with the same token. The insert only proceeds when no matching device is found. This prevents duplicate device records.

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



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

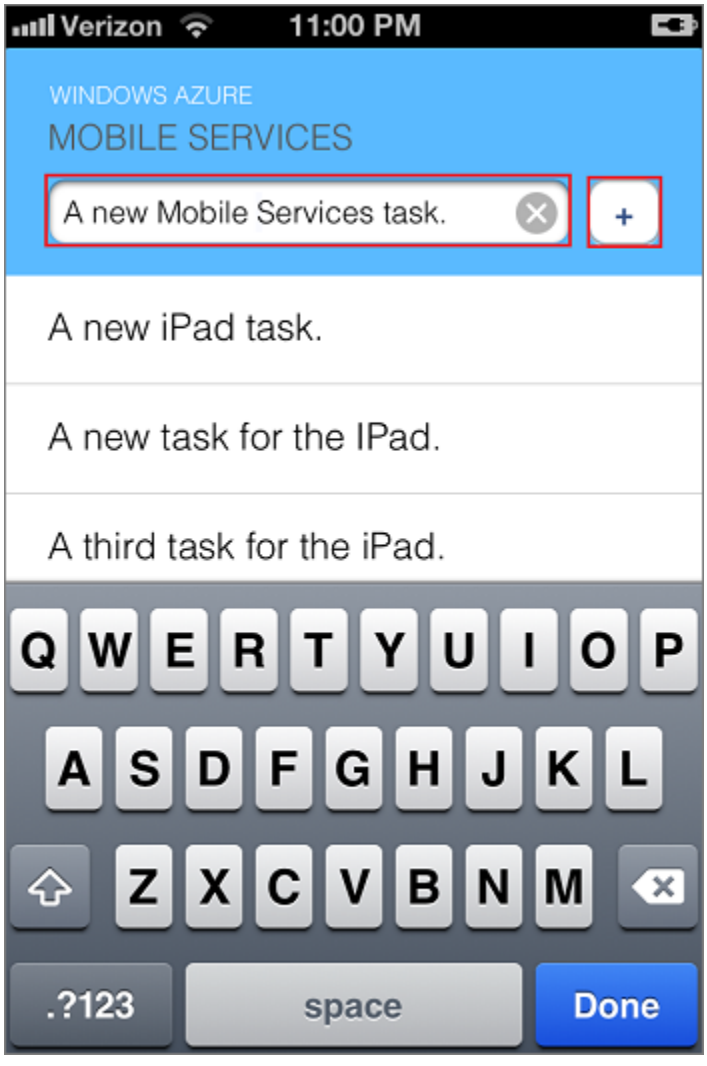
```
function insert(item, user, request) {
  request.execute({
      success: function() {
          request.respond();
          sendNotifications();
      }
  });


  function sendNotifications() {
      var devicesTable = tables.getTable('Devices');
      devicesTable.read({
          success: function(devices) {
```

```
            // Set timeout to delay the notifications,
            // to provide time for the app to be closed
            // on the device to demonstrate toast notifications.
            setTimeout(function() {
                devices.forEach(function(device) {


                push.apns.send(device.deviceToken, {
                    alert: "Toast: " + item.text,
                    payload: {
                        inAppMessage:
                        "Hey, a new item arrived: '" +
                        item.text + "'"
                    }
                });
            });
        }, 2500);
    }
});


}
}
```

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

Test push notifications in your app

1. Press the **Run** button to build the project and start the app in an iOS capable device, then in the app, type meaningful text, such as *A new Mobile Services task* and then click the plus (**+**) icon.



2. Verify that a notification is received, then click **OK** to dismiss the notification.

3. Repeat step 2 and immediately close the app, then verify that the following toast is shown.

You have successfully completed this tutorial.

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.

