

**API Specification guide for developers – Version 1.1.2**

## Welcome to Digimiles Developers hub. With this API documentation guide, we will be assisting you here to configure, access and use of Digimiles SMS API into your CRM/app/software.

**#1. API URL, Parameters & Error codes**

[http://www.loginurl.com/bulksms/bulksms?username=di78-](http://www.loginurl.com/bulksms/bulksms?username=di78-XXXX&password=XXXX&type=0&dlr=1&destination=99160XXXXX&source=DigiML&message=Your) [XXXX&password=XXXX&type=0&dlr=1&destination=99160XXXXX&source=DigiML&message](http://www.loginurl.com/bulksms/bulksms?username=di78-XXXX&password=XXXX&type=0&dlr=1&destination=99160XXXXX&source=DigiML&message=Your)

[=Your](http://www.loginurl.com/bulksms/bulksms?username=di78-XXXX&password=XXXX&type=0&dlr=1&destination=99160XXXXX&source=DigiML&message=Your) OTP is {#var#}&entityid=XXXXXXXXXXXX&tempid=XXXXXXXXXXXXXXX

**Parameters:**

1. **Username:** Username of your SMS Account di78- XXXX

**Example: -** di78-trans

1. **Password:** Password of your SMS Account = miles

## **type:** Indicates the type of message.

**Values for "type":-**

## 0: Plain Text (GSM 3.38 Character encoding)

1: Flash Message (GSM 3.38 Character encoding) 2: Unicode

3: Reserved

1. **dlr:**

## Indicates whether the client wants delivery report for this message Range of values for "dlr":-

0: No Delivery report required

1: Delivery report required

1. **Destination:** Mobile number to which message has to go (Example: +919916011355 or 9916011355)
2. **Source:** Sender ID (6 Alphabetical)
3. **Entity ID** – 19 digit entity ID of your Principal Entity (P.E). It can be accessed from your DLT account
4. **Temp ID** – 19 digit template ID and it can be accessed from your DLT account.

# Error Codes:

## **1701:** Success, Message Submitted successfully, in this case you will receive the response 1701|<CELL\_NO>|<MESSAGE ID>, the message Id can Then be used later to map the delivery reports to this message.

**1702:** Invalid URL Error, This means that one of the parameters was not provided or left blank

**1703:** Invalid value in username or password field

**1704:** Invalid value in "type" field

**1705:** Invalid Message

**1706*:*** Invalid Destination

**1707:** Invalid Source (Sender) **1708:** Invalid value for "dlr" field **1709:** User validation failed **1710:** Internal Error

**1025:** Insufficient Credit

**1028:** Spam message content

## **1042:** Number blocked at operator end

**#2. Message length (Standard GSM & Unicode)**

**#1. Standard GSM 03.38 Character Set**

## Each SMS has a length of 160 characters, including spaces.

|  |  |
| --- | --- |
| **No. of credits deducted** | **No. of characters** |
| 1 | 160 character |
| 2 | 306 (2\*153) |
| 3 | 459 (3\*153) |
| … | … |

According to the standard GSM 03.38-character set, one can send up to 1000 characters as multi-part text which will be split into 7 text messages according to the receiver’s device.

More than 1000 characters text message will split into more text message.

**#2. Unicode character**

Unicode messaging, such as Hindi, Telugu, Kannada, Marathi, Malayalam, Tamil etc, restricts your text to 70 characters per SMS.

|  |  |
| --- | --- |
| **No. of credits deducted** | **No. of characters** |
| 1 | 70 character |
| 2 | 134 (2\*67) |
| 3 | 201 (3\*67) |
| … | … |

**#3. Unicode Messaging (API)**

**What can you do with this API?**

You can send a Unicode SMS to a number(s), in the language of your choice.

**API for sending a Unicode SMS**

[http://www.loginurl.com/bulksms/bulksms?username=di78-](http://www.loginurl.com/bulksms/bulksms?username=di78-XYZ&password=mile&type=2&dlr=1&destination=99160XXXXX&source=DIGIML&message=Test) [XYZ&password=mile&type=2&dlr=1&destination=99160XXXXX&source=DIGIML&message=Test](http://www.loginurl.com/bulksms/bulksms?username=di78-XYZ&password=mile&type=2&dlr=1&destination=99160XXXXX&source=DIGIML&message=Test)

## On calling the above link by replacing the username, password, mobile number and source, you should get the Unicode SMS.

