Enterprise API Document

https://enterprise.smsgupshup.com

Published: 30th September 2020

Contents

1. Overview

This User Manual provides specifications of the API for the automated sending of SMS via the Internet through HTTPS mode. This guide is intended for the developers and clients alike who plan to integrate their systems with Bulk SMS service of Gupshup.

2. The API End Point

Our app resides at **https://enterprise.smsgupshup.com/GatewayAPI/rest**

Please use this URL for all API methods. Our API has been designed to allow you to access an SSL Enabled connection for added security i.e. supports Hypertext Transfer Protocol over Secure Socket Layer (HTTPS) protocol.

2.1. User Authentication Scheme

Currently, our API supports only Plain Authentication Scheme for user. This authentication scheme requires only the user ID and password. The connection security is provided through HTTPS protocol.

2.2. Data Encryption

In addition to SSL, our API has been designed to allow you to securely send sensitive data to the Gupshup platform by encrypting the data using Advanced Encryption Standard i.e. AES 256-bit encryption. On your request (please reach out to us at 022 42006799 or email us at (**enterprise-support@smsgupshup.com),** a 256-bit symmetric key is generated by Gupshup and set up for your account.

You must use this key to encrypt API parameter values when sending the API request. Once the request is received by Gupshup, the payload is decrypted by Gupshup and sent ahead to the operator.

3. Pre Start Checklist

Here's what you need to know prior to using any API

- User name & password. If you don't have an account you can create one at **https://enterprise.smsgupshup.com**
- URL encoding of your message, password etc.

For any queries our support is available for you at 022 42006799 or email us at **enterprise-support@smsgupshup.com**

4. Message Length Chart

No of Messages	Text characters	Unicode characters
1 SMS	160-characters	70-characters
2 SMS	306-characters	134-characters
3 SMS	459-characters	201-characters
4 SMS	612-characters	268-characters
5 SMS	765-characters	335-characters
6 SMS	918-characters	402-characters
7 SMS	1071-characters	469-characters
8 SMS	1224-characters	536-characters
9 SMS	1377-characters	603-characters
10 SMS	1530-characters	670-characters
11 SMS	1683-characters	737-characters
12 SMS	1836-characters	804-characters
13 SMS	1989-characters	871-characters
14 SMS	2142-characters	938-characters
15 SMS	2295-characters	1005-characters
16 SMS	2448-characters	1072-characters
17 SMS	2601-characters	1139-characters
18 SMS	2754-characters	1206-characters
19 SMS	2907-characters	1273-characters
20 SMS	3060-characters	1340-characters
21 SMS	3213-characters	1407-characters
22 SMS	3366-characters	1474-characters
23 SMS	3519-characters	1541-characters
24 SMS	3672-characters	1608-characters
25 SMS	3825-characters	1675-characters
26 SMS	3978-characters	1742-characters
27 SMS	4131-characters	1809-characters
28 SMS	4284-characters	1876-characters
29 SMS	4437-characters	1943-characters

30 SMS	4590-characters	2000-characters
31 SMS	4743-characters	
32 SMS	4896-characters	
33 SMS	5049-characters	
34 SMS	5202-characters	
35 SMS	5355-characters	
36 SMS	5508-characters	
37 SMS	5661-characters	
38 SMS	5814-characters	
39 SMS	5967-characters	
40 SMS	6120-characters	
41 SMS	6273-characters	
42 SMS	6426-characters	
43 SMS	6579-characters	
44 SMS	6732-characters	
45 SMS	6885-characters	
46 SMS	7038-characters	
47 SMS	7191-characters	
48 SMS	7344-characters	
49 SMS	7497-characters	
50 SMS	7650-characters	
51 SMS	7803-characters	
52 SMS	7956-characters	
53 SMS	8000-characters	

5. Sending a Single Message

Try the URL below to send a message. (Or copy-paste it into your browser's address bar). We will then take a look at other details.

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=SendMessage&send_to=9199xxxxxxxx&msg=Welcome%20to%20SMS%20GupShup%20API&msg_type=TEXT&userid=XXXXXX&auth_scheme=plain&password=password&v=1.1&format=text
```

5.1. Parameters

The following are the required parameters:.

Parameter	Description
userid	Specify the user id provided by Gupshup
password	Password provided by Gupshup for authentication of user id
send_to	Phone no. of the recipient
msg	Text that needs to be sent on mobile handset. Text should be URL encoded

Parameter	Value	Description
userid	Account number provided by Enterprise SMS GupShup	The number must be in pure numeric format with no special characters.
password	UrlEncoded string of UTF-8 characters	Password provided by Gupshup for authentication of user id .The password must be the same as used to log on to the Enterprise SMS GupShup website.
auth_scheme	Plain	Only Plain authentication supported.
method	sendMessage	Method for performing a specific action.
v	1.1	Default version is 1.1, unless otherwise specified.
format	TEXT, XML, JSON	specific format for response message

send_to	Phone no. of the receiver	The number must be in pure numeric format with no special characters
msg	UrlEncoded string of UTF-8 characters	The message that needs to be sent. It can contain alphanumeric & special characters
msg_type	Text, Unicode_text, or flash, VCARD, binary	Indicates the type of the message to be sent
Optional		
port	Port Number	It is a pure number and needs to be specified if the message is being sent to a port.
timestamp	URL encoded timestamp in the given format	Sender can specify a particular time for sending the message. Accepted timestamp formats are 1. yyyy-MM-dd HH:mm:ss (2008-11-21 23:12:32 or 2008-3-4 2:44:23) 2. MM/dd/yy HH:mm:ss (11/21/08 23:12:32 or 3/4/08 2:44:33) 3. MM/dd/yy hh:mm:ss (11/21/08 12:09:08)
mask	Alphabetical characters	Sender id to be sent with the SMS. It has to be preconfigured 6 characters alphabetical sender id for the enterprise account.
msg_id	Numeric/Alphanumeric value	Custom message ID. This will be attached to Callbacks. You can use your internal IDs to identify the DLRs more easily. 500 characters numeric/alphanumeric value is allowed for message id.
extra	Alphanumeric characters	You can input any text in this parameter and the same value will be forwarded in callback url. 50 alphanumeric characters are allowed for this parameter.
principalEntityId	Numeric characters provided by operators DLT Portal	The Entity ID registered with the DLT platform. Every entity has to register themselves on operators DLT portal to send messages.
dltTemplateId	Numeric characters provided by operators DLT Portal	The Template ID registered with the DLT platform for the Entity ID. Each entity has to register templates to send messages to their registered customers.

DLT Implementation details:

Post DLT implementation, you need to pass principalEntityId and dltTemplateId in the API request.
In case you do not want to make any changes in your API call you can share above data (principalEntityId and dltTemplateId) offline with us i.e. contact your sales manager/Account manager or send an email to our support team at enterprise-support@smsgupshup.com. We will match the data which will be uploaded offline at our end and sent to operators for delivery with correct principalEntityId and dltTemplateId.

5.2. Encoding the Message

The message text should be UrlEncoded. The message should be UrlEncoded (also known as percent encoding) string of UTF-8 characters.

For more information on URL encoding, please see this: http://en.wikipedia.org/wiki/Percent-encoding

Click here to encode message

Original text:

Hi Amar!
Happy Diwali to you
Regards,
nk@w.com

Encoded text:

Hi%20Amar%21%0AHappy%20Diwali%20to%20you%0ARegards%2C%0Ank%40w.com

5.3. Success Response

Successful execution of the request will generate an HTTP 200 response. The response to any request is a string of tokens separated by pipe symbol (|).

A typical success response is