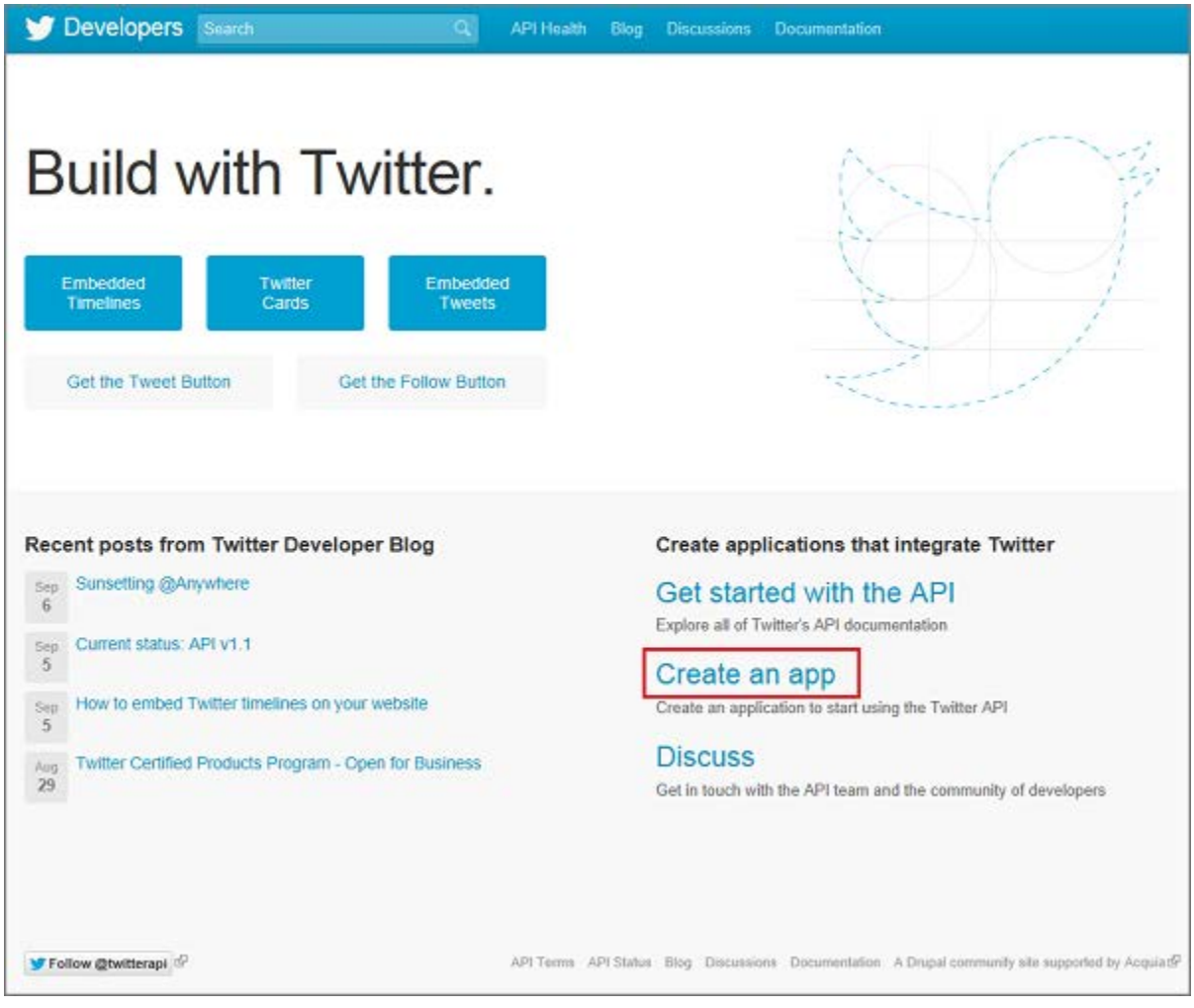
Mobile Services technical references (http://aka.ms/zumodocs)
Reference documentation for Mobile Services client libraries and server scripts.

