Send a SMS Message from Apex

Prerequisites

The developer will need proficiency in:

- Salesforce.com Object model
- Apex Programming

The APIs' can be worked on via this link

Apex is a strongly-typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Salesforce Lightning Platform server, together with calls to the API.

Object & Fields Information

There is a custom object in the SMSMagic Interact Managed package known as SMS History, and the corresponding API name is smagicinteract__smsMagic__c. This object stores SMS message data. Considering the need for complex customization for implementing various business workflows, we have provided a simple way to send SMS from Apex code.

The following table contains fields that must be populated to successfully send messages from Apex:

Name	Field API name	Purpose	Required
SenderID	smagicinteract__SenderId__c	Phonenumber or business identity of your business	Yes
Mobile Number	smagicinteract__PhoneNumber__c	Phone number of Contact/Lead to whom you are sending the message.	Yes
Name	smagicinteract__Name__c	Name of person to whom you are sending this message.	No
SMSText	smagicinteract__SMSText__c	Content of message	Yes
Disable SMS on Trigger	smagicinteract__ disableSMSOnTrigger__c	This is used to control triggers. If set 1, the trigger is deactivated, so the message won't be sent, only a record will be created. If set to 0, a message will be sent. It should be 0 by default.	Yes

External Field	smagicinteract__external_field__c	This is indexed and unique field used as a reference to update delivery reports.	Yes
Object Type	smagicinteract__ObjectType__c	Identifier of the object from which the message will be sent	No
Direction	smagicinteract__Direction__c	With 1.49 onwards, this new field is used to indicate whether it's an outgoing or incoming message. Set its value as "OUT" for sending messages.	Yes

Send an SMS message from Apex code

The developer would first create an instance of the **SMS History** object, populate all required fields, and then insert the instance of that object using a database insert. SMS-Magic provides a custom trigger that will (a) execute after the insertion of the record and (b) send out messages to the **Mobile Number**. The trigger will also populate other fields in the **SMS History** object instance with default values.

This sample code sends SMS messages. Feel free to copy it and modify it according to your environment.

```
List  smsObjectList = new List ();
String senderId = 'smsMagic'; // Please replace the 'smsMagic' with your relevant sender ID.
String templateText = 'test SMS by Screen Magic'; // you can fetch the template text by querying the record on smagicinteract__SMS_Template__c object
smagicinteract__smsMagic__c smsObject = new smagicinteract__smsMagic__c();
smsObject.smagicinteract__SenderId__c = senderId;
smsObject.smagicinteract__PhoneNumber__c = contact.MobilePhone;
smsObject.smagicinteract__Name__c = contact.Name; // records name
smsObject.smagicinteract__ObjectType__c = 'Contact'; // record type
smsObject.smagicinteract__disableSMSOnTrigger__c = 0; // this field either be 0 or 1, if you specify the value as 1 then sms will not get send but entry of sms will get create under SMS History object
smsObject.smagicinteract__external_field__c = smagicinteract.ApexAPI.generateUniqueKey();
smsObject.smagicinteract__SMSText__c = templateText;
smsObjectList.add(smsObject);
Database.insert(smsObjectList, false);
```

Troubleshooting

If you encounter any problems, consider the following:

- Ensure that your code is not invoked from a scheduled method of any

other trigger.
- A user on whose behalf this code is executed must have permission to use **SMS History** objects.

Schedule SMS Messages

With the Salesforce.com scheduler, you can schedule a SMS job for a particular day and time. We can accomplish this by employing the **Schedulable** Apex interface.

The **Schedulable** interface contains one method that must be implemented, the method is **Execute**.

```
global void execute(SchedulableContext sc){}
```

In this method you can write your business logic, and then follow the steps given below and schedule the Apex class.

Step 1 – Goto to Setup –> App Setup –> Develop –> Apex classes.

Step 2 – Click the Schedule Apex button, complete the form, and save

In the figure above, we can see that the class is ready to execute. This class is set to run each day and send SMS messages according to the business logic.

Step 3 – Choose the type of scheduling

Prepare SMS at time of schedule

To schedule an SMS message to be sent to a group of numbers, first create the instance of **smagicinteract__Scheduled_SMS__c** object and attach the required fields such as **smagicinteract__MobilePhone__c**, **smagicinteract__status__c**, **smagicinteract__Scheduled_Date__c**, **smagicinteract__SMSText__c**. Any records that are scheduled will be picked at the time of distribution. Note that smagicinteract above is a package prefix which will differ from package to package.