```
success | 919XXXXXXXXX| 728014710863298817-1234567890
```

This indicates that the message request has been successfully accepted for mobile number 919812345678 under a Unique Identifier '728014710863298817-1234567890'. The identifier string is unique for each recipient number and is auto generated at the time of message submission. **First number is the transaction ID and second one is message ID.**

5.4. Failure Response

An error response is generated when any of the required parameters is not specified correctly. The error response will indicate an error code along with the actual error message.

A typical error response is

```
error | 102 | Authentication failed due to invalid userId or password.
```

5.5. API Error codes

Below is the complete list of API failure error codes

Error Code	Description
100	An unknown exception has occurred. Please retry the request after some time.
101	The parameter X is required. Please resend request.
102	Authentication failed due to invalid userId or password.
103	Authentication Failed as userid X does not exist.
104	This user with number is currently disabled. Please contact support for further details.
105	The phone number is not a valid phone number.
106	The method is not supported.
110	Only preconfigured masks are allowed. The mask is not in the list of your preconfigured masks.
111	The uploaded compressed file format is not supported.
112	The phone number field cannot be null.
115	The file type parameter does not match with the uploaded file extension
124	Validity of your SMS pack has expired on. You are not allowed to send messages now.
128	The first row of the file should contain headers. Please see the sample file for details.
171	You are not allowed to perform this action.
175	The "INTERNATIONAL_PHONE" service is disabled for you. Kindly get the service enabled before using this action
183	Cannot process the request before your working start hour.
223	Your post request is exactly same as previous and hence has been rejected. If you still want to post please call Customer Delight at 022 42006799
253	Message specified does not match with any template.

327	Unable to decrypt
326	Encryption key for user was not found

5.6. Examples

Here are examples for the 2 types of message types. You can replace the highlighted text with your credentials and sent the messages.

1. Text

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?msg= Hi%20Test%20Message
&v=1.1&userid=XXXXXXXXX&password=XXXXX&send_to=9XXXXXXXXX&msg_type=text&method=sendMessage
```

Response: success|919899999999|660362025761505631-520576818555598760

Message on mobile: Hi Test Message

2. Unicode

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?msg=%E0%A4%8F%E0%A4%B8%20%E0%A4%8F%E0%A
4%AE%20%E0%A4%8F%E0%A4%B8%20%E0%A4%97%E0%A4%AA%E0%A4%B6%E0%A4%AA&v=1.1&useri
d=XXXXXXXXX&password=XXXXX&send_to=9XXXXXXXXX&msg_type=Unicode_Text&method=sendMessage
```

Response: success | 919899999999 | 660362044229091694-472194266127262392

Received on phone: (copy paste into your browser if you can't see this)

5.7. Comma / Pipe separated API Call

We have already seen an example of sending a message to a single number using a single API call. Users can also send messages to multiple numbers in a single API call by using comma-separated or pipe-separated numbers as values in the 'send_to' parameter.

Here's an example for comma-separated (,) multiple numbers

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=SendMessage&send_to=9199xxxxxxxx,9198xxxxxxxx,9199xxxxxxxx&msg=Welcome%20to%20SMS%20GupShup%20API&msg_type=TEXT&userid=XXXXXXXX&auth_scheme=plain&password=password&v=1.1&format=text
```

Here's an example for pipe-separated (|) multiple numbers

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=SendMessage&send_to=9199xxxxxxxx|9198xxxxxxxx|9199xxxxxxxx&msg=Welcome%20to%20SMS%20GupShup%20API
&msg_type=TEXT&userid=XXXXXXX&auth_scheme=plain&password=password&v=1.1&format=text
```

Just replace highlighted sections in blue with your credentials.

Note: Maximum 1000 numbers can be passed (Comma/Pipe separated) in a single API call.

5.8. Encrypted payload in Send Message API call

A sample request with encrypted data in the payload looks like this. (See sample code in 9)

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=XXXXXXXXX &encrdata={{Base64 Encoded Encrypted  Data}}

https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=XXXXXXXXX&encrdata=Dl77gZZ1yQlkvnJKHFWdr5kBm0sdDQxFWDGT0C0-S17y_-AWolWkJ-DxYxJ4SV7ZWjXemxmEuPZgvg3x1mXclMDv04vl1PrycitcjNd5YLJxvXTZ8ypJcP1YFFJd-zvhYCOAy_TODeap9htSeJ3QEeREU0A3jVI1UfDTdJnFMV5vEfm11JlNJQeojuOK3BO3oPv23pJ64WmNCcUN9FrUlJNYBIIgE84ZGyF1XvR2RAR7yGxqgUgnGVs5u1M74MSpy6ftYjtLl4OVPFP4VdI

where

encrdata={{method=SendMessage&send_to=919XXXXXXXXX&msg=This%20is%20a%20test%20message&msg_type=TEXT&auth_scheme=plain&password=password&v=1.1&format=text}}
```

Note : Output of encrdata will be different every time as the IV(Initialization Vector) value will be unique for every request.

Please note that **"userid"** parameter is mandatory and its value is to be sent in unencrypted format. All the other parameters must be sent in encrypted format using AES 256 encryption and base64 encoded using the key.

You must use the key generated for your enterprise account to encrypt API parameter values when sending the API request. Once the request is received by Gupshup, the payload is decrypted by Gupshup and sent ahead to the operator. Refer 2.2 section.

Steps for Encryption:

1) Form a querystring using rest of the API parameters and its values
Querystring: method=SendMessage&send_to=919XXXXXXXXX&msg=
This%20is%20a%20test%20message&msg_type=TEXT&auth_scheme=plain&password=password&v=1.1
&format=text

2) Encrypt the querystring using AES encryption algorithm(256 bit algorithm)
   - Use only GCM mode
   - Length of IV(Initialization Vector) parameter should be 12 bytes. IV value should be unique for every API request/call.
   - Length of authentication tag should be 16 bytes

3) Output of AES Encryption(256 bit) should be encoded using base 64 Urlsafe.

4) Base64 encoded encrypted cipher will be passed as a payload in **encrdata** parameter.

5) Sample encrypted payload using above steps: Dl77gZZ1yQlkvnJKHFWdr5kBm0sdDQxFWDGT0C0-S17y_-AWolWkJ-DxYxJ4SV7ZWjXemxmEuPZgvg3x1mXclMDv04vI1PrycitcjNd5YLJxvXTZ8ypJcP1YFFJd-zvhYCOAy_TODeap9htSeJ3QEeREU0A3jVI1UfDTdJnFMV5vEfm11JlNJQeojuOK3BO3oPv23pJ64WmNCcUN9FrUIJNYBIIgE84ZGyF1XvR2RAR7yGxqgUgnGVs5u1M74MSpy6ftYjtLI4OVPFP4VdI

Methods Supported: GET, POST

The following are the required parameters:.

Parameter	Description
userid	Specify the user id XXXXXX. This must be sent in plaintext (unencrypted). The number must be in pure numeric format with no special characters.
encrdata	Encrypted parameter and values to be sent in query parameters / payload.