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

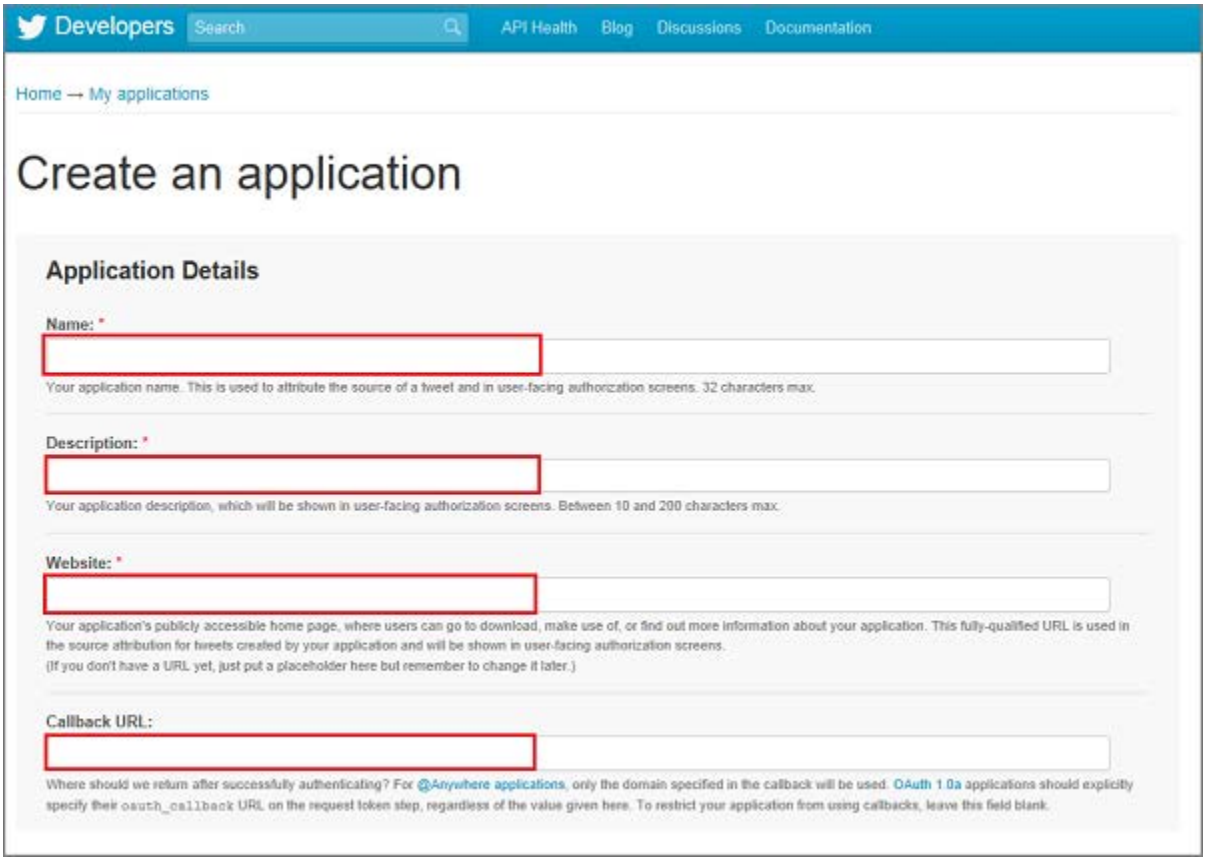
This appendix shows you how to register your apps to be able to use Twitter to authenticate with Windows Azure Mobile Services.

> **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**.