The advantage of this type is that you can cancel the schedule by simply deleting the records. The following sample code demonstrates this more clearly.

```
List conList = [select Id,FirstName, LastName, MobilePhone, Name from
Contact];
Date scheduleDate = Date.valueOf('2011-08-10');
List scheduleSMSList = new
List();
if(conList != null){
for(Contact contact : conList){
smagicinteract__Scheduled_SMS__c scheduleSMSObject = new
smagicinteract__Scheduled_SMS__c();
if(contact.MobilePhone != null){
scheduleSMSObject.smagicinteract__MobilePhone__c = contact.MobilePhone;
scheduleSMSObject.smagicinteract__jobId__c = '1';
scheduleSMSObject.smagicinteract__status__c = 'Schedule';
scheduleSMSObject.smagicinteract__Scheduled_Date__c = scheduleDate;
scheduleSMSObject.smagicinteract__SMSText__c = 'Test Of Schedule';
}
scheduleSMSList.add(scheduleSMSObject);
}
insert scheduleSMSList;
}
```

After this, use a **Schedulable** class to pickup the **scheduled_SMS__c** object record—according to **scheduleDate** and send the SMS messages by creating the **SMS_Magic__c** object—or us **SMSPushAPI**.

Message rendered at schedule execution

With this approach, you simply schedule on a particular object. We do not create the object of **smagicinteract__Scheduled_SMS__c**, but only schedule on a particular object. You can write one Schedulable class that will pick up particular object records according to the timestamps of each. Render SMS drafts using the template created for that object and the SMS messages will be sent. To cancel a scheduled SMS distribution, you'll need to delete the scheduled job itself. The following sample code will schedule SMS messages from a custom object named **Time sheet**.

```
global class TodaysScheduleSMS implements Schedulable {
String day = '';
String query = '';
List todaysScheduleList = null;
String smsText = '';
String status = '';
String week = '';
global void execute(SchedulableContext sc){
// Here we can get today's day.
day = DateTime.now().format('EEEEE');
todaysScheduleList = [select SMS_Template__c, Status__c, Day__c, Week__c from
Time_Sheet_Schedule__c where Day__c =:day ];
if(todaysScheduleList.size() > 0){
for(Time_Sheet_Schedule__c timeSheetScheduleObj : todaysScheduleList){
status = timeSheetScheduleObj.Status__c;
```

```
week = timeSheetScheduleObj.Week__c;
List smsTemplateList = [select
smagicinteract__Text__c from smagicinteract__SMS_Template__c where Id
=:timeSheetScheduleObj.SMS_Template__c];
if(smsTemplateList != null){
smsText = smsTemplateList[0].smagicinteract__Text__c;
}
}
/* write here code to send SMS to number */
}
}
}
```

There are limits on how many callouts and DML statements you can execute from a scheduled job. Only 10-20 SMS messages can be sent in one job. There is a workaround to exceed this limit, in which you can use a batchable class – as outlined below.

```
global class SendScheduleSMSBatch implements Database.Batchable,
Database.AllowsCallouts {

global SendScheduleSMSBatch(){
}
global Database.QueryLocator start(Database.BatchableContext BC){
String day = DateTime.now().format('EEEEE');
String scheduleStatus = 'Schedule';
String query = 'select SMS_Template__c, Status__c, Day__c, Week__c from
Time_Sheet_Schedule__c where Day__c =:day';
return Database.getQueryLocator(query);
}

global void execute(Database.BatchableContext BC, List scope){
if(scope.size() > 0){
for(Time_Sheet_Schedule__c timeSheetScheduleObj : todaysScheduleList){
status = timeSheetScheduleObj.Status__c;
week = timeSheetScheduleObj.Week__c;
List smsTemplateList = [select
smagicinteract__Text__c from smagicinteract__SMS_Template__c where Id
=:timeSheetScheduleObj.SMS_Template__c];
if(smsTemplateList != null){
smsText = smsTemplateList[0].smagicinteract__Text__c;
}
}
/* write here code to send SMS to number */
}
global void finish(Database.BatchableContext BC){
// send batch execution email;
}
}
```

From the Schedulable class, you need to call a batch class:

```
SendScheduleSMSBatch batchSMS = new SendScheduleSMSBatch();
Database.executeBatch(batchSMS);
```