Parameters and Values to be passed in encrdata. Please note both the parameters and its values should be encrypted.

Parameter	Value	Description
password	UrlEncoded string of UTF-8 characters	Password provided by Gupshup for authentication of user id. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.

auth_scheme	Plain	Only Plain authentication supported. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
method	SendMessage	Method for performing a specific action i.e. send a text message. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
v	1.1	Default version is 1.1 and must be passed. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
format	TEXT, XML, JSON	Specific format for API response message Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
send_to	Phone no. of the receiver	The number must be in pure numeric format with no special characters, and should include country code and mobile number of recipient Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
msg	String of UTF-8 characters	The message that needs to be sent. It can contain alphanumeric & special characters Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
msg_type	Text, Unicode_text, or flash, VCARD, binary	Indicates the type of the message to be sent. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
Optional		
timestamp	URL encoded timestamp in the given format	Sender can specify a particular time for sending the message. Accepted timestamp formats are 1. yyyy-MM-dd HH:mm:ss (2008-11-21 23:12:32 or 2008-3-4 2:44:23) 2. MM/dd/yy HH:mm:ss (11/21/08 23:12:32 or 3/4/08 2:44:33) 3. MM/dd/yy hh:mm:ss (11/21/08 23:09:08) Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
msg_id	Numeric value	Custom message ID. This will be attached to Callbacks. You can use your internal IDs to identify the DLRs more easily. 500 characters numeric/alphanumeric value is allowed for message id. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
extra	Alphanumeric characters	You can input any text in this parameter and the same value will be forwarded in callback url. 50 alphanumeric characters are allowed for this parameter. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.

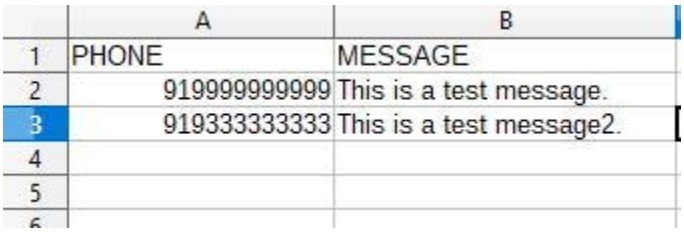
API response format is the same as unencrypted payload.

6. Uploading a File

The file upload API is designed to enable a user to upload a file through which the user can send different messages to large set of numbers in single authentication. The API supports uploading files of the following formats:

1. XLS file
2. CSV file
3. ZIP file containing either an XLS or a CSV file

The first row in each file contains the headers for which the values are provided in the following rows. All the headers are case sensitive in both .csv and .xls files



	A	B
1	PHONE	MESSAGE
2	919999999999	This is a test message.
3	919333333333	This is a test message2.
4		
5		

Checks while using a CSV File:

Ensure the following guidelines are followed when using a CSV file. To know more about CSV files, please see this: http://en.wikipedia.org/wiki/Comma-separated_values

1. CSV file should be UTF-8 encoded.
2. Enclose the headers within quotation marks.
3. Delimit the fields with a comma.
4. Enclose the entries in the MESSAGE field within quotation marks.
5. Do not use a space between the comma and the starting or ending quotes enclosing the field.
6. Ensure that you do not leave a stray quotation mark in the message otherwise an error occurs in processing the message and all the further entries of the file. A stray quotation mark should be nullified by another quotation mark. Message like You”ll be late should be written as “You””ll be late”. Excel exports file to CSV format in the same format.
7. Ensure that there are commas in the right place. Do not add any extra commas or do not skip commas in the entry, else the file will not be processed after the erroneous entry.

6.1. Uploading a file with HTTPS request

User can upload a file with the HTTPS request as a part of multi-part form data. The following parameters need to be sent as a part of request. Please refer sample file upload codes. Refer section 9. Sample codes.

Parameter	Value	Description
Method	xlsUpload	It specifies the way the uploading should take place

filetype	.xls, .csv or zip	It specifies the format of the file being uploaded
msg_type	Text, Unicode_text, or flash, binary	Indicates the type of the message to be sent. Message can be either text, Unicode_Text (non-English) or flash. Currently, a Flash message can be sent only on GSM phones and the maximum length of a flash message is 160 characters.
Parameter (optional)	Value	Description
port	Port Number	It is a pure number and needs to be specified if the message is being sent to a port
timestamp	URL encoded timestamp in the given format	Sender can specify a particular time for sending the message. Accepted timestamp formats are 1. yyyy-MM-dd HH:mm:ss (2008-11-21 23:12:32 or 2008-3-4 2:44:23) 2.MM/dd/yy HH:mm:ss (11/21/08 23:12:32 or 3/4/08 2:44:33) 3.MM/dd/yy hh:mm:ss a (11/21/08 11:12:32 PM or 3/4/08 2:44:33 AM) 4. MM/dd/yy hh:mm a (11/21/08 11:12 PM or 3/4/08 2:44 AM)

6.2. Success Response

When the file upload is successful, the following message is displayed

```
Your file is being processed. Transaction id 3374138381907707211. Please refer upload history below for final status.
```

Final status can be checked on Enterprise panel (**enterprise.smsgupshup.com)** under SMS → Bulk → Upload History.

```
For the transaction id : 3374138381907707211, 10 entries were successfully uploaded and 0 entries failed. Duplicate entries found were: 0
```

6.3. Error Response

Sample error that can occur for an HTTPS request:

When a request is formed properly and if all the entries from the input file fail, then the following response is generated. This status can be checked in enterprise panel (**enterprise.smsgupshup.com)** under SMS → Bulk → Upload History.

For the transaction id : <transaction-id>, all the <num-of-failed-entries> entries failed.

```
For the transaction id : "3161044544991409483", all the "2" entries failed.
```

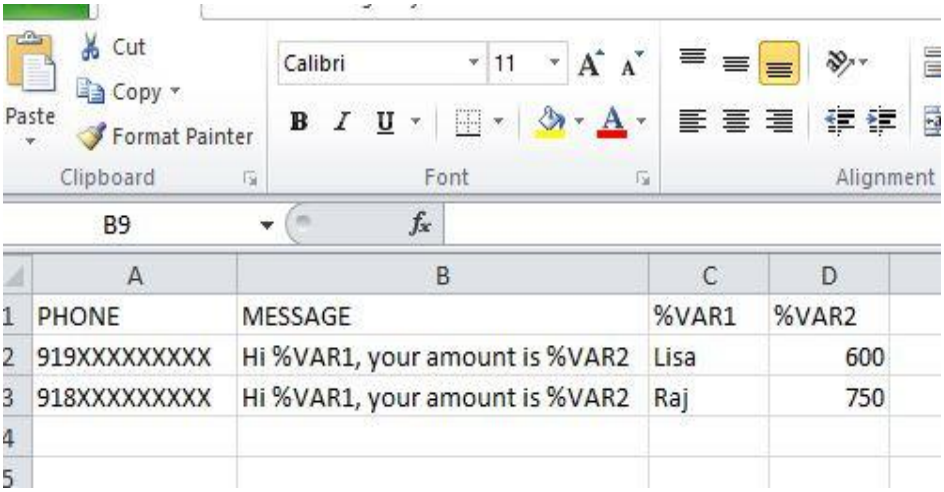
6.4. Customized File Upload

To send customized messages to a list of numbers.