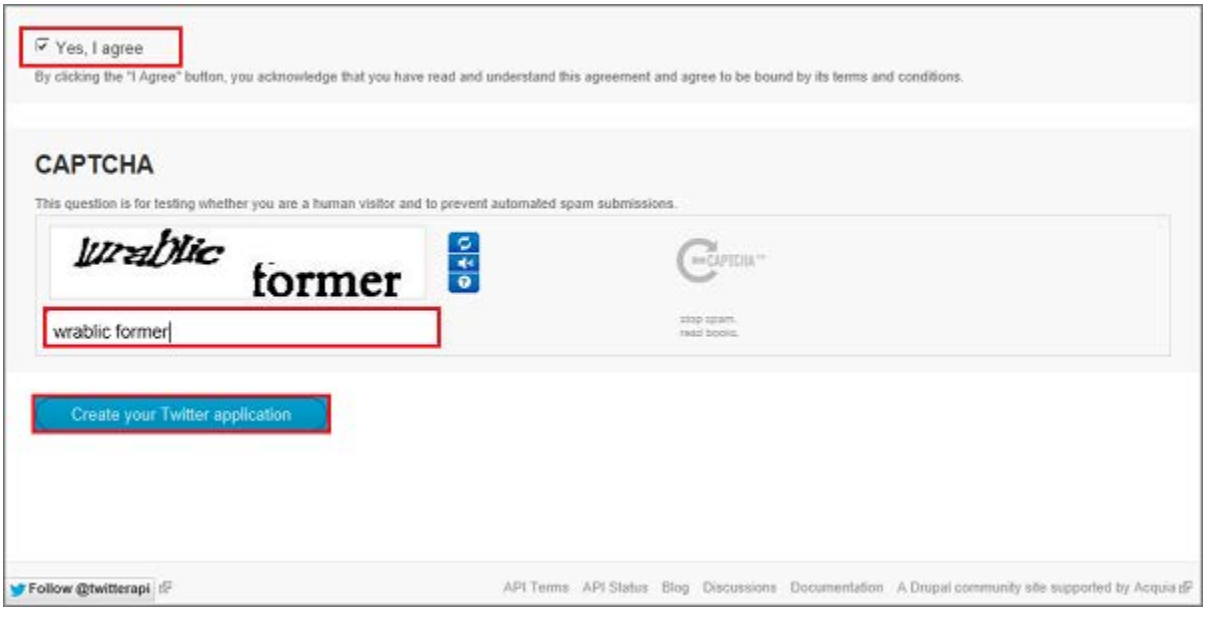


2. Type the **Name**, **Description**, and **Website** values for your app, and type the URL of the mobile service in **Callback URL**.



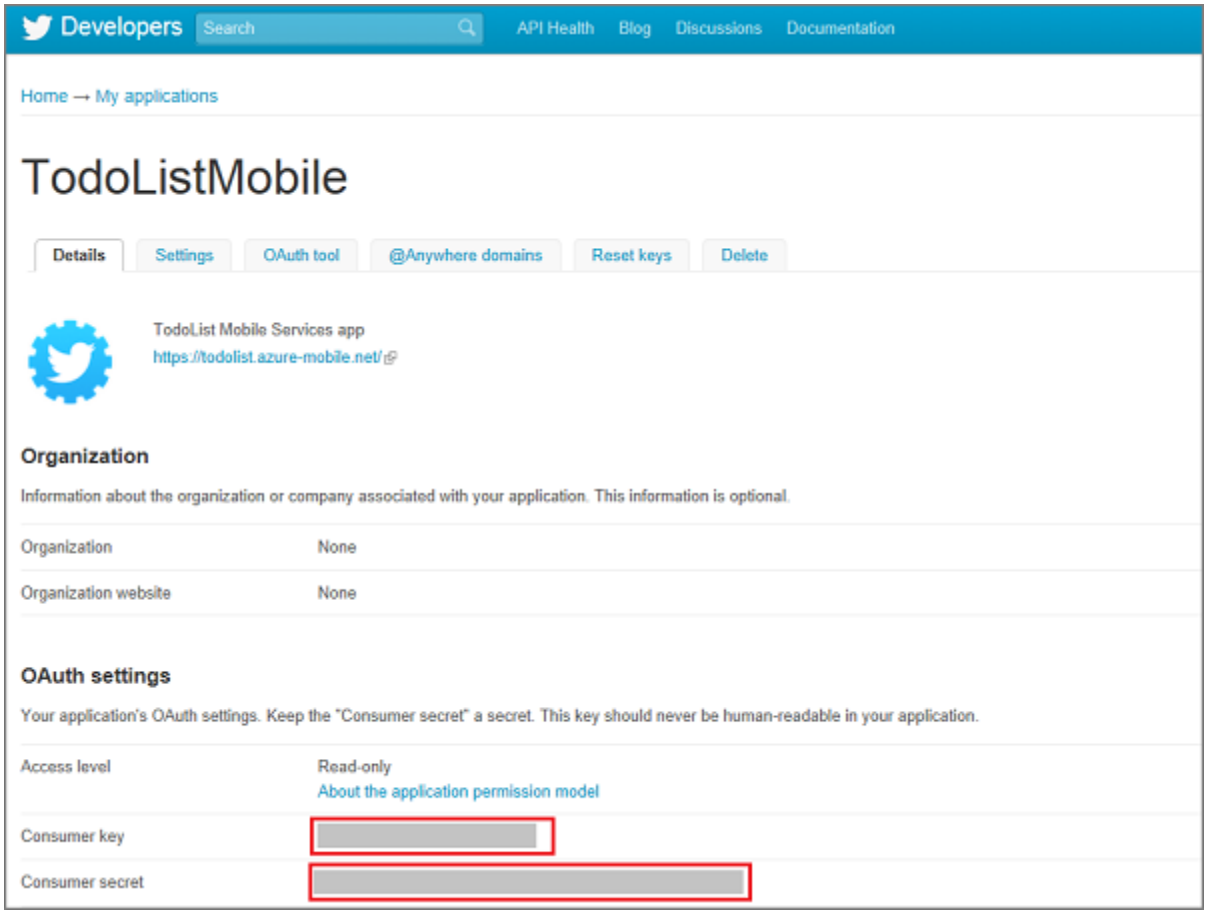
Note: The **Website** value is required but is not used.

