

How to Send Daily SMS Reminders Using C#, Azure Functions and Twilio



by Brent Schooley (<https://www.twilio.com/blog/author/brent-schooley>) on January 24, 2017 (<https://www.twilio.com/blog/2017/01/how-to-send-daily-sms-reminders-using-c-azure-functions-and-twilio.html>)

[Like](#)
[Tweet](#)
[Follow @twilio](#)

Are you remembering to keep up with your New Year's Resolutions? Using C#, [Azure Functions](https://azure.microsoft.com/en-us/services/functions/) (<https://azure.microsoft.com/en-us/services/functions/>) and [Twilio](http://twilio.com/docs) (<http://twilio.com/docs>) we'll build a service that delivers daily [SMS](https://www.twilio.com/docs/api/rest/sending-messages) (<https://www.twilio.com/docs/api/rest/sending-messages>) reminders to help keep up with those new goals.

Recipe

Here's a list of the things we'll use in the creation of our reminder service:

- Free Twilio account – [sign up for one here](https://twilio.com/try-twilio) (<https://twilio.com/try-twilio>)
- SMS enabled phone number – [you can find the perfect one for your needs here](https://www.twilio.com/console/phone-numbers/search) (<https://www.twilio.com/console/phone-numbers/search>)
- [Microsoft Azure account](https://azure.microsoft.com/en-us/free/) (<https://azure.microsoft.com/en-us/free/>)
- [Visual Studio 2015](https://www.visualstudio.com/downloads/) (<https://www.visualstudio.com/downloads/>) (while it is possible to work on Azure Functions directly in the web portal, the developer experience is better in VS2015)
- [Microsoft Azure SDK for VS 2015](https://go.microsoft.com/fwlink/?LinkId=518003&clcid=0x409) (<https://go.microsoft.com/fwlink/?LinkId=518003&clcid=0x409>)
- [Visual Studio Tools for Azure Functions](https://aka.ms/azfunctiontools) (<https://aka.ms/azfunctiontools>)
- [Microsoft Azure Storage Explorer](http://storageexplorer.com/) (<http://storageexplorer.com/>)

Set up your accounts and install any of the software you don't have before moving on.

Structuring the Reminder Service

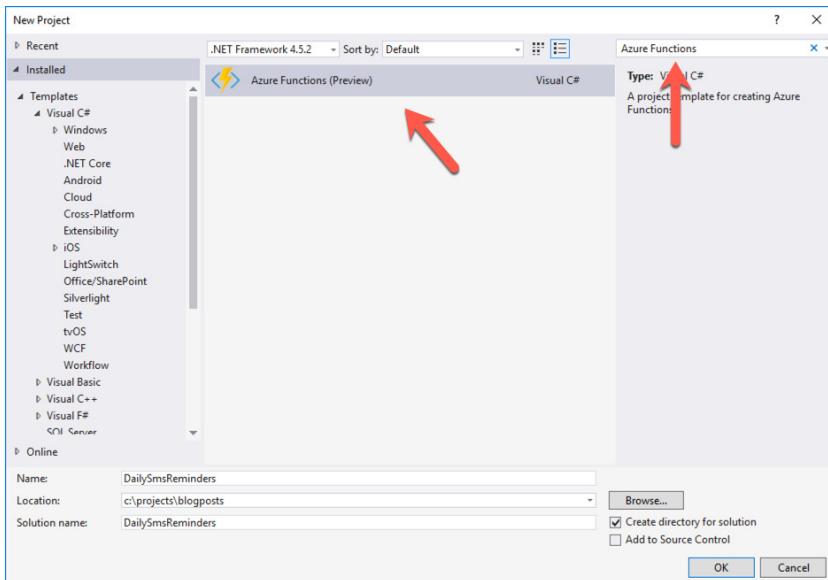
Azure Functions make it easy to quickly create the type of service we're building. It's "[serverless](https://en.wikipedia.org/wiki/Serverless_computing)" (https://en.wikipedia.org/wiki/Serverless_computing) which means we don't have to worry about web server architecture. Instead, we can just focus on the small bits of code or "functions" that make our service work.

We'll create one function that handles receiving SMS messages from Twilio. When it receives a request from Twilio it will create a record in an [Azure Table](https://azure.microsoft.com/en-us/services/storage/tables/) (<https://azure.microsoft.com/en-us/services/storage/tables/>) if the text message contains "subscribe". A second function will fire once a day to send an SMS reminder to each number in the Azure table.