a) Excel sheet format

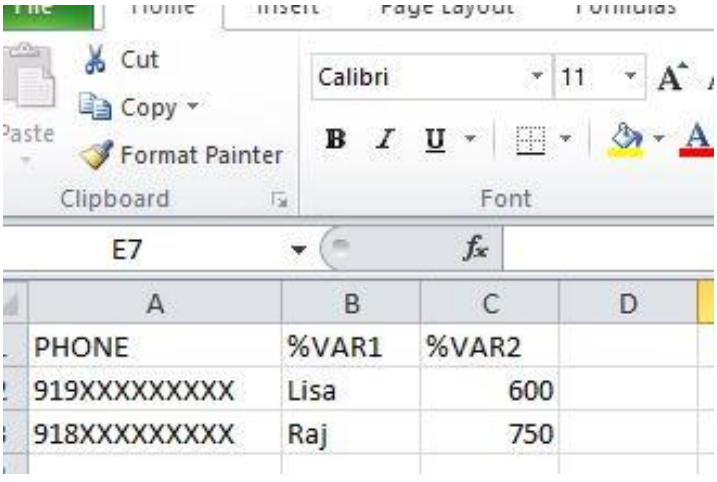
Stated below is a list of the headers to be used to send customized messages.

i) PHONE – list of phone numbers to which messages need to be send.
ii) MESSAGE – Message which needs to be sent out to the customer. This message should clearly indicate the variable fields which are denoted by "%VAR". The first variable in the text will be stated as %VAR1 . The next variable stated as %VAR2 and so forth.
iii) %VAR – This field indicates the variable which needs to be sent to the customer.
As indicated in point 2, for each variable in the text a relevant %VAR header need to be stated.



	A	B	C	D
1	PHONE	MESSAGE	%VAR1	%VAR2
2	919XXXXXXXXX	Hi %VAR1, your amount is %VAR2	Lisa	600
3	918XXXXXXXXX	Hi %VAR1, your amount is %VAR2	Raj	750
4				
5				

Note: If the message text is same for all the customers then the message header can be ignored in the below sample excel file. However the message along with the variable fields needs to be clearly indicated in the API parameter.



	A	B	C	D
	PHONE	%VAR1	%VAR2	
	919XXXXXXXXX	Lisa	600	
	918XXXXXXXXX	Raj	750	

6.5. Scheduling Feature in file upload

Sender can specify a particular time for sending the message. Timestamps column can be added in the file. Formats are as follows:

1. yyyy-MM-dd HH:mm:ss (2008-11-21 23:12:32 or 2008-3-4 2:44:23)
2.MM/dd/yy HH:mm:ss (11/21/08 23:12:32 or 3/4/08 2:44:33)
3.MM/dd/yy hh:mm:ss a (11/21/08 11:12:32 PM or 3/4/08 2:44:33 AM)
4.MM/dd/yy hh:mm a (11/21/08 11:12 PM or 3/4/08 2:44 AM)

	A	B	C
1	PHONE	MESSAGE	TIMESTAMPS
2	919XXXXXXXXX	Hello	11/21/2018 23:12
3	9198XXXXXXXX	Your due amount is Rs.8790.	11/26/2018 21:12
4			
5			

7. API Features

7.1. International Messaging

For international messages, user needs to append 00 before the country code and mobile number.

Mobile Number format: 00(CountryCode)(MobileNumber)

Sample API request:

To send a message to a UK mobile number (country code +44)

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=SendMessage&send_to=0044079xxxxxxxx&msg=Welcome%20to%20SMS%20GupShup%20API&msg_type=TEXT&userid=XXXXXXXXX&auth_scheme=plain&password=password&v=1.1&format=text
```

7.2. Link Tracking

Link Tracking is a powerful feature that delivers meaningful insights about customer behavior based on how, where and when customers interact with unique links embedded in SMS campaigns.

Link Tracking API will automatically detect the long URL in the message text (based on prefixes like http:, https:, www) and shorten the long URL. The Long URL is shortened into a URL which is 28 characters long that can uniquely track the URL click at a recipient level.

Sample API request:

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=SendMessage&send_to=9198xxxxxxxx&msg=It%27s%20working%20www.gupshup.io&msg_type=TEXT&userid=XXXXXXXX&auth_scheme=plain&password=password&v=1.1&format=text&linkTrakingEnabled=TRUE
```

Response message:

```
success | 919XXXXXXXXX| 3288762331521070385-37609775252770110
```

7.3. Zone Info API

Zone Info API allows a user to fetch Number Info i.e (Carrier, Circle) information for a given number via API. This API will help customers to get NumberInfo on a real-time basis.

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?method=GET_MOBILE_NUMBER_INFO&userid=XXXXXXXX&password=password&format=json&mobileNumber=919XXXXXXXXX&v=1.1&auth_scheme=PLAIN
```

Response message:

```
{
"response": {
"id": "3503979002041402733",
"phone": "XXXXXXXXX",
"details": "",
"status": "success"
},
"data": {
"location": "Gajwel, Andhra Pradesh",
"carrier": "Airtel",
"mobileNumber": "919XXXXXXXX"
}
}
```

Note: NumberInfo API does not reflect MNP data for the number i.e. it is possible the number has been ported to a different operator/carrier and circle, but only the original operator / circle is returned.

7.4. Two Factor Authentication (2FA) API

Two Factor Authentication, also known as 2FA, is the dynamic generation of a numeric/alphanumeric/alphabetical code that authenticates the user for a single transaction or session. The code or one-time password (OTP) can be sent to the customer via SMS.

Pre Start Checklist

- User name & password. If you don't have an account you can create one at **https://enterprise.smsgupshup.com**
- For 2FA Account configuration please share below details with support team (**enterprise-support@smsgupshup.com**)

Details to be shared with Gupshup Support team

Parameters	Description
Enterprise Account ID	Numeric Account Id
Otp Expiry in Seconds	Expiry time of OTP for an enterprise user
Otp Retry in Seconds	Number of seconds an enterprise user has to wait before the user can resend OTP
Allowed OTP Verification Attempts	Number of attempts allowed to verify an OTP

a) Generate and send OTP SMS API

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=XXXXXXXX&password=XXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXXX&msg=Your%20OTP%20code%20is%20%25code%25&format=text&otpCodeLength=4&otpCodeType=NUMERIC
```

Response Message:

```
success | 919XXXXXXXXX | 3545912456442574189 | OTP sent
```

b) Verify OTP API

```
https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=XXXXXXX&password=XXXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXX& otp_code=1564
```

Response Message:

```
success | 919XXXXXXXXX | 3545913275288024429 | OTP matched
```

Parameters:

Parameter	Value	Description
userid	Account number provided by Enterprise SMS GupShup	The number must be in pure numeric format with no special characters.
password	UrlEncoded string of UTF-8 characters	Password provided by Gupshup for authentication of user id .The password must be the same as used to log on to the Enterprise SMS GupShup website.
method	TWO_FACTOR_AUTH	Method for performing a specific action.
v	1.1	Default version is 1.1, unless otherwise specified.
Phone_no	Phone no. of the receiver	The number must be in pure numeric format with no special characters
msg	UrlEncoded string of UTF-8 characters	The message that needs to be sent. It can contain alphanumeric & special characters. The message must contain **%code%**
msg_type	Text or Unicode_text	Indicates the type of the message to be sent. Default is “text”.
format	TEXT, XML, JSON	specific format for response message
otpCodeLength	Numeric	This is the length of OTP. It must be a positive integer less than or equals to 10.
otpCodeType	NUMERIC, ALPHABETIC or ALPHANUMERIC	This parameter specifies a type of OTP. 3 values are permitted NUMERIC(only numbers), ALPHABETIC(only alphabets), ALPHANUMERIC(mix of numbers and alphabets)
otp_code	OTP to be verified	The API verifies OTP if given otherwise sends OTP (This is the only difference between Send OTP and Verify OTP)

Error Codes:

Error Code	Error Description
311	This OTP does not exist.
309	You have exceeded maximum number of attempts.
301	OTP token has expired.