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.

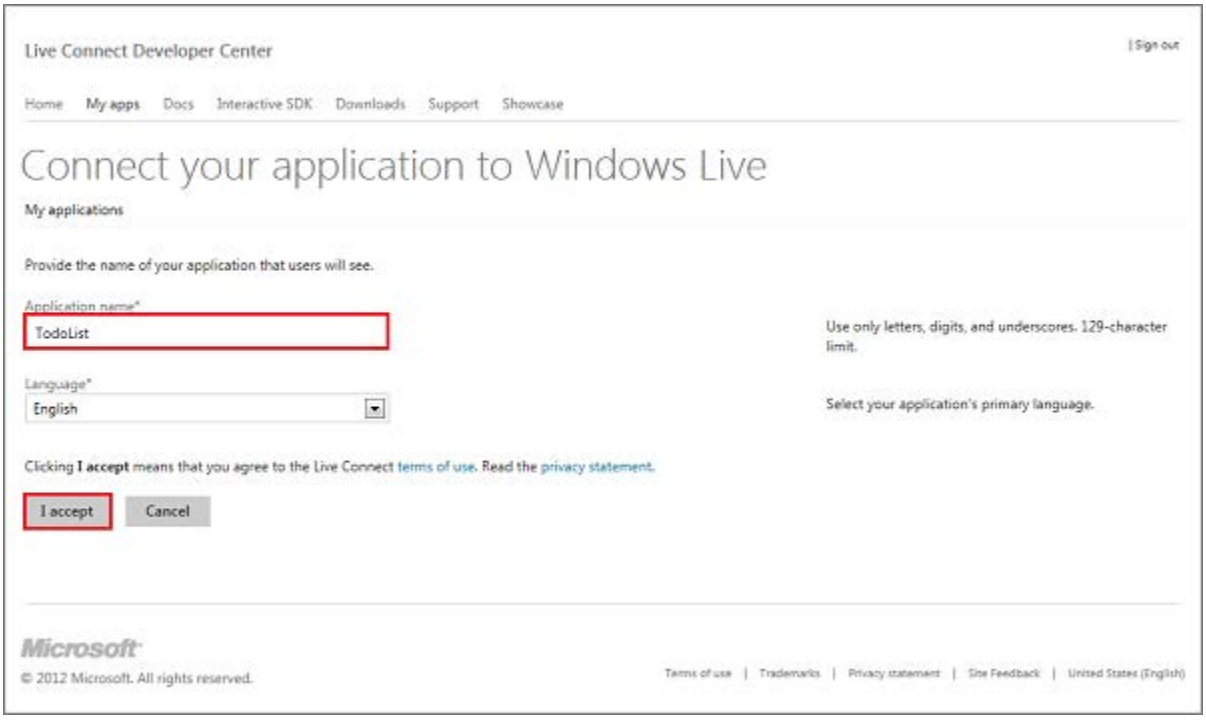
You are now ready to use a Twitter login for authentication in your app by providing the consumer key and consumer secret values to Mobile Services.