An example test class for this batch class is given below.

```
@isTest
private class SendScheduleSMSBatchTest {

static Time_Sheet_Schedule__c timeSheetSchedule = null;
static smagicinteract__SMS_Template__c smsTpl= null;
static void setupTest(){

smsTpl = new smagicinteract__SMS_Template__c();
smsTpl.smagicinteract__Text__c = 'Test of Schedule SMS';
smsTpl.smagicinteract__Name__c = 'Schedule SMS Template'
smsTpl.smagicinteract__ObjectName__c = 'Time_Sheet_Schedule__c';
insert smsTpl;
timeSheetSchedule = new Time_Sheet_Schedule__c();
timeSheetSchedule.SMS_Template__c= smsTpl.Id;
timeSheetSchedule.Status__c = 'Scheduled';
timeSheetSchedule.Day__c = 'Monday';
timeSheetSchedule.Week__c = 'this';
insert timeSheetSchedule;
}
static void testMethod test_ScheduleSMSBatch(){
Test.StartTest();
setupTest();
SendScheduleSMSBatch sscb = new SendScheduleSMSBatch();
ID batchprocessid = Database.executeBatch(sscb);
Test.stopTest();
List timeSheetScheduleList = [select Id, Status__c
from Time_Sheet_Schedule__c where Id =:timeSheetSchedule.Id];
system.assertEquals(timeSheetScheduleList[0].Status__c, 'Sent');
}
}
```

Push to URL Configuration

Configure Incoming & Delivery Push URL from customer portal and for individual accounts:

Incoming Number Configuration from Customer Portal:

Follow the given procedure to assign Incoming number to your account from the customer portal:

1. Log in to the customer portal.

2. On the top menu bar click SMS Services.

3. In the navigation pane on the right, Under Incoming Numbers, click My Number.
4. On the My Number page that appears, select the country.
5. Click Details. The Incoming Number pop up appears.

6. Under Incoming Configuration, click Add Push URL. A blank text area appears.
7. Insert any URL to which you want to push your incoming text.

   (e.g. http://api.webhookinbox.com/i/fcL3J7dy/in/)

8. Click Save. All Incoming SMS received on the Portal, is pushed to the assigned URL.

9. The Push to URL status is shown under IncomingSMSHistory(Response) on Customer Portal as shown in screen below.

10. A sample Push Incoming SMS Request Payload is shown below:

Delivery Configuration of Incoming Number for individual account:

We can set Delivery report URL for Incoming Number per Account.

Incoming Number:

1. Assign Incoming number to your account from the admin portal.
2. Log in to the customer portal.
3. On the top menu bar, click SMS Services.
4. In the navigation pane on the right, under Incoming Numbers, click My Number.
5. On the My Number page that appears, select the country.
6. Click Details. The Incoming Number pop up appears.

7. Under Delivery Configuration, click Add Push URL. The blank text area appears.

8. Insert any URL to which you want to get the Delivery report. (e.g. http://api.webhookinbox.com/i/fcL3J7dy/in/)
   All Delivery reports for outgoing messages on the Portal, are pushed to assigned URL.

   A sample Push Delivery Report SMS Request Payload :

Refer to the API Doc URL for more details on the functioning of the Incoming SMS Push URL.

Resolving a Template with Merge Fields

This article explains how to render a template with merge fields, using Apex.

Prerequisites

The developer will need proficiency in:

- Salesforce.com Object model
- Apex programming

What are templates?

SMS-Magic is a native application that integrates with Salesforce.com, which is primarily used to store lead and contact information. SMS communication is all about 1-to-1 communication, and many customers will give a prompt response if the message is personalized.

Merge-field templates can be used by sales or marketing teams to send personalized messages. This article explains how to do this with Salesforce.com Apex. It is very simple to implement, so that you can maintain focus on your business logic while SMS-Magic handles the SMS messaging.

Class & Method Details

Class: TemplateResolver

This class renders a template by resolving merge fields in the template text. There is only one global method definition in this class (see below).

Method: resolveTemplate

resolveTemplate(String originalTemplate, SObjectType objectType, List recordIds, Set extraFields)
Return Type : Map

The returned **Map** object key will reference the record in the object. The corresponding value will contain the template text.

Parameters:

- originalTemplate – the text of the template. If this value is passed as null or blank, an exception error will be raised.
- objectType – the sObjectType of the object instance that corresponds to

the template. An exception error will be raised if this is null.
- recordIds – IDs of the records in the object instance. The field values will be used to populate the dynamic fields in originalTemplate. An exception error will be raised if this is null.
- extraFields – fields in addition to those in the template text.

Suppose mobile number of the record is also needed, then the user will not have to write separate query to fetch mobile number. Just mention that field in the set of extraFields, returned map will fetch the mobile number also.