Simple enough? Let's get started.

Handling Subscription Requests

Create a new Project in Visual Studio 2015 and search the installed templates for "Azure Functions":



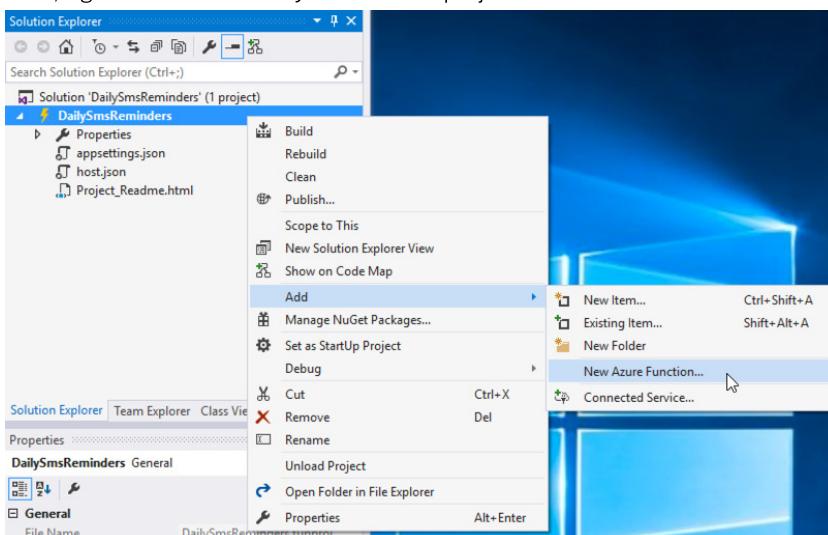
Name your project `DailySmsReminders` and click `OK`.

Before we add our functions, let's configure the app settings. Open `appsettings.json` and replace its contents with the following:

```
JavaScript
1 {
2     "IsEncrypted": false,
3     "Values": {
4         "AzureWebJobsStorage": "UseDevelopmentStorage=true",
5         "AzureWebJobsDashboard": "",
6         "TwilioAccountSid": "[Your Twilio Account Sid]",
7         "TwilioAuthToken": "[Your Twilio Auth Token]",
8         "TwilioPhoneNumber": "[Your Twilio Phone Number]",
9         "ReminderMessage": "Are you keeping up with your resolutions?"
10    }
11 }
```

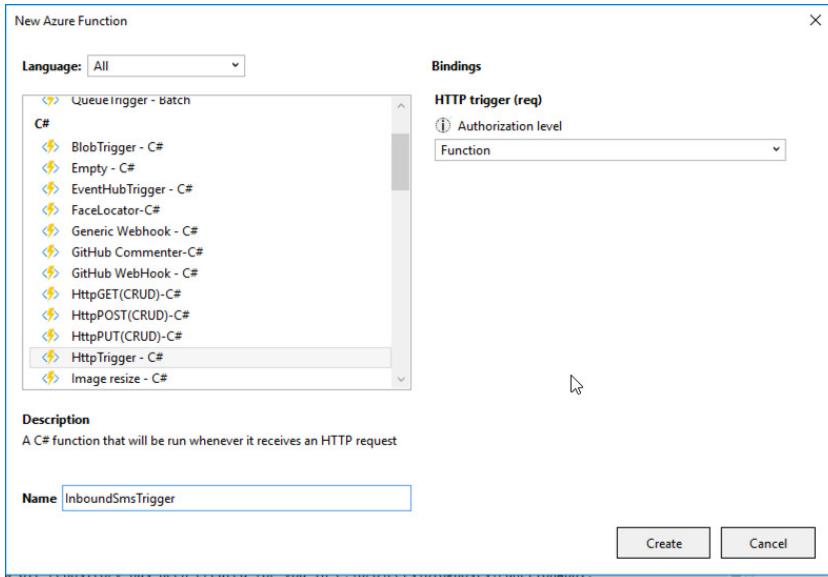
The `UseDevelopmentStorage=true` setting on line 4 makes it so that our project will use the [Azure Storage Emulator](#) (<https://docs.microsoft.com/en-us/azure/storage/storage-use-emulator>) any time we make a request to use Azure storage. Make sure to replace the Twilio credentials with your actual values.