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.

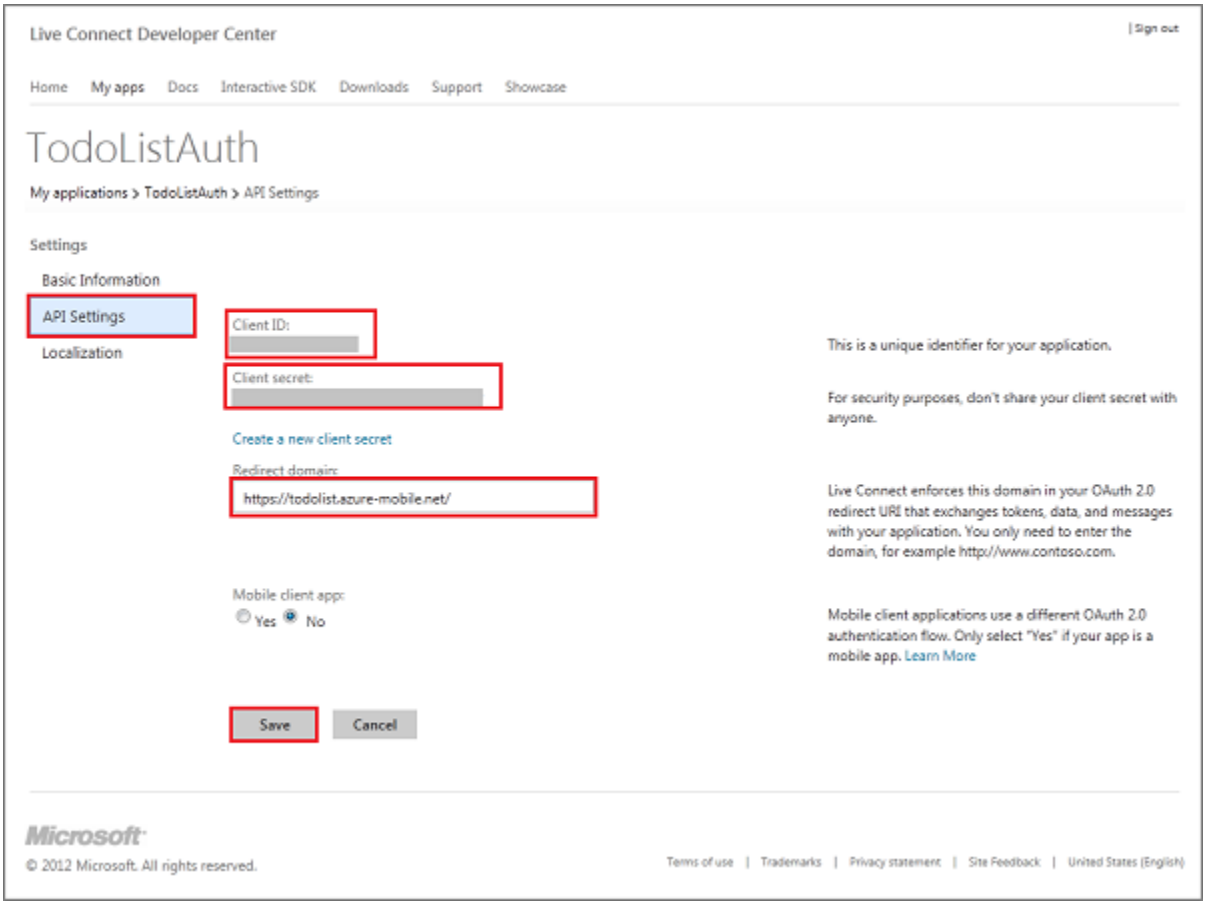
> **Note:** When you intend to also provide single sign-on or push notifications from a Windows Store app, consider also registering your app with the Windows Store. For more information, see Register your Windows Store apps for Windows Live Connect authentication.