How to render the template text

Now, let's see how to use the **TemplateResolver** class and **resolveTemplate** method in Apex.

First, a template containing merge fields must be created from the **Template** tab in SMS-Magic. It will be stored in the **SMS Template** custom object, which is accessible with API name

smagicinteract__SMS_Template__c.

Any template that corresponds to an object instance (and template text) will contain the merge fields of that object. Resolving a template consists of substituting actual field values in the merge fields of the template text.

Template text is resolved with the **TemplateResolver** class using its **resolveTemplate** method. Follow these steps:

1. Select a predefined template record that corresponds to a Salesforce.com object.
2. Using **Schema.getGlobalDescribe()**, set **sObjectType** to the object that correspond to the template.
3. Create a list of IDs for the records that correspond to the template.
4. Optionally, create a set of **extraFields**.
5. Create an instance of the object **smagicinteract.TemplateResolver** class.
6. Call the **resolveTemplate** method, using that object instance.

Code Sample

Feel free to copy and modify this code example:

```
String templateText ='Hello  {!Contact.name}, Welcome to SMS-Magic . Regards,
{!Contact.Account.name}';
List contactlist=[select id from Contact limit 10];
List IdList=new List();
for(Contact key:contactlist){
IdList.add(key.id);


Set extraFieldSet=new Set{'name' , 'department', 'MobilePhone'};
smagicinteract.TemplateResolver temp = new smagicinteract.TemplateResolver();
Map objectTextmap =temp.resolveTemplate(templateText,
contactlist.getSObjectType(), IdList, extraFieldSet);
```

Double checking

1. Length of List of Objects (Number of record IDs) should not be greater than **200** else it will violate SF Limits.
2. Don’t send the TemplateText, ObjectType, or the record list as null or blank.
3. Don’t pass the Object ID field in **extraFields**.
4. If there are more than 20 merge fields in the template, the result will be null.
5. If the user does not have field-level permissions for each of the fields in the template text, then returned result will be null.
6. It is also necessary for the user to have read permission for the object.

PushSMSCallout

The purpose of this API resource is to send SMS messages in bulk. It does not create records of smsMagicObject inside your Salesforce organization.

The Apex API class contains a **pushSMSCallout()** static method, which accepts the list of objects in **smagicinteract__smsMagic__c** and returns a text response.

You can use this API resource as follows:

String responseText = smagicinteract.ApexAPI.pushSMSCallout(smsObjectList);

There are two ways to use **PushSMSCallout**.

1.When we call from a trigger, it’s necessary to use future annotation

@future(callout=true)

This is necessary because Salesforce does not permit making a callout directly from a trigger. Here is a code example:

```
trigger SendSmsToContact on Contact (after insert) {
List conList=Trigger.new;
List idList=new List();

for(Contact key:conList){
idList.add(key.id);
}
sendSmsToContactHelperClass.sendSms(idList);
}
```

```
public class SendSmsToContactHelperClass {
@future(callout=true)
public static void sendSms(List idList){
String query='select id,MobilePhone,Name from Contact where id in : idList';
List conList=Database.query(query);
List smsObjectList = new List();
String templateText = 'test SMS by Screen Magic';
String senderId = 'smsMagic'; // Please replace the 'smsMagic' with your
relevant sender ID.

for(Contact contact:conList){
smagicinteract__smsMagic__c smsObject = new smagicinteract__smsMagic__c();
if(contact.MobilePhone != null){
smsObject.smagicinteract__SenderId__c = senderId;
smsObject.smagicinteract__PhoneNumber__c =contact.MobilePhone;
smsObject.smagicinteract__Name__c =contact.Name;
smsObject.smagicinteract__ObjectType__c = 'Contact';
smsObject.smagicinteract__disableSMSOnTrigger__c =1;
smsObject.smagicinteract__external_field__c =
smagicinteract.ApexAPI.generateUniqueKey();
smsObject.smagicinteract__SMSText__c = templateText;
smsObjectList.add(smsObject);
}
}

/*
* Note : When you are using pushSMSCallout method to send the SMS
* please make sure that smagicinteract__disableSMSOnTrigger__c
* should have value as 1.
*/
String response = smagicinteract.ApexAPI.pushSMSCallout(smsObjectList);
Database.insert(smsObjectList,false);
}
}
```