Next, right-click on the `DailySmsReminders` project and select `Add->New Azure Function...`:



The dialog that pops up contains a lot of options. These represent the various ways an Azure function can be triggered. For instance, a function can be triggered when something is added to a queue using a `QueueTrigger`. Read more about what types of triggers are

available [here](https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview). Our first function will need to run when an HTTP request is made to it. The `HTTPTrigger - C#` option is perfect for this. Select this option, name the function `InboundSmsTrigger` and click `Create`:



Edit `InboundSmsTriggerproject.json` to add a dependency on `Twilio.Twiml` which we'll use to return TwiML to Twilio when a text message comes in:

```
1 {
2   "frameworks": {
3     "net46": {
4       "dependencies": {
5         "Twilio.Twiml": "3.5.0"
6       }
7     }
8   }
9 }
```

The `InboundSmsTriggerfunction.json` file is used to configure bindings for our function. The two existing bindings specified in the file correspond to the inbound HTTP request and the return from the function. We need to add a binding for the Azure Table we'll use to store our records. Edit `function.json` so that it looks like this:

```
1 {
2   "bindings": [
3     {
4       "authLevel": "function",
5       "name": "req",
6       "type": "httpTrigger",
7       "direction": "in"
8     },
9     {
10       "tableName": "Reminder",
11       "connection": "",
12       "name": "table",
13       "type": "table",
14       "direction": "in"
15     },
16     {
17       "name": "$return",
18       "type": "http",
19       "direction": "out"
20     }
21   ],
22   "disabled": false
23 }
```

In the highlighted lines we've added a table binding with a `tableName` of `Reminder` and set the connection for the table to the Azure Storage Emulator.

The function's code can be found in `InboundSmsTrigger.csx`. Delete what's there and replace it with the following code:

```

1 #r "Microsoft.WindowsAzure.Storage"
2 #r "System.Runtime"
3
4 using Microsoft.WindowsAzure.Storage;
5 using Microsoft.WindowsAzure.Storage.Table;
6 using System.Net;
7 using System.Text;
8 using Twilio.TwiML;
9
10 public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, CloudTable table, TraceWriter log)
11 {
12 }
13
14 public class Reminder : TableEntity
15 {
16 }
```

On lines 1-2 we pull in references for `Microsoft.WindowsAzure.Storage` and `System.Runtime`. These are special case external assemblies that can be referenced in this way. There's a full list [in the documentation](https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference-csharp) (<https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference-csharp>).

Line 10 adds a `CloudTable` `table` parameter to the function that corresponds to the table binding we added to `function.json`. This gives us access to the `Reminder` table in Azure Storage.

The `Reminder` class at the bottom is a `TableEntity` subclass that we will use to store records in Azure Table Storage. The class contains a `PartitionKey` which we'll always set to `Reminders` and a `RowKey` which we'll use for storing the user's phone number to send reminders to later.

The HTTP request from Twilio will contain form data in the body of the request that contains the parameters such as `Body` and `To` / `From` phone numbers. Let's add some code to parse those into a `Dictionary` and store the body of the message and the phone number the text message was sent from:

```

1 var data = await req.Content.ReadAsStringAsync();
2
3 var formValues = data.Split('&')
4     .Select(value => value.Split('='))
5     .ToDictionary(pair => System.Uri.UnescapeDataString(pair[0]),
6                  pair => System.Uri.UnescapeDataString(pair[1]));
7
8 var body = formValues["Body"];
9 var phoneNumber = formValues["From"];
10 var response = new TwilioResponse();
```

We also created a `TwilioResponse` object that will be used to return TwiML to Twilio.

If the user sends a message that contains anything other than `subscribe` we'll want to send back a message letting them know about the keyword. Let's set that up by adding the following code:

```

1 if (body.Trim('+').ToLower() == "subscribe")
2 {
3     // User is subscribing
4 }
5 else
6 {
7     response.Message("Welcome to Daily Updates. Text 'Subscribe' receive updates.");
8 }
```

When we parsed the `Body` value using `System.Uri.UnescapeDataString`. Any spaces in the string will be left as + signs. The highlighted line trims any + signs from the beginning and end of the `body` value, makes it lowercase, and compares it against `subscribe`.