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.

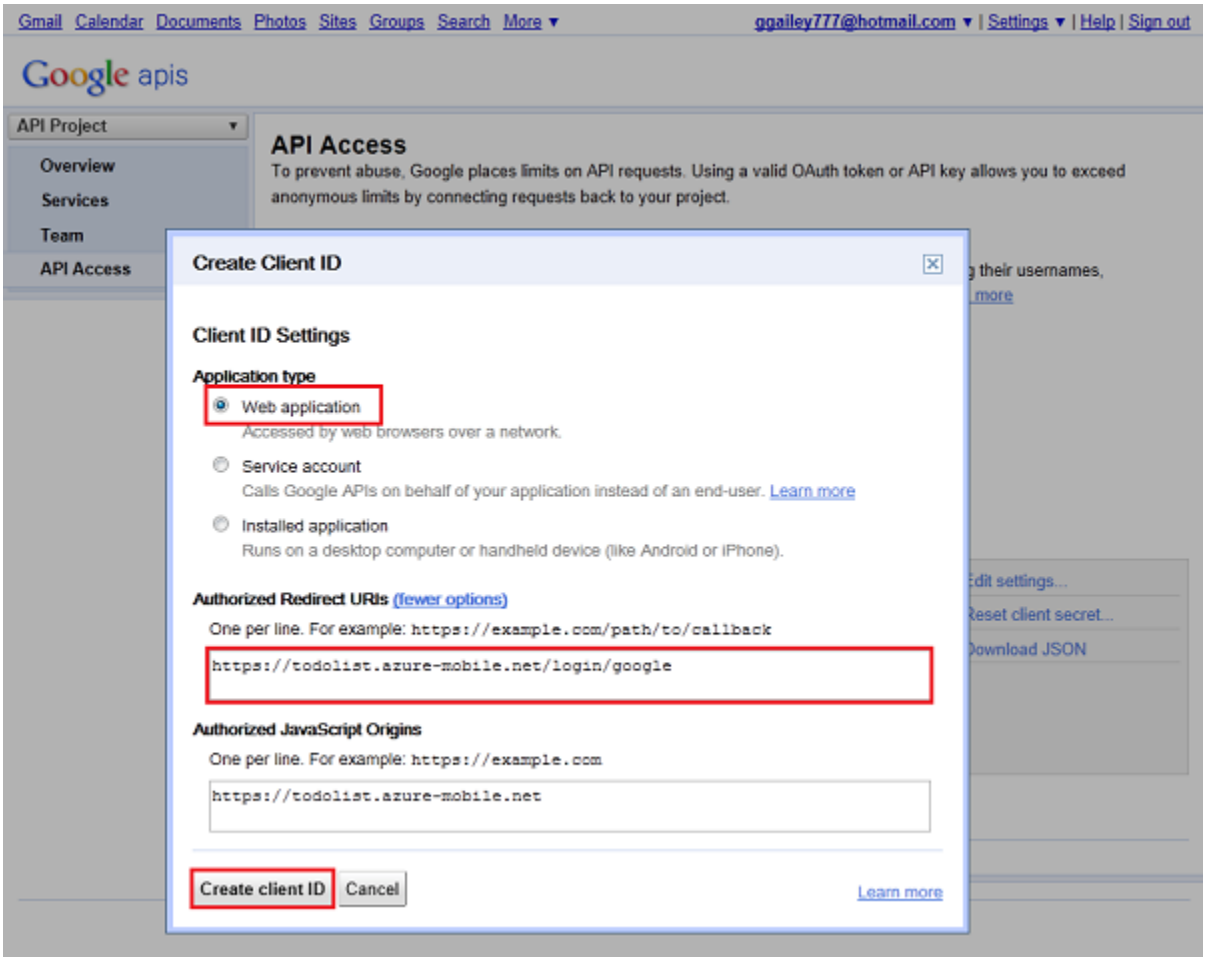
> **Note:** To complete the procedure in this topic, you must have a Google account that has a verified email address. To create a new Google account, go to accounts.google.com.