The message has to be encoded on the UTF-16BE format and the type parameter has to be set to 6 i.e. (type=2).

**#4. Credit Check API & Response**

This section helps you to view the account’s credit balance by using credit check api.

**GET**

<http://www.loginurl.com/CreditCheck/checkcredits?username=X&password=Y>

**Response Message for Credit Check API**

|  |  |
| --- | --- |
| **Name** | **Description** |
| AUTHORIZATION\_FAILED | User is inactive or credentials are wrong |
| BALANCE | Credit balance remaining in the account. |
| INVALID\_URL | Username or Password or both are incorrect |
| INTERNAL\_ERROR | Cannot fetch credit balance |
| PERMISSION\_DENIED | You have reached the maximum number of allowed this |



**#5. Delivery Push API, Response & Error Codes**

**POST**

The DLR push API sends the delivery report to the client’s URL/ server address.

The URL for each client would look like <http://www.abc.net/Demo/Demo.aspx>

Above URL is the server address with folder name where SMS reports to be retrieved.

The parameters appended to the URL would be static. The parameters are explained below:

|  |  |
| --- | --- |
| **Parameter Name** | **Description** |
| sSender | Source Address |
| sMobileNo | Destination number of the sent message |
| sStatus | Status of the message.( 'UNKNOWN', 'ACKED', |
| 'ENROUTE', 'DELIVRD', 'EXPIRED', 'DELETED', |
| 'UNDELIV', 'ACCEPTED', 'REJECTD') |
| sMessageId | The unique identification for message given at the time of |
| submission. |
| dtDone | Date-Time at which delivery report is received. |
| dtSubmit | Date-Time when the message is submitted. |

**Error codes**

|  |  |
| --- | --- |
| **Code** | **Description** |
| 200 | OK: The request has succeeded. The information |
| returned with the response is dependent on the |
| method used in the request. |
| 202 | Accepted: The request has been accepted for |
| processing, but the processing has not been |
| completed. |

**#6. Schedule SMS HTTP API Specifications**

1. **Schedule SMS**

### The below URL is an example of the API used to schedule bulk messages.

http://<ipaddress>:<port>/bulksms/schedulesms?username=XXXXX&password=XXXXX&messag e=XXXXXX&type=X&dlr=X&source=XXXXX&destination=XXXXXXXXXX&date=XXXX&time=XXXX&g mt=XXXXX[&url=XXXX]

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Username | Your SMS account username |
| Password | Password of the username |
| Message | Message to be sent |
| Destination | Mobile number. More than one mobile number  needs to be separated by comma (,). |
| Type | Type of the message 0 – Text  1 – Flash  2- Unicode  4 – WAP  6 – Unicode Flash |
| Source | Sender ID |
| Date | Scheduled date for the message. Data needs to be in the following format:  MM/DD/YYYY  Example: 03/03/2013 |
| Time | Scheduled time for the message. Time needs to be in the following format:  hh:mm am/pm  hh: Hours range from 00 to 12  mm: Examples of minutes: 00, 15,30  Example of scheduletime: 03:45 am OR 11:51 pm |
| Gmt | GMT +hh:mm OR GMT -hh:mm OR  +hh:mm OR -hh:mm  hh: Hours range from 0 to 12  mm: Examples of minutes: 00 or 30  Example: GMT +1:30 OR -10:00 |
| url | URL is used in case of type=4 messages .In other  cases, it can be empty. |
| dlr | Indicates whether client wants delivery report for  this message; 0 – No Report required, 1- Required. |

**Response codes for Schedule HTTP SMS API**

|  |  |
| --- | --- |
| **Error code** | **Description** |
| **1701** | Message scheduled successfully. |
| **1702** | One of the parameter is missing. |
| **1703** | User authentication has failed. |
| **1704** | Invalid message type |
| **1705** | Invalid message. |
| **1706** | Invalid destination. |
| **1707** | Invalid sender. |
| **1710** | Unknown error. |
| **1712** | Bad DB connection. |
| **1713** | Destinations error |
| **1901** | Invalid XML content. |
| **1902** | Bad schedule date |
| **1903** | Invalid GMT. |
| **1904** | Bad schedule time. |
| **1905** | Invalid date and time. |
| **1813** | Error while creating job file for the scheduled job |