The bulk of the work for the function occurs when the text message body is `subscribe`. Add this code to that block:

```
1 TableOperation operation = TableOperation.Retrieve<Reminder>("Reminders", phoneNumber);
2 TableResult result = table.Execute(operation);
3
4 if (result.Result != null)
5 {
6     response.Message("You already subscribed!");
7 }
8 else
9 {
10    response.Message("Thank you, you are now subscribed. Reply 'STOP' to stop receiving updates.");
11    var insertOperation = TableOperation.Insert(
12        new Reminder()
13        {
14            PartitionKey = "Reminders",
15            RowKey = phoneNumber
16        }
17    );
18
19    table.Execute(insertOperation);
20 }
```

Lines 1 and 2 look in the `Reminder` table for an entry that corresponds to the `PartitionKey` of `Reminders` and the `RowKey` of `phoneNumber`. If the result exists then we set the response message in line 6 to let the user know they're already subscribed. If they're not subscribed already then on line 10 we let the user know they are subscribed and the rest of the code in this block executes an insert operation to add their phone number to the table.

All that's left to do is return an `HttpResponse` from the function. We'll return the response TwiML as `application/xml` using `UTF8` encoding with the following code:

```
1 return new HttpResponseMessage  
2 {  
3     Content = new StringContent(response.ToString(), System.Text.Encoding.UTF8, "application/xml")  
4 };
```

Now it's time to test this function. First we need to start up the Azure Storage Emulator. Search in the Start menu for Azure Storage Emulator and run it. This will open a command prompt window that runs a command to start it:

```
C:\Program Files\Microsoft SDKs\Azure\Storage Emulator>AzureStorageEmulator.exe start  
Windows Azure Storage Emulator 4.6.0.0 command line tool  
The storage emulator was successfully started.
```

Once that's running you can build and run the Azure Functions solution. On first run, it will ask you to install the Azure CLI tools. Do this and then run the project again:



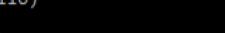
Just under the cool Azure Functions ASCII art you should see a `localhost` URL with a port number. You'll need to use something like [ngrok](https://ngrok.com/) (<https://ngrok.com/>) to forward this localhost port to the Internet so that Twilio can make an HTTP request to it. You can use this guide (<https://www.twilio.com/blog/2014/03/configure-windows-for-local-webhook-testing-using-ngrok.html>) if you're unsure.

how to set this up. A quick and easy way to get ngrok running from Visual Studio is [this very handy ngrok extension](#) (<https://marketplace.visualstudio.com/items?itemName=DavidProthero.NgrokExtensions>).

Head over to your [Twilio phone number](https://www.twilio.com/console/phone-numbers/incoming) (<https://www.twilio.com/console/phone-numbers/incoming>) and configure the A Message comes in webhook to point to [https://\[your ngrok url\]/api/InboundSmsTrigger](https://[your ngrok url]/api/InboundSmsTrigger) and click Save.

```
Command Prompt - c:\ngrok\ngrok http 7071 -host-header="localhost:7071"
ngrok by @inconshreveable

Session Status                online
Account                      Twilio (Plan: Twilio)
Version                       2.1.18
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://523b671f.ngrok.io -> localhost:7071
Forwarding                    https://523b671f.ngrok.io -> localhost:7071

Connections                  ttl     opn     rt1        rt5      p50      p90
                            0       0       0.00      0.00      0.00      0.00

ngrok url
```

Messaging

CONFIGURE WITH	Webhooks/TwiML	▼
A MESSAGE COMES IN	Webhook	▼
PRIMARY HANDLER FAILS	Webhook	▼

Test the inbound trigger by sending SMS messages. Try sending something that's not `subscribe` then send `subscribe` and finally try subscribing again. Now that we have subscriptions, let's send the reminders.

Don't Forget, Send Daily SMS Reminders!

Add another Azure Function to the project. This time select `TimerTrigger - c#` and name the function `SendsSmsReminders`. A timer trigger will fire according to a schedule determined by a `cron` expression in the `function.json` file. The `cron` expression has six fields: `{second} {minute} {hour} {day} {month} {day of the week}`. For example, a `cron` expression that will trigger at 10:00:00 AM in the Eastern US time zone every day would look like this: `0 0 10 * * *`. Modify `SendSmsRemindersfunction.json` to contain this `cron` expression and also add a table binding like we did in the previous function:

JavaScript

```
1  {
2    "bindings": [
3      {
4        "name": "myTimer",
5        "type": "timerTrigger",
6        "direction": "in",
7        "schedule": "0 0 15 * * *"
8      },
9      {
10        "tableName": "Reminder",
11        "connection": "",
12        "name": "table",
13        "type": "table",
14        "direction": "in"
15      }
16    ],
17    "disabled": false
18 }
```

Next, add a Twilio dependency to `SendSmsRemindersproject.json`:

<https://www.twilio.com/blog/2017/01/how-to-send-daily-sms-reminders-using-c-azure-functions-and-twilio.html>

```

1 {
2   "frameworks": {
3     "net46": {
4       "dependencies": {
5         "Twilio": "4.7.2"
6       }
7     }
8   }
9 }
```

Now we're ready to send the reminders. Open up `sendsmsRemindersrun.csx` and replace the contents with this code which references the Twilio and Azure Storage assemblies and adds the CloudTable binding to the parameter list:

```

C#
1 #r "Microsoft.WindowsAzure.Storage"
2
3 using Microsoft.WindowsAzure.Storage;
4 using Microsoft.WindowsAzure.Storage.Table;
5 using System;
6 using Twilio;
7
8 public static void Run(TimerInfo myTimer, CloudTable table, TraceWriter log)
9 {
10 }
11
12 public class Reminder : TableEntity
13 {
14 }
15 }
```

Inside the `Run` function we'll start by setting up a `TwilioRestClient`:

```

C#
1 var accountSid = Environment.GetEnvironmentVariable("TwilioAccountSid");
2 var authToken = Environment.GetEnvironmentVariable("TwilioAuthToken");
3 var twilio = new TwilioRestClient(accountSid, authToken);
```

Next, construct a query over the `Reminder` table that matches the `Reminders` `PartitionKey`. This will return all of the users that have subscribed:

```

C#
1 var query = new TableQuery<Reminder>().Where(
2   TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "Reminders")
3 );
```

Now that we have all of the entries, loop through each one and send the SMS reminder using the `TwilioRestClient`:

```

C#
1 foreach (Reminder reminder in table.ExecuteQuery(query))
2 {
3   var message = twilio.SendMessage(
4     Environment.GetEnvironmentVariable("TwilioPhoneNumber"),
5     reminder.RowKey,
6     Environment.GetEnvironmentVariable("ReminderMessage")
7   );
8 }
```

To test this out you'll want to change the cron expression in `function.json` to a time a few minutes in the future and then run the project. You should receive an SMS containing the `ReminderMessage` when the cron expression evaluates to the time you set. Now you won't lose track of those resolutions!

What's Next?

Now that you know how to use C#, Azure Functions and Twilio to send daily SMS reminders, try some of these things as next steps:

- Try deploying the project to Azure. The process is described in [this step-by-step guide \(https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function-azure-portal\)](https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function-azure-portal).
- Modify the service so that the user can specify a custom reminder.
- Learn how to set up appointment reminders (<https://www.twilio.com/docs/tutorials/walkthrough/appointment-reminders/csharp/mvc>) in this tutorial (<https://www.twilio.com/docs/tutorials/walkthrough/appointment-reminders/csharp/mvc>)

I'm really excited to see what you come up with using these technologies. Let me know what you build by hitting me up on Twitter [@brentschooley](https://twitter.com/brentschooley?lang=en) (<https://twitter.com/brentschooley?lang=en>).

[Like](#) [Tweet](#) [Follow @twilio](#)  by Brent Schooley (<https://www.twilio.com/blog/author/brent-schooley>) brent@twilio.com
[\(mailto:brent@twilio.com\)](mailto:mailto:brent@twilio.com)

Use Twilio Tutorials to Start Building

Call Tracking

(<https://www.twilio.com/docs/tutorials/walkthrough/call-tracking/php/laravel>)

SMS and MMS Notifications

(<https://www.twilio.com/docs/tutorials/walkthrough/server-notifications/php/laravel>)

Two-Factor Authentication with Authy

(<https://www.twilio.com/docs/tutorials/walkthrough/two-factor-authentication/php/laravel>)

0 Comments [Twilio Blog](#)

 [Login](#) ▾

 [Recommend](#)  [Share](#)

[Sort by Oldest](#) ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

 [Subscribe](#)  [Add Disqus to your site](#) [Add Disqus](#)  [Privacy](#)

Search ...

Power modern communications. Build the next generation of voice and SMS applications.

[Start Building For Free \(<http://twilio.com/try-twilio>\)](#)

Posts By Stack

[.NET \(<https://www.twilio.com/blog/tag/net>\)](#)

[Arduino \(<https://www.twilio.com/blog/tag/arduino>\)](#)

[Java \(<https://www.twilio.com/blog/tag/java>\)](#)

[JavaScript \(<https://www.twilio.com/blog/tag/javascript>\)](#)