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

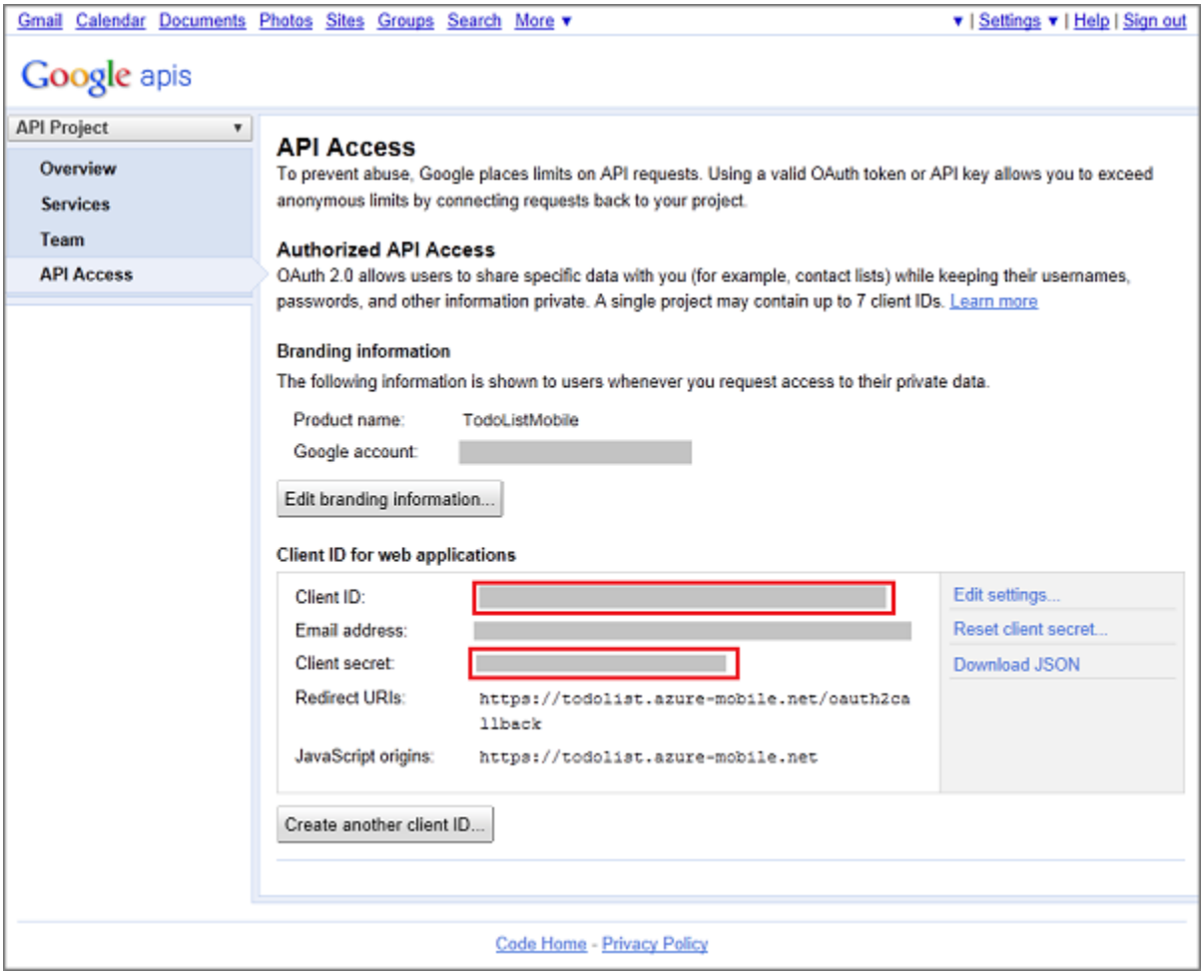
2. Click **API Access** and then click **Create an OAuth 2.0 client ID...**.

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.