310	This OTP is incorrect.
308	You are re-trying too early. Please wait for some time.

a) Encrypted payload for Two Factor Authentication API call

A sample request for "Generate and Send OTP + Verify OTP API" with encrypted data in the API payload looks like this. (See sample code in 9)

https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=xxxxxxxx&encrdata={{Base64 Encoded Encrypted Data}}

Generate and send OTP SMS:

https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=xxxxxxxx&encrdata=VEYcNxNQQcZ0t6VcfxZwzTpeTmBb0UKPYORWA1_VEIaGWrg7zS6aOdUYvmXCpOZbyGs2w9xKA_c43r0OnrL3KHUgUrvlY1_5e-Om8y5zoWyCaWbV2OKFpa5yZQe0GQrm8PZyBJ-YIDgE7qswrrPj005nZZyFiCtDP-50UDnaeg6RTVKV748lkURa1RDcs3lMkh3n5UsaNM5TU_lhkC0yreyuuNu8mlRQmNFDMUMWOfvNvhLL_H4

where

encrdata={{password=XXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXX&msg=Your%20OTP%20code%20is%20%25code%25&format=text&otpCodeLength=4&otpCodeType=NUMERIC}}

Verify OTP API:

https://enterprise.smsgupshup.com/GatewayAPI/rest?userid=xxxxxxxx&encrdata=2qrYFF2dZpT1kLC9iFlfv2mdJgbtUntUuXtRj_Yz5-tVw3JDUNgvGIBHZF-QMtJjNsMQwGEV6WZ3uFQPGdGYeVV43Ah3u82BVGO8naTea_Dbw1WbDd-cwc1uoGN8ip14suC6fEOJE2oz7Gy0

where

encrdata= {{password=XXXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXX&otp_code=1564}}

API response format is the same as unencrypted payload.

Note : Output of encrdata will be different every time as the IV(Initialization Vector) value will be unique for every request.

Please note that **"userid"** parameter is mandatory and its value is to be sent in unencrypted format. All the other parameters must be sent in encrypted format using AES 256 encryption and base64 encoded using the key.

You must use the key generated for your enterprise account to encrypt API parameter values when sending the API request. Once the request is received by Gupshup, the payload is decrypted by Gupshup and sent ahead to the operator. Refer 2.2 section.

Steps for Encryption:

1) Form a querystring using the rest of API parameters and its values
Ex:password=XXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXX&msg=Your%20OTP%20code%20is%20%25code%25&format=text&otpCodeLength=4&otpCodeType=NUMERIC

2) Encrypt the querystring using AES encryption algorithm(256 bit algorithm)
   - Use only GCM mode
   - Length of IV (Initialization Vector) parameter should be 12 bytes. IV value should be unique for every API request/call.
   - Length of authentication tag should be 16 bytes

3) Output of AES Encryption (256 bit) should be encoded using base 64 Urlsafe.

4) Base64 encoded encrypted cipher will be passed as a payload in encrdata parameter.

5) Sample encrypted payload using above steps:
VEYcNxNQQcZ0t6VcfxZwzTpeTmBb0UKPYORWA1_VEIaGWrg7zS6aOdUYvmXCpOZbyGs2w9xKA_c43r0OnrL3KHUgUrvIY1_5e-Om8y5zoWyCaWbV2OKFpa5yZQe0GQrm8PZyBJ-YIDgE7qswrrPj005nZZyFiCtDP-50UDnaeg6RTVKV748lkURa1RDcs3lMkh3n5UsaNM5TU_lhkC0yreyuuNu8mlRQmNFDMUMWOfvNvhLL_H4

The following are the required parameters:.

Parameter	Description
userid	Specify the user id provided by Gupshup. This must be sent in plaintext (unencrypted). The number must be in pure numeric format with no special characters.
encrdata	Encrypted parameter and values to be sent in query parameters / payload.

Parameters and Values to be passed in encrdata. Please note both the parameter and its value should be encrypted.

Parameter	Value	Description
method	TWO_FACTOR_AUTH	Method for performing a specific action i.e. send a OTP SMS Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
v	1.1	Default version is 1.1 and must be passed. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
Phone_no	Phone no. of the receiver	The number must be in pure numeric format with no special characters, and should include country code and mobile number of recipient. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
msg	UrlEncoded string of UTF-8 characters	The message that needs to be sent. It can contain alphanumeric & special characters. The message must contain **%code%.** Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
msg_type	Text or Unicode_text	Indicates the type of the message to be sent. Default is “text”. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.

format	TEXT, XML, JSON	Specific format for API response message. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
otpCodeLength	Numeric	This is the length of OTP. It must be a positive integer less than or equals to 10. Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
otpCodeType	NUMERIC, ALPHABETIC or ALPHANUMERIC	This parameter specifies a type of OTP. 3 values are permitted NUMERIC(only numbers), ALPHABETIC(only alphabets), ALPHANUMERIC(mix of numbers and alphabets). Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.
otp_code	OTP to be verified	The API verifies OTP if given otherwise sends OTP (This is the only difference between Send OTP and Verify OTP Parameter and its value must be sent in encrypted format .Encrypted string should be URL encoded before sending.

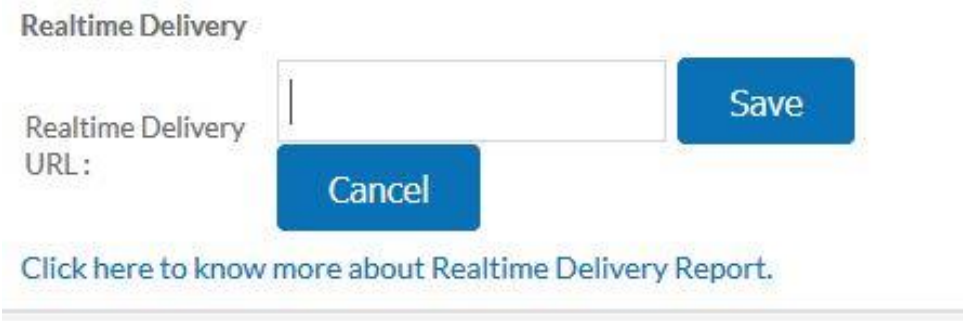
7.5. Realtime Delivery Reports

Let's say you have given **www.example.com/RealTimeDLR/readurl** as the callback URL for a given user account, Gupshup can send a HTTP POST or GET request.

Get Request:

You can set a callback URL for each group and APIs to receive real time delivery reports.