2. The other way to use **PushSMSCallout** is a direct callout—when we do not call from a trigger. Here is a code example.

```
String query='select id,MobilePhone,Name from Contact where id in : idList';
List conList=Database.query(query);
List smsObjectList = new List();
String templateText = 'test SMS by Screen Magic';
String senderId = 'smsMagic'; // Please replace the 'smsMagic' with your
relevant sender ID.
for(Contact contact:conList){
smagicinteract__smsMagic__c smsObject = new smagicinteract__smsMagic__c();
if(contact.MobilePhone != null){
smsObject.smagicinteract__SenderId__c = senderId;
smsObject.smagicinteract__PhoneNumber__c =contact.MobilePhone;
smsObject.smagicinteract__Name__c =contact.Name;
```

```
smsObject.smagicinteract__ObjectType__c = 'Contact';
smsObject.smagicinteract__disableSMSOnTrigger__c =1;
smsObject.smagicinteract__external_field__c =
smagicinteract.ApexAPI.generateUniqueKey();
smsObject.smagicinteract__SMSText__c = templateText;
smsObjectList.add(smsObject);
}
}
/*
* Note : When you are using pushSMSCallout method to send the SMS
* please make sure that smagicinteract__disableSMSOnTrigger__c
* should have value as 1.
*/
String response = smagicinteract.ApexAPI.pushSMSCallout(smsObjectList);
Database.insert(smsObjectList,false);
```

Alternative method for PushSMSCallout

If you do not want to use **PushSMSCallout**, you can use the **SendSMS** code of trigger to send out SMS. When you insert an **SMSHistory** record, it invokes a trigger code – which will send out an SMS message to your recipient. You'll need to set the value of **disablesmsonTrigger** to 0.

Below is a sample code.

```
String query='select id,MobilePhone,Name from Contact where id in : idList';
List conList=Database.query(query);
List smsObjectList = new List();
String templateText = 'test SMS by Screen Magic';
String senderId = 'smsMagic'; // Please replace the 'smsMagic' with your
relevant sender ID.
for(Contact contact:conList){
smagicinteract__smsMagic__c smsObject = new smagicinteract__smsMagic__c();
if(contact.MobilePhone != null){
smsObject.smagicinteract__SenderId__c = senderId;
smsObject.smagicinteract__PhoneNumber__c =contact.MobilePhone;
smsObject.smagicinteract__Name__c =contact.Name;
smsObject.smagicinteract__ObjectType__c = 'Contact';
smsObject.smagicinteract__disableSMSOnTrigger__c =0;
smsObject.smagicinteract__external_field__c =
smagicinteract.ApexAPI.generateUniqueKey();
smsObject.smagicinteract__SMSText__c = templateText;
smsObjectList.add(smsObject);
}
}
Database.insert(smsObjectList,false);
```

Limitation

The maximum size of a callout request or response is 3 MB.

Send Bulk SMS Messages

Follow the steps below to sending SMS messages in bulk for all records that correspond to a particular Contact object.

1. Create a List of type **smagicinteract__smsMagic__c**.
2. Iterate the contact list.
3. Create an object of **smagicinteract__smsMagic__c**.
4. Assign fields to the object.
5. Iterate through and perform the SMSCallout to send the SMS.
6. Finally, use database.insert to insert the object list.

Here is some example code that sends bulk SMS messages.

```
List smsObjectList = new List();
List  conList; //List of contact populated from the visual force page or
fetch from the database.
String templateText = 'test SMS by Screen Magic';
String senderId = 'smsMagic'; // Please replace the 'smsMagic' with your
relevant sender ID.
for(Contact contact :conList){
   smagicinteract__smsMagic__c smsObject = new smagicinteract__smsMagic__c();
   if(contact.MobilePhone != null){

     smsObject.smagicinteract__SenderId__c = senderId;
     smsObject.smagicinteract__PhoneNumber__c = contact.MobilePhone;
     smsObject.smagicinteract__Name__c = contact.Name;
     smsObject.smagicinteract__ObjectType__c = 'Contact';
     smsObject.smagicinteract__disableSMSOnTrigger__c = 1;
     smsObject.smagicinteract__external_field__c =
     smagicinteract.ApexAPI.generateUniqueKey();
     smsObject.smagicinteract__SMSText__c = templateText;
     smsObjectList.add(smsObject);
   }
}
/*
 * Note : When you are using pushSMSCallout method to send the SMS
 *        please make sure that smagicinteract__disableSMSOnTrigger__c
 *        should have value as 1.
 */
String response = smagicinteract.ApexAPI.pushSMSCallout(smsObjectList);
Database.insert(smsObjectList,false);
```