**#7. Sample coding of various platforms (Dot Net, PHP & Java)**

**Calling HTTP API Using .Net**

Imports System.IO Imports System.Net Imports System.Data

Partial Class SendUsingSMPP Inherits System.Web.UI.Page

Protected Sub Page\_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

Dim WebRequest As Net.WebRequest 'object for WebRequest Dim WebResonse As Net.WebResponse 'object for WebResponse ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ''''''''

' DEFINE PARAMETERS USED IN URL ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ''''''''

'To what server you need to connect to for submission 'i.e. Dim Server As String = "xxxxx.xxxxx.xxxxx"

Dim Server As String = ""

'Port that is to be used like 8080 or 8000 Dim Port As String = "" 'Username that is to be used for submission 'i.e. Dim UserName As String = "tester" Dim UserName As String = ""

' password that is to be used along with username 'i.e. Dim Password As String = "password"

Dim Password As String = ""

'What type of the message that is to

be sent. '0:means plain text

'1:means flash

'2:means Unicode (Message content should be in Hex) '6:means Unicode Flash(Message content should be in Hex) Dim type As Integer = 0 'Message content that is to be transmitted

Dim Message As String = "Test Message" 'Url Encode message

Message = HttpUtility.UrlEncode(Message) If (Message = 2) Or (Message = 6) Then Message = ConvertToUnicode(Message)

End If

'Require DLR or not

'0:means DLR is not Required '1:means DLR is Required Dim DLR As Integer = 1

'Sender Id to be used for submitting the message 'i.e. Dim SenderName As String = "test"

Dim Source As String = ""

'Destinations to which message is to be sent For submitting more than one

'destination at once destinations should be comma separated Like '91999000123,91999000124

Dim Destination As String = ""

'''''''CODE COMPLETE TO DEFINE PARAMETER''''''''''''''''

Dim WebResponseString As String = ""

Dim URL As String = "https://" & Server & ":" & Port &type & "&dlr=" & DLR & "&destination=" & Destination & "&source=" &Source & "&message=" & Message & ""

WebRequest = Net.HttpWebRequest.Create(URL) 'Hit URL Link WebRequest.Timeout = 25000

Try

WebResonse = WebRequest.GetResponse 'Get Response Dim reader As IO.StreamReader = New IO.StreamReader(WebResonse.GetResponseStream) 'Read Response and store in variable

WebResponseString = reader.ReadToEnd() WebResonse.Close()

Response.Write(WebResponseString) 'Display Response. Catch ex As Exception

WebResponseString = "Request Timeout" 'If any exception occur.

Response.Write(WebResponseString)

End Try End Sub

### ‘Function to convert string to Unicode if message type =2 and 6. PublicFunction ConvertToUnicode (ByVal str As String) As String

Dim ArrayOFBytes() As Byte = System.Text.Encoding.Unicode.GetBytes(str) Dim UnicodeString As String = ""

Dim v As Integer

For v = 0 To ArrayOFBytes.Length - 1 If v Mod 2 = 0 Then

Dim t As Integer = ArrayOFBytes(v) ArrayOFBytes(v) = ArrayOFBytes(v + 1) ArrayOFBytes(v + 1) = t

End If

Next

For v = 0 To ArrayOFBytes.Length - 1 Dim c As String = Hex$(ArrayOFBytes(v))

If c.Length = 1 Then c = "0" & c

End If

UnicodeString = UnicodeString & c Next

Return UnicodeString End Function

End Class

# Calling HTTP API Using php:

<?php

class Sender{ var $host;

var $port;

/\*

\* Username that is to be used for submission

\*/

var $strUserName;

/\*

* password that is to be used along with username

\*/ var

$strPassword; /\*

* Sender Id to be used for submitting the message

\*/ var

$strSender; /\*

* Message content that is to be transmitted

\*/

var $strMessage;

/\*

* Mobile No is to be transmitted.

\*/ var

$strMobile; /\*

* What type of the message that is to be sent
* <ul>
* <li>0:means plain text</li>
* <li>1:means flash</li>
* <li>2:means Unicode (Message content should be in Hex)</li>
* <li>6:means Unicode Flash (Message content should be in Hex)</li>

\* </ul>

\*/

var $strMessageType;

/\*

* Require DLR or not
* <ul>
* <li>0:means DLR is not Required</li>
* <li>1:means DLR is Required</li>

\* </ul>

\*/

var $strDlr;

private function sms unicode($message){

$hex1='';

if (function\_exists('iconv')) {

$latin = @iconv('UTF-8', 'ISO-8859-1', $message); if (strcmp($latin, $message)) {

$arr = unpack('H\*hex', @iconv('UTF-8', 'UCS- 2BE', $message));

$hex1 = strtoupper($arr['hex']);

}

if($hex1 ==''){

$hex2='';

$hex='';

for ($i=0; $i < strlen($message); $i++){

$hex = dechex(ord($message[$i]));

$len =strlen($hex);

$add = 4 - $len; if($len < 4){

for($j=0;$j<$add;$j++){

$hex="0".$hex;

}

}

$hex2.=$hex;

}

return $hex2;

}

else{

return $hex1;

}

}

else{

print 'iconv Function Not Exists !';

}

}

//Constructor..

public function Sender ($host,$port,$username,$password,$sender, $message,$mobile,

$msgtype,$dlr){ $this->host=$host;

$this->port=$port;

$this->strUserName = $username;

$this->strPassword = $password;

$this->strSender= $sender;

$this->strMessage=$message; //URL Encode The Message..

$this->strMobile=$mobile;

$this->strMessageType=$msgtype;

$this->strDlr=$dlr;

}

public function Submit(){ if($this-

>strMessageType=="2" ||

$this->strMessageType=="6") {

//Call The Function Of String To HEX.

$this->strMessage = $this->sms unicode($this-

>strMessage); try{

//Smpp http Url to send sms.

$live\_url="http://".$this->host.":".$this->port."/bulksms/bulksms?username=".$this-

>strUserName."&password=".$this->strPassword."&type=".$this-

>strMessageType."&dlr=".$this-

>strDlr."&destination=".$this->strMobile."&source=".$this->strSender."&message=".$this-

>strMessage."";

$parse\_url=file($live\_url); echo $parse\_url[0];

}catch(Exception $e){

echo 'Message:' .$e->getMessage();

}

}

else

$this->strMessage=urlencode($this->strMessage); try{

//Smpp http Url to send sms.

$live\_url="http://".$this->host.":".

$this->port."/bulksms/bulksms?username=".$this->strUserName."&password=".$this-

>strPassword."&type=".$this ->strMessageType."&dlr=".$this-

>strDlr."&destination=".$this->strMobile."&source=".$this-

>strSender."&message=".$this->strMessage."";

$parse\_url=file($live\_url); echo $parse\_url[0];

}

catch(Exception $e){

echo 'Message:' .$e->getMessage();

}

}

}

//Call The Constructor.

$obj = new Sender("IP","Port","","","Tester"," " "," 919990001245 ,"2","1");

$obj->Submit ();

?>

# Calling HTTP API Using Java:

import java.io.BufferedReader; import java.io.DataOutputStream; import java.io.InputStreamReader; import java.net.HttpURLConnection; import java.net.URL;

import java.net.URLEncoder;

import javax.net.ssl.HostnameVerifier; import javax.net.ssl.SSLSession;

/\*\*

* An Example Class to use for the submission using HTTP API You can perform
* your own validations into this Class For username, password,destination,
* source, dlr, type, message, server and port

\*

\*/

public class Sender {

// Username that is to be used for submission String username;

// password that is to be used along with username String password;

// Message content that is to be transmitted String message; /\*\*

* What type of the message that is to be sent <ul> <li>0:means plain
* text</li> <li>1:means flash</li> <li>2:means Unicode (Message content
* should be in Hex)</li> <li>6:means Unicode Flash (Message content should \* be in Hex)</li> </ul>

\*/

String type;

/\*\*

* Require DLR or not <ul> <li>0:means DLR is not Required</li> <li>1:means \* DLR is Required</li> </ul>

\*/

String dlr;

/\*\*

* Destinations to which message is to be sent For submitting more than one
* destination at once destinations should be comma separated Like

\* 91999000123,91999000124

\*/

String destination;

// Sender Id to be used for submitting the message String source;

// To what server you need to connect to for submission String server;

// Port that is to be used like 8080 or 8000 int port;

public Sender(String server, int port, String username, String password, String message, String dlr, String type, String destination, String source) { this.username =

username; this.password = password; this.message = message; this.dlr = dlr; this.type = type; this.destination = destination; this.source = source; this.server = server; this.port = port;

}

private void submitMessage() { HttpURLConnection httpConnection = null; try {

// Url that will be called to submit the message

URL sendUrl = new URL("https://" + this.server + ":" + this.port

+ "/bulksms/bulksms");

HostnameVerifier hostVerfier = new HostnameVerifier() {

public boolean verify(String urlHostName, SSLSession session) { return true;

}

};

trustAllHttpsCertificates();

httpConnection = (java.net.HttpURLConnection) sendUrl.openConnection();

// This method sets the method type to POST so that

// will be send as a POST request httpConnection.setRequestMethod("POST");

// This method is set as true wince we intend to send

// input to the server httpConnection.setDoInput(true);

// This method implies that we intend to receive data from server. httpConnection.setDoOutput(true);

// Implies do not use cached data httpConnection.setUseCaches(false);

// Data that will be sent over the stream to the server. DataOutputStream dataStreamToServer = new DataOutputStream( httpConnection.getOutputStream()); dataStreamToServer.writeBytes("username="

+ URLEncoder.encode(this.username, "UTF-8") + "&password=" + URLEncoder.encode(this.password, "UTF-8")

+ "&type=" URLEncoder.encode(this.type, "UTF-8") + "&dlr="

+ URLEncoder.encode(this.dlr, "UTF-8") + "&destination="

+ URLEncoder.encode(this.destination, "UTF-8") + "&source=" + URLEncoder.encode(this.source, "UTF-8") + "&message=" + URLEncoder.encode(this.message, "UTF- 8")); dataStreamToServer.flush();

dataStreamToServer.close();

// Here take the output value of the server.

BufferedReader dataStreamFromUrl = new BufferedReader( new InputStreamReader(httpConnection.getInputStream())); String dataFromUrl = "", dataBuffer = "";

// Writing information from the stream to the buffer

while ((dataBuffer = dataStreamFromUrl.readLine()) != null)

{ dataFromUrl += dataBuffer;

}

/\*\*

* Now dataFromUrl variable contains the Response received from the \* server so we can parse the response and process it accordingly. \*/

dataStreamFromUrl.close(); System.out.println("Response: " + dataFromUrl); } catch (Exception ex) { ex.printStackTrace();

} finally {

if (httpConnection != null) { httpConnection.disconnect();

}

}

}

public static void main(String[] args)

{ try {

// Below exmaple is for sending Plain text Sender s = new Sender("server", 8443, "xxxx", "xxxx", "test for unicode", "1", "0", "440000xxx", "Update"); s.submitMessage();

// Below exmaple is for sending unicode Sender s1 = new Sender("server", 8443, "xxxx", "xxx", convertToUnicode("test for unicode").toString(), "1", "2", "44000xx", "Update"); s1.submitMessage();

} catch (Exception ex) {

}

}

/\*\*

* Below method converts the unicode to hex value

\*

* @param regText \* @return

\*/

private static StringBuffer convertToUnicode(String regText) { char[] chars = regText.toCharArray(); StringBuffer hexString = new StringBuffer();

for (int i = 0; i < chars.length; i++) {

String iniHexString = Integer.toHexString((int) chars[i]); if (iniHexString.length() == 1) { iniHexString = "000" + iniHexString;

} else if (iniHexString.length() == 2) { iniHexString = "00" + iniHexString; } else if (iniHexString.length() == 3) { iniHexString = "0" + iniHexString;

}

hexString.append(iniHexString);

}

System.out.println(hexString); return hexString;

}

private static void trustAllHttpsCertificates() throws Exception {

// Create a trust manager that does not validate certificate chains: javax.net.ssl.TrustManager[] trustAllCerts =

new javax.net.ssl.TrustManager[1]; javax.net.ssl.TrustManager tm = new miTM(); trustAllCerts[0] = tm; javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL"); sc.init(null, trustAllCerts, null); javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory( sc.getSocketFactory());

}

public static class miTM implements javax.net.ssl.TrustManager, javax.net.ssl.X509TrustManager { public java.security.cert.X509Certificate[] getAcceptedIssuers()

{ return null;

### }

public boolean isServerTrusted( java.security.cert.X509Certificate[] certs) { return true;

### }

public boolean isClientTrusted(

java.security.cert.X509Certificate[] certs) { return true;

}

public void checkServerTrusted( java.security.cert.X509Certificate[] certs, String authType) throws java.security.cert.CertificateException {

return;

}

public void checkClientTrusted( java.security.cert.X509Certificate[] certs, String authType) throws java.security.cert.CertificateException {

return;

}

}