

## Introduction

Thank you for your interest in the twiSQL project. twiSQL is a set of purpose built CLR assemblies that will enable your SQL server to place telephone calls and send text messages. Harnessing the power of web based API calls to Twilio.com this library will allow you to create feature rich outbound messaging applications with zero infrastructure costs and minimal transactional costs.

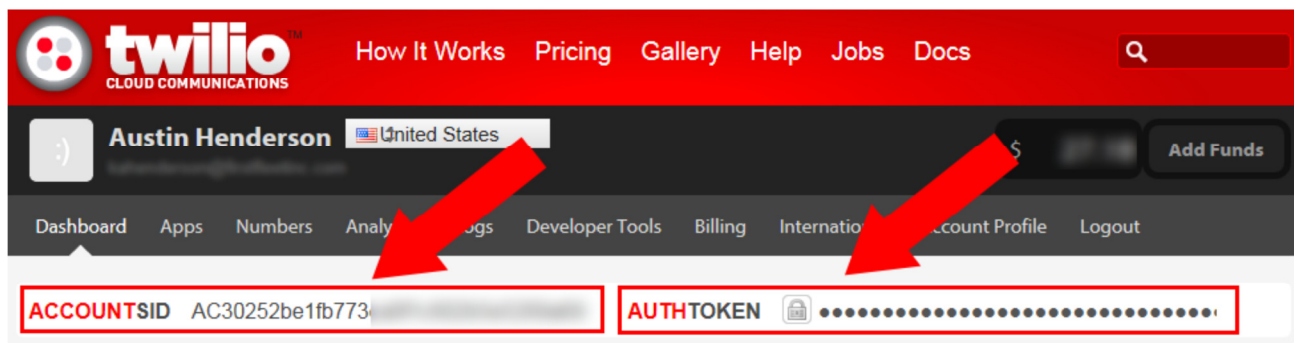
## Is This Project Free?

Yes, the project can be downloaded freely from austinHenderson.com, and it will work with all functionality without contribution to our cause. However, as a token of my appreciation those that contribute \$20 USD to our cause will get access to these features without the rate limiting function (five second delay on function response) that is enforced on the project as downloaded from the site. It is my sincere hope that this method of distribution gives credibility to the project and encourages those that feel motivated to support our cause.

## How Does it Work?

twiSQL is a DLL designed to be installed and used on a SQL server through CLR functionality and is enabled only by the services of Twilio.com. Twilio.com has built a rock solid API that is the real brains of this project which allows you to place calls or text messages for a very low transactional cost. This project will net you a set of user defined functions that you can nest directly in TSQL code to enable this exciting functionality. If you are not familiar with Twilio's services you really need to start this discovery at [www.twilio.com](http://www.twilio.com). You don't need to understand the API or be a .Net programmer, but you do need to have an **active** and **funded** Twilio.com account. Once you have your account look for your Account SID and the AuthToken values on the site – these are like username and password values.

The current view of the Twilio.com site (once logged in) looks like the screenshot below. Notice there is an **ACCOUNTSID** and an **AUTHTOKEN** section below. For our purposes think of these values as a username and password, you need these but these are your keys to accessing the API – these are private values – do not distribute them with your application.



## Installation

The process of installing the library on your SQL server is straightforward and will take less than five minutes to complete. Here is a basic outline of how to get the coder working in your environment.

- Copy the DLLs to the machine in a local path that is accessible by the SQL server

- Enable CLR on the SQL server to allow extension to function
- Set the trust mode of the database to which you will be adding the user defined functions
- Create the assembly reference on the SQL server
- Create the UDF in the database on the SQL server

## Copy the DLLs

Extract the content of the zip archive to C:\ah.Solutions\twilioSQL\. This path actually is insignificant as long as you make record of where you are planning to install the product. It is important that the credentials the SQL server is running under have access to this path.

## Enable CLR

Enabling CLR on the SQL server allows the SQL server to run .Net code. In this case we need to enable this feature because we are asking SQL server to make use of a dll that is a .Net dll. To enable CLR on the SQL server please run the following command.

```
EXEC sp_configure 'clr enabled', 1;  
RECONFIGURE WITH OVERRIDE;  
GO
```

## Trust Mode

Because we are installing a custom library that will be accessing network resources we must indicate that the database where the UDF will be installed is running in TRUSTWORTHY mode. To complete this task run the following code [replacing the word YourDatabase](#) with the name of the database you desire to alter. If this option scares you create a new database, perhaps called ah.Solutions, for the purpose of creating these functions. Where the functions are installed isn't particularly important.

```
ALTER DATABASE YourDatabase SET TRUSTWORTHY ON
```

## Create Assembly Reference

The user defined functions we will be creating require access to the assembly and thus we must tell SQL server about the assembly before we can define the functions. To add the references run the following statement.

```
CREATE ASSEMBLY twilioSQL from 'C:\ah.Solutions\twilioSQL\twiSQL.dll'  
with permission_set = UNSAFE  
go
```

## Create User Defined Functions

The user defined functions are the extensions that make this process possible. These functions will be called in your TSQL code to process the requests. To add the user defined functions run the following statements.

```
Create Function msgTwilioSendText  
(  
@tonumber nvarchar(max),  
@fromnumber nvarchar(max),  
@txtbody nvarchar(max),  
@accountSID nvarchar(max),  
@authToken nvarchar(max)
```

```
)  
Returns nvarchar(max)  
As External Name twilioSQL.[twilioSQL.twilioSQL].msgTwilioSendText  
Go  
Create Function msgTwilioMakeCall  
(  
@tonumber nvarchar(max),  
@fromnumber nvarchar(max),  
@txtbody nvarchar(max),  
@gender nvarchar(max),  
@accountSID nvarchar(max),  
@authToken nvarchar(max)  
)  
Returns nvarchar(max)  
As External Name twilioSQL.[twilioSQL.twilioSQL].msgTwilioMakeCall  
Go  
Create Function msgTwilioViewCall  
(  
@callsid nvarchar(max),  
@accountSID nvarchar(max),  
@authToken nvarchar(max)  
)  
Returns nvarchar(max)  
As External Name twilioSQL.[twilioSQL.twilioSQL].msgTwilioViewCall  
Go  
Create Function msgTwilioViewSMSMessage  
(  
@messagesid nvarchar(max),  
@accountSID nvarchar(max),  
@authToken nvarchar(max)  
)  
Returns nvarchar(max)  
As External Name twilioSQL.[twilioSQL.twilioSQL].msgTwilioViewSMSMessage  
Go
```

## Installation Summary

The installation of this library on your SQL server should now be complete. Because SQL server has so many configuration options it is difficult to describe every possible problem you may run into, but if you run into problems along the way please contact us at [ahSolutions.Help@gmail.com](mailto:ahSolutions.Help@gmail.com) and we will try to help resolve any issues that may arise.

## UDF Definitions

### msgSendText()

This UDF is built to simply send a text message to any recipient valid as a destination for SMS messages through the Twilio interface.

#### Example Call:

```
DECLARE @testResult varchar(max)
SELECT @testResult = dbo.msgTwilioSendText('11 Digit Recipient', '11 Digit
Sender', 'This is a test.', 'ACCOUNT SID', 'AUTH TOKEN')
SELECT @testResult
```

**toNumber** – Valid 11 digit number for receiving SMS messages.

**fromNumber** – 11 digit number assigned to your Twilio.com account.

**sendText** – The body of the text message you wish to send.

**accountSID** – The Account SID for your Twilio.com API call.

**authToken** – The Auth Token value for your Twilio.com API call.

**Return** – The return content is HTTP RESPONSE CODE|MessageSID or Returned XML

- If the text message was successful in submission for delivery the response will be **201 | MessageSID**
  - The MessageSID value should be stored if you desire to check delivery status later.
- If the return does not begin with 201 the submission for delivery failed and you will need to parse the XML contained after the | to better understand the specific error.

### msgTwilioMakeCall()

This UDF will place a telephone call and use text to speech to deliver a message or it can play an MP3 file that is publically accessible using HTTP(s).

**toNumber** – Valid 11 digit number for receiving SMS messages.

**fromNumber** – 11 digit number assigned to your Twilio.com account.

**txtBody** – This is the text you wish to speak to the recipient.

**gender** – Specify male or femal gender to read the text (m/f).

**accountSID** – The Account SID for your Twilio.com API call.

**authToken** – The Auth Token value for your Twilio.com API call.

**Return** – The return content is HTTP RESPONSE CODE|CallSID or Returned XML

- If the text message was successful in submission for delivery the response will be **201 | CallSID**
  - The CallSID value should be stored if you desire to check details / status of the call later.
- If the return does not begin with 201 the submission of the call request failed and you will need to parse the XML contained after the | to better understand the specific error.

#### Example Call Text to Speech:

```
DECLARE @testResult varchar(max)
SELECT @testResult = dbo.msgTwilioMakeCall(('11 Digit Recipient', '11 Digit
Caller', 'This is a test.', 'f', 'ACCOUNT SID', 'AUTH TOKEN')
SELECT @testResult
```

## Example Call Play MP3:

```
DECLARE @testResult varchar(max)
SELECT @testResult = dbo.msgTwilioMakeCall(('11 Digit Recipient', '11 Digit Caller', 'http://site.com/file.mp3', '', 'ACCOUNT SID', 'AUTH TOKEN'))
SELECT @testResult
```

## msgTwilioViewCall()

This UDF is built to check the status of a previously queued telephone call by unique callSID value.

## Example Call:

```
DECLARE @testResult varchar(max)
SELECT @testResult = dbo.msgTwilioViewCall('CALLSID', 'ACCOUNT SID', 'AUTH TOKEN')
SELECT @testResult
```

**callSID** – The callSID value returned from a previous call to msgTwilioMakeCall.

**accountSID** – The Account SID for your Twilio.com API call.

**authToken** – The Auth Token value for your Twilio.com API call.

**Return** – The return content is HTTP RESPONSE CODE|Status|Returned XML Content

- If the call to the API was successful in the response will be **201|Status|Returned XML Content**
  - In the condition that the returned content begins with 201 the next position in the returned content (after the |) will be the status of the call.
- If the return does not begin with 201 the request to the API was not successful and you will need to parse the XML returned content to better understand the specific error.

## msgTwilioViewSMSMessage()

This UDF is built to check the status of a previously queued text message by unique messageSID value.

## Example Call:

```
DECLARE @testResult varchar(max)
SELECT @testResult = dbo.msgTwilioViewCall('MESSAGESID', 'ACCOUNT SID', 'AUTH TOKEN')
SELECT @testResult
```

**messageSID** – The messageSID value returned from a previous call to msgTwilioSendText.

**accountSID** – The Account SID for your Twilio.com API call.

**authToken** – The Auth Token value for your Twilio.com API call.

**Return** – The return content is HTTP RESPONSE CODE|Status|Returned XML Content

- If the call to the API was successful in the response will be **201|Status|Returned XML Content**
  - In the condition that the returned content begins with 201 the next position in the returned content (after the |) will be the status of the call.
- If the return does not begin with 201 the request to the API was not successful and you will need to parse the XML returned content to better understand the specific error.

## Uninstallation

Should you desire to remove this product from your SQL server please issue the following commands against the database where you installed the assemblies.

```
drop function msgTwilioSendText
drop function msgTwilioMakeCall
drop function msgTwilioViewCall
drop function msgTwilioViewSMSMessage
drop assembly twilioSQL
```

## Conclusion

This library is designed to extend the functionality of SQL server through the use of generic UDFs that can be integrated into business processes with relative ease. The examples provided in this document are for illustration purposes only, and they are by no means exhaustive. If you have ideas or suggestions for the future development of this product or have questions for us we would love to hear from you at [ahSolutions.Help@gmail.com](mailto:ahSolutions.Help@gmail.com).

## Sample Project

Sometimes the best way to understand an idea is to see it in action in the form of a working project. In this sample project we will illustrate the idea of a list of prospects that need to be contacted with a telephone campaign.

First we build some tables to hold the details of these transactions.

```
CREATE TABLE [dbo].[myProspects] (
    [ProspectID] [int] IDENTITY(1,1) NOT NULL,
    [ProspectName] [varchar](max) NULL,
    [ProspectPhone] [varchar](25) NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[myProspectCalls] (
    [CallID] [int] IDENTITY(1,1) NOT NULL,
    [ProspectID] [int] NULL,
    [CallDate] [datetime] NULL,
    [CallSID] [varchar](255) NULL,
    [CallStatus] [int] NULL,
    [CallDetails] [varchar](max) NULL
) ON [PRIMARY]
```

In the example code below we will insert a few records into our myProspects table and then loop through those records placing telephone calls and playing the predefined greeting.

```
-- CLEANING OUT THE TABLE FOR PROSPECTS - THIS IS JUST FOR TEST RUN
truncate table myProspects
-- BUILDING THE PROSPECT RECORDS - NORMALLY THIS WOULD BE LOADED FROM A FILE
insert into myProspects (ProspectName, ProspectPhone)
select 'John Doe', '18005551212'

insert into myProspects (ProspectName, ProspectPhone)
select 'Bill Smith', '18005551212'
```

```
-- THIS SECTION HANDLES PLACING THE CALLS
declare @prospectID int
declare @callingPhoneNumber varchar(11)
declare @prospectPhone varchar(25)
declare @prospectName varchar(255)
declare @CallSID varchar(255)
declare @authID varchar(255)
declare @token varchar(255)
declare @whatToSay varchar(max)
declare @whatToSayTemplate varchar(max)

-- HERE YOU WOULD USE YOUR CUSTOM MESSAGE TO SPEAK
select @whatToSayTemplate = 'Hello %NAME%. As a representative of We Make Stuff
company I would like to thank you for reaching out to our agent for your needs. If
you should need to reach us at any time please feel free to call 888-555-1212 for
prompt assistance. We sincerly appreciate your time, and please remember to look us
up on Facebook. Have a fantastic day!'
-- YOU WILL NEED TO FILL IN YOUR AUTHID AND TOKEN VALUES FROM TWILIO
select @authID = 'AC30252be1fb773ca8f1c602b5e12dr56'
select @token = '3601418be2da090f3701d50890234r56'
-- YOU HAVE TO HAVE A PHONE NUMBER WITH TWILIO - REPLACE THE NUMBER BELOW
select @callingPhoneNumber = '16158008999'
-- CHECKING TO GET THE FIRST PROSPECT
select @prospectID =
(select top 1 ProspectID from myProspects order by ProspectID asc)

-- WE HAVE CALLS TO MAKE HERE PEOPLE - LETS GET TO WORK
while isnull(@prospectID,0) > 0
begin
    -- GETTING THE VARIABLES TO USE FOR OUR CALL
    select @prospectPhone = ProspectPhone, @prospectName = prospectName
    from myProspects where ProspectID = @prospectID
    -- NUMBER MUST BE 11 DIGITS
    if isnumeric(@prospectPhone) = 1 and len(@prospectPhone) = 11
    begin
        select @prospectPhone
        -- REPLACING THE NAME IN OUR STRING TO SAY - NICE TOUCH
        select @whatToSay = replace(@whatToSayTemplate, '%NAME%', @prospectName)

        -- HERE WE MAKE THE CALL THROUGH OUR UDF
        SELECT @CallSID =
        dbo.msgTwilioMakeCall(@prospectPhone, @callingPhoneNumber, @whatToSay, 'm', @authID, @to
        ken)

        if charindex('|', @CallSID) > 0
        begin
            SELECT @CallSID =
            right(@CallSID, len(@CallSID) - charindex('|', @CallSID))
            insert myProspectCalls (ProspectID, CallDate, CallSID, CallStatus)
            values (@prospectID, getdate(), @CallSID, 0)

        end
        else
        begin
            insert myProspectCalls (ProspectID, CallDate, CallSID, CallStatus)
            values (@prospectID, getdate(), @CallSID, -1)

        end
    end
end
```

```

        -- GETTING THE NEXT PROSPECT
        select @prospectID = (select top 1 prospectID from myProspects
        where prospectID > @prospectID order by prospectID asc)
    end
end

```

Now that we have placed the telephone calls at some point in the future you may want to know that the call was actually completed. The code below is used to illustrate how you can use the provided functions to check the details of a specific call.

```

-- THIS LOGIC WILL CHECK THE STATUS OF THE CALLS
declare @authID varchar(255)
declare @token varchar(255)
-- YOU NEED TO REPLACE YOUR AUTHID AND TOKEN
select @authID = 'AC30252be1fb773ca8f1c602b5e12dr56'
select @token = '3601418be2da090f3701d50890234r56'
declare @callSID varchar(255)
declare @callID int
declare @callDetails varchar(max)
declare @callStatus int

-- WE WILL BE SETTING UP A LOOP HERE TO GO THROUGH THE RECORDS
select @callID =
(select top 1 CallID from myProspectCalls where CallStatus = 0 order by callID asc)

while isnull(@callID,0) > 0
begin
    select @callSID = callSID from myProspectCalls where callID = @callID
    select @callDetails = dbo.msgTwilioViewCall(@callSID,@authID,@token)

    -- THE FULL XML IS RETURNED IN THE @callDetails VARIABLE
    -- YOU CAN PROCESS THAT XML HOWEVER YOU NEED AS THERE IS GOOD
    -- INFO CONTAINED IN IT - FOR THIS CASE WE JUST CHECK TO SEE THE STATUS
    if charindex('|completed|',@callDetails) > 0
    begin
        select @callStatus = 1
    end
    else
    begin
        select @callStatus = 0
    end

    -- UPDATING THE STATUS OF THE CALL RECORD
    update myProspectCalls
    set callDetails = @callDetails, callStatus = @callStatus
    where callID = @callID

    -- GETTING THE NEXT CALL ID RECORD
    select @callID =
    (select top 1 CallID from myProspectCalls
    where CallStatus = 0 and callID > @callID
    order by callID asc)
end

```

One other option to consider is that as opposed to using the text to speech engine you can alternatively record an mp3 file, host it on a public URL and pass the URL to the file as the text of the call. This option provides for a very personalized campaign with a small modification to the process.

When we make the call to msgTwilioMakeCall() we just replace the variable @whatToSay with the URL to the file. A simple way to pull this off will be to host a file using the shared link functionality of Dropbox – or any other public URL. In the example below we are using Dropbox.

```
SELECT @CallSID = dbo.msgTwilioMakeCall(@prospectPhone,@callingPhoneNumber,  
    'http://dl.dropbox.com/u/1541272/recorded.mp3', 'm', @authID, @token)
```

Using the above method you could easily pull of the familiar telephone call campaign component of running for public office or perhaps the reminder of your dentist appointment, and all for almost nothing in expense.