1. Log in on https://enterprise.smsgupshup.com
2. Click on Settings in the menu bar
3. Under Advanced Account Settings you can see Realtime Delivery URL



Let's say you have given **www.example.com/RealTimeDLR/readurl** as the callback URL, the format of the URL called by GupShup is as follows:

```
http://example.com/RealTimeDLR/readurl?externalId=%0&deliveredTS=%1&status=%2&cause=%3&phoneNo=%4
&errCode=%6& noOfFrags=%7&mask=%8
```

Note: Mask parameter is optional and can be forwarded as per request.

Post Request:

HTTP POST request contains a JSON array of one or more events to the call back URL. Maximum 20 events will be sent in a batch to the callback URL.

- Note: Please raise a request to support team **(enterprise-support@smsgupshup.com)** to set post request URL.

The below shows a JSON array with two event objects. You will receive maximum 20 such events in a batch.

```
[response=

[

{

"externalId": 3562707498794989059-328736121207676738,

"eventType": “DELIVERED",

"eventTs": 1526347800000,

"destAddr”: 919892488888,

“srcAddr”: TESTIN,

“cause”: ”SUCCESS”,

“errCode”: 000,

“channel”: “SMS,

“noOfFrags”:1

},

{

"externalId": 3798318073013708082-252169030017029882,

"eventType": “FAILED",

"eventTs": 1526347800000,

"destAddr”: 91989237777,

“srcAddr”: ABCDEF,
```

```
"cause": " UNKNOWN_SUBSCRIBER",

"errCode": 003,

"channel": "SMS",

"noOfFrags":1

}

]]
```

Following is the explanation of various parameters:

- **externalId** - Unique ID for each message.
- **deliveredTS -** Time of delivery of message as LONG number.
- **status** - Final status of the message, possible values are SUCCESS, FAILURE or UNKNOWN.
- **phoneNo** - Phone number of the receiver.
- **cause -** This is the response you will get depending on the status. Various statuses and their explanation are below.
- **Errorcode –** error code assigned to different failure cause
- **No of frags –** one message (fragment) is of 160 characters. No of chars states the total count of fragments.
- **Mask –** Sender id sent with the sms

globalErrorCode	globalErrorName	status	responseStatus
0	SUCCESS	DELIVRD	SUCCESS
1	ABSENT_SUBSCRIBER	UNDELIV	FAIL
2	CALL_BARRED	UNDELIV	FAIL
3	UNKNOWN_SUBSCRIBER	UNDELIV	FAIL
4	SERVICE_DOWN	UNDELIV	FAIL
5	SYSTEM_FAILURE	UNDELIV	FAIL
6	DND_FAIL	FAILED	FAIL
7	BLOCKED	FAILED	FAIL
8	DND_TIMEOUT	FAILED	FAIL
9	OUTSIDE_WORKING_HOURS	FAILED	FAIL
00a	OTHER	UNKNOWN	UNKNOWN
00b	BLOCKED_MASK	FAILED	FAIL
00c	SMSCTIMEDOUT	EXPIRED	SUBMITTED
00d	CANCEL_CAUSEID	FAILED	FAIL
00e	CANCEL_SCHEDULE	FAILED	FAIL

10	DEFERRED	FAILED	FAIL
11	INBOXFULL	UNDELIV	FAIL
12	CONGESTION	UNDELIV	FAIL
13	NO_ACK_FROM_OPERATOR	EXPIRED	SUBMITTED
14	OTHER	FAILED	FAIL
15	OTHER	FAILED	FAIL
16	OTHER	FAILED	FAIL
17	OTHER	FAILED	FAIL
18	OTHER	FAILED	FAIL
19	OTHER	FAILED	FAIL
01a	OTHER	FAILED	FAIL
01b	OTHER	FAILED	FAIL
01c	OTHER	FAILED	FAIL
20	OTHER	FAILED	FAIL
22	BLOCKED_FOR_USER	FAILED	FAIL
23	UNKNOWN_SUBSCRIBER	FAILED	FAIL
24	OTHER	UNDELIV	FAIL
038	MSG_DOES_NOT_MATCH_TEMPLATE	FAIL	FAIL

Cause Explanation:

- ABSENT_SUBSCRIBER: When the service provider fails to reach the member/subscriber, this value is passed
- UNKNOWN_SUBSCRIBER: Unknown/invalid number
- BLOCKED_MASK: Mask is blocked by SMS GupShup
- SYSTEM_FAILURE: Failure due to a problem in the Operator's systems (Originating or Destination Operator)
- CALL_BARRED: Subscriber has some kind of call barring active on the number due to which messages from unknown sources are blocked.
- SERVICE_DOWN: Operator service is temporarily is down.
- OTHER: Message that are sent but could not be delivered for reasons that don't fall under any mentioned category
- DND_FAIL: Number is either in DND or Blocked due to being in DND or a complaint.
- DND_TIMEOUT: Latest DND status is not available in time for the message to be sent. (Max 1 day)
- MSG_DOES_NOT_MATCH_TEMPLATE: Template passed in the message content does not match with dltTemplateId uploaded offline in the system.
- OUTSIDE_WORKING_HOUR: Message sending is outside mentioned working hours

We will call the URL provide by you with above mentioned parameters as we receive delivery reports from the service provider.

8. Setting a Response SMS to Marketing Call-to-Actions

8.1. Keyword Response URL

If you have set up a SMS keyword on a long code (10-digit VMN) or a short code and want to be able to send a dynamic Response SMS to the user, you need to set a Keyword Response URL.

To access keyword response URL:

1. Log in on https://enterprise.smsgupshup.com
2. Click on Keywords in the menu bar
3. Click Create Keyword Group
4. Check Response URL

Whenever a request is received on the defined keyword, it will be forwarded to the given Response URL, such as **www.example.com**. On receiving an incoming message corresponding to a keyword, the GupShup server calls the following URL:

```
http://www.example.com?phonecode=%pcode&keyword=%kw&location=%loc&carrier=%car&content=%con&msisdn=%ph&timestamp=%time
```

The response URL consists details of response such as the sender's phone number, the time when request was received, the keyword on which request was received, the additional message with the request, and so on. Thus, for the keyword Test, phone code 9220092200, and message "Test Nagpur", the server calls the following URL:

```
http://www.example.com?phonecode=9220092200&keyword=Test&location=Mumbai&carrier=Vodafone&content=TestNagpur&msisdn=9812348765&timestamp=13082098000
```

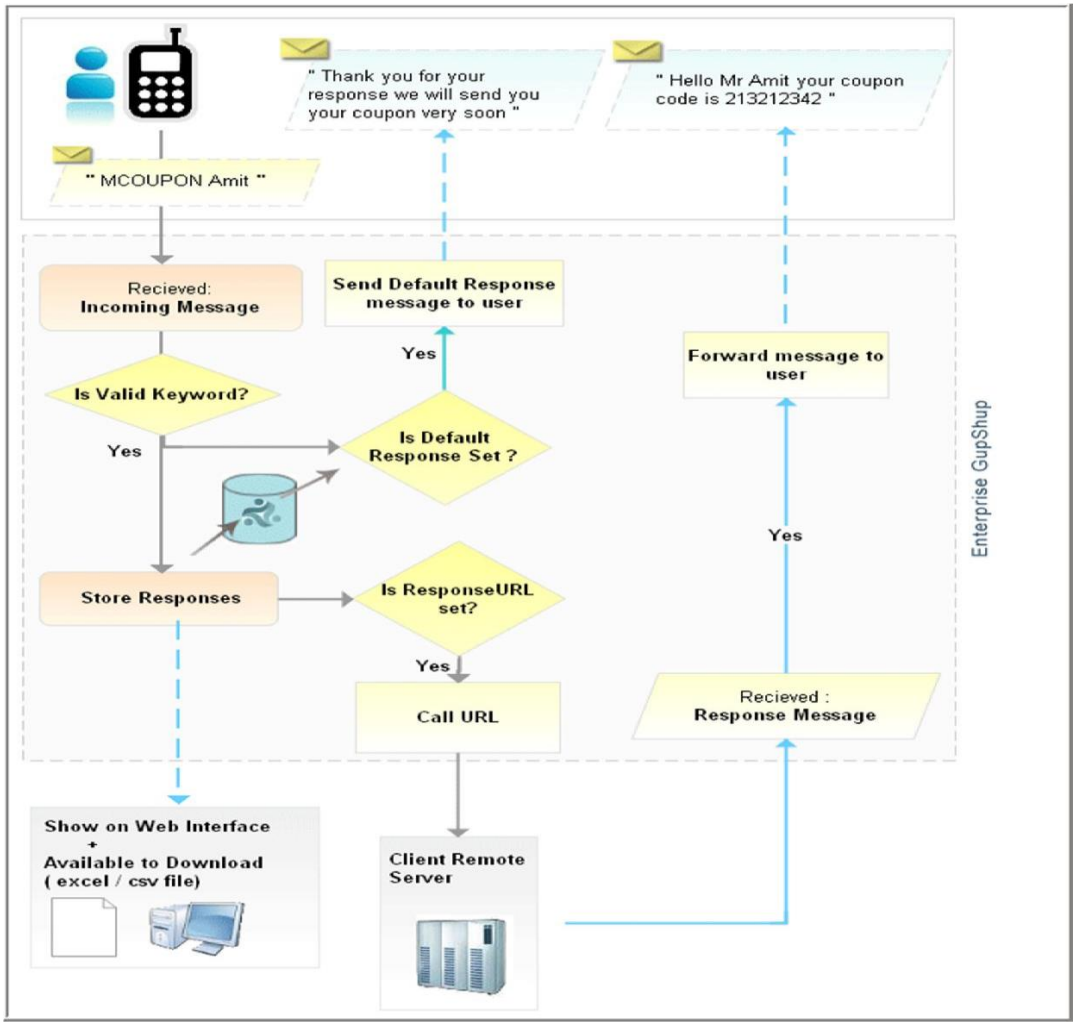
Note: The timestamp shows the epoch time. It has 3 more zeros as it is stored with milliseconds' significance. View more information on Epoch time at http://www.epochconverter.com/

If you wish to generate a response through the Callback URL, you must ensure that the response conforms to a specified XML format. If the remote server returns an invalid XML message or does not return an XML message at all, the first 160 characters of server response are used to compose the message.

XML Example:

```
<response>
 <message type="text">Message 1</message>
 <message type="text">Message 2</message>
</response>
```

The following diagram explains the entire flow of the keyword response process.



8.2 Missed Call Response URL

To enable missed call response url, we request you to provide us with your url so that the same can be configured with your miss call number.

1. A request is received on the defined miscall and it will be forwarded to the given Response URL, such as www.example.com.
2. To respond to the missed call, the GupShup server calls the following URL:

```
http://www.example.com/getresponse.php?msisdn=$msisdn&extension=$extension&causeId=$causeId&hasUserHungUp=$hasUserHungUp&timestamp=$timestamp&carrier=$carrier&location=$location
```

3. The response URL consists of details such as msisdn which is sender's phone number, the time when request was received, the extension on which miscall was received, the additional hasUserHungUp.

Sample URL:

```
http://www.example.com/getresponse.php?msisdn=9199XXXXXXXX&extension=61XXXXXX&causeId=2747975091988275239&hasUserHungUp=false&timestamp=1427095786196&carrier=Airtel&location=Mumbai
```

Parameters are as follows:

- **msisdn** =Sender's mobile number from which request has been received.
- **extension** = Missed call number
- **causeId** = Transaction id (This Id is unique for each request)
- **hasUserHungUp** = if user has hung up at the time of giving missed call, it will be true else it will be false.
- **timestamp =** Time at which missed call was given
- **carrier =** home operator of the mobile number
- **location =** location of the mobile number

9. Sample Java code for AES Encryption

```
import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;

public class AES
{
   private static final int GCM_IV_LENGTH = 12;

   private static final int GCM_TAG_LENGTH = 16;

   private static final String GIVEN_KEY = "QOahfcdo98NLjYJuhP4-VKigx51NkUETsKlIu9uXZFY";

   public static String encrypt(String text) throws Exception
   {
      byte[] bytes = text.getBytes(StandardCharsets.UTF_8);
```

```
        Key secretKey = new SecretKeySpec(Base64.decodeBase64(GIVEN_KEY), "AES");
        byte[] iv = new byte[GCM_IV_LENGTH];
        new SecureRandom().nextBytes(iv);
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        SecretKeySpec keySpec = new SecretKeySpec(secretKey.getEncoded(), "AES");

        GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENGTH * 8, iv);

        cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmParameterSpec);

        byte[] cipherText = cipher.doFinal(bytes);

        byte[] finalArray = new byte[cipherText.length + GCM_IV_LENGTH];

        System.arraycopy(iv, 0, finalArray, 0, GCM_IV_LENGTH);
        System.arraycopy(cipherText, 0, finalArray, GCM_IV_LENGTH, cipherText.length);

        return new String(Base64.encodeBase64URLSafe(finalArray), StandardCharsets.UTF_8);
    }

    public static void main(String[] args) throws Exception
    {
        /* Note that values in query String are URL encoded. */
        String queryString =
"password=XXXXXX&method=TWO_FACTOR_AUTH&v=1.1&phone_no=919XXXXXXXXX&otp_code=1564";

        System.out.println(AES.encrypt(queryString));
    }
}
```

10. Sample Codes

10.1. Sample PHP Code for sending single message

```
<?php
$request =""; //initialise the request variable $param[method]= "sendMessage";
$param[send_to] = "919xxxxxxxxx"; $param[msg] = "Hello"; $param[userid] = "xxxxxxxx"; $param[password] =
"xxxxxxxx"; $param[v] = "1.1";
$param[msg_type] = "TEXT"; //Can be "FLASH"/"UNICODE_TEXT"/"BINARY" $param[auth_scheme] = "PLAIN";
//Have to URL encode the values foreach($param as $key=>$val) {
$request.= $key."=".urlencode($val); //we have to urlencode the values $request.= "&";
//append the ampersand (&) sign after each parameter/value pair
}
$request = substr($request, 0, strlen($request)-1); //remove final (&) sign from the request
$url = "https://enterprise.smsgupshup.com/GatewayAPI/rest?".$request;
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); $curl_scraped_page = curl_exec($ch); curl_close($ch);
echo $curl_scraped_page;
?>
```

10.2. Sample JAVA Code for sending a single message

```
Import  java.io.BufferedReader;
Import  java.io.InputStreamReader;
import  java.io.OutputStreamWriter;
import  java.net.HttpURLConnection;
import  java.net.URL;
import  java.net.URLEncoder;
import  java.util.Date; public class GatewayAPITest {
public static void main(String[] args){ try {
Date mydate = new Date(System.currentTimeMillis());
String data = "";
data += "method=sendMessage";
data += "&userid=xxxxxxxx"; // your loginId data += "&password=" +
URLEncoder.encode("xxxxxx", "UTF-8"); // your password
data += "&msg=" + URLEncoder.encode("GUPSHUP
message" + mydate.toString(), "UTF-8");
data += "&send_to=" +
URLEncoder.encode("9xxxxxxxxx", "UTF-8"); // a valid 10 digit phone no.
data += "&v=1.1" ;
data += "&msg_type=TEXT"; // Can by "FLASH" or
"UNICODE_TEXT" or “BINARY"
data += "&auth_scheme=PLAIN";
URL url = new URL("https://enterprise.smsgupshup.com/GatewayAPI/rest?" + data);
HttpURLConnection conn = (HttpURLConnection)url.openConnection();
conn.setRequestMethod("GET");
conn.setDoOutput(true);
conn.setDoInput(true);
conn.setUseCaches(false);
conn.connect();
BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
String line;
StringBuffer buffer = new StringBuffer(); while ((line = rd.readLine()) != null){
buffer.append(line).append("\n");
}
System.out.println(buffer.toString());
rd.close();
conn.disconnect();
}
catch(Exception e){ e.printStackTrace();
}
}
}
```

10.3. Sample C# Code for sending a single message

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.IO; namespace GupshupAPI{
class Program{
```

```
static void Main(string[] args){ string result = ""; WebRequest request = null;
HttpWebResponse response = null; try{
String sendToPhoneNumber = "919xxxxxxxx"; String userid = "xxxxxxxxx";
String passwd = "xxxxx"; String url =
"https://enterprise.smsgupshup.com/GatewayAPI/rest?method=sendMessage&send_to=" + sendToPhoneNumber +
"&msg=hello&userid=" + userid +"&password=" + passwd + "&v=1.1"&msg_type=TEXT&auth_scheme=PLAIN";
request = WebRequest.Create(url);
//in case u work behind proxy, uncomment the commented code and provide correct details
/*WebProxy proxy = new WebProxy("http://proxy:80/",true); proxy.Credentials = new
NetworkCredential("userId","password", "Domain"); request.Proxy = proxy;*/
// Send the 'HttpWebRequest' and wait for response. response = (HttpWebResponse)request.GetResponse(); Stream
stream = response.GetResponseStream();
Encoding ec = System.Text.Encoding.GetEncoding("utf-8"); StreamReader reader = new
System.IO.StreamReader(stream, ec);
result = reader.ReadToEnd(); Console.WriteLine(result);
reader.Close();
stream.Close();
}
catch (Exception exp){ Console.WriteLine(exp.ToString());
}
finally{
if(response != null) response.Close();
}
}
}
}
```

10.4. Sample Ruby Code

You can access the GupShup HTTP API by using the net/http standard Ruby library. But the plugin provided by SMS GupShup is much easier than the standard one. The plugin is available at http://github.com/nileshtrivedi/gupshup. Install the plugin as

sudo gem sources – a http://gems.github.com sudo gem install nileshtrivedi-gupshup

To override some of the API parameters, pass an options hash as below:

```
gup.send_text_message("hello","919xxxxxxxx", {:mask => "TESTING"}) require 'rubygems'
require 'gupshup'
gup = Gupshup::Enterprise.new("XXXXXX","your_password") gup.send_text_message("hello","919xxxxxxxx")
gup.send_flash_message('sms message text',"919xxxxxxxx")
gup.send_unicode_message("\xE0\xA4\x97\xE0\xA4\xAA\xE0\xA4\xB6\xE0\xA4\xAA","91 9xxxxxxxx")
```

10.5. Sample HTML code for File Upload

```
<html>
<head></head>
<body>
<form name="xlsUploadForm" action="https://enterprise.smsgupshup.com/GatewayAPI/rest" method="post"
enctype="multipart/form-data">
```

<input type="text" name="method" id="method" value="xlsUpload" /> <input type="text" name="userid" id="userid" value=<login-id> /> <input type="text" name="password" id="password" value=<url-encodedpassword> />
<input type="text" name="v" id="version" value="1.1" /> <input type="text" name="auth_scheme" id="auth_scheme" value="PLAIN" />
<input type="file" name="xlsFile" /> <select name="filetype" >
<option value="xls">xls</option> <option value="csv">csv</option> <option value="zip">zip</option>
</select>
<input value="Send Message" type="submit" /> </form>
</body>
</html>

10.6. Sample Java code for File Upload

```
package com.webaroo.gatewayapi.v1;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.NameValuePair;
import org.apache.commons.httpclient.methods.InputStreamRequestEntity;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.multipart.FilePart;
import org.apache.commons.httpclient.methods.multipart.MultipartRequestEntity;
import org.apache.commons.httpclient.methods.multipart.Part;
import org.apache.commons.httpclient.methods.multipart.StringPart;
import com.mysql.jdbc.log.LogFactory;
public class TestClient1 {
public static void main(String[] args) throws HttpException, IOException {
try{
HttpClient client = new HttpClient(); PostMethod method = new
PostMethod("https://enterprise.smsgupshup.com/GatewayAPI/rest"); File f = new File("C:\\xlsUpload1.xls");
Part[] parts ={
new StringPart("method", "xlsUpload"), new StringPart("userid", "XXXXXXX"), new StringPart("password", "XXXXX"),
new StringPart("filetype", "xls"),
new StringPart("v", "1.1"),
new StringPart("auth_scheme", "PLAIN"), new FilePart(f.getName(), f)
};
method.setRequestEntity(new MultipartRequestEntity(parts, method.getParams()));
int statusCode = client.executeMethod(method); System.out.println(statusCode);
}
catch (Exception e){ e.printStackTrace();
}
}
}
```

10.7. Sample Ruby Code for File Upload

```
gup.bulk_file_upload("/home/myname/addressbook.csv")
```

10.8. Sample PHP Code for File Upload

```
<?php
 /**
*  Use a Gupshup Enterprise account to send messages.
*  Supports setting time and mask for individual messages.
*
*  @author Anshul <anshula@webaroo.com>
*/
class EnterpriseSender{
public $id;
public $password;
/**
*  Mask that would appear on receiver's phone. For what can appear here,
*  contact SMS GupShup Support as the mask needs to be set in the account
*  before it can be used here.
*  @var String
*/
public $mask;
private $_url = "https://enterprise.smsgupshup.com/GatewayAPI/rest"; private $_messages = array();
public function __construct($id, $password, $mask = NULL) {
$this->id    =  $id;
$this->password =    $password;
$this->mask  =  $mask;
}
/**
*
*  @param String $msisdn MSISDN of the recipient (will include 91)
*  @param String $content Message content
*  @param String $mask One of the mask as set in the enterprise account
*  @param String $time In any acceptable format for PHP. Time Zone assumed to be
IST.
* @return Boolean */
public function addMsg($msisdn, $content, $mask = NULL, $time = "now"){ $message = new stdClass();
$message->msisdn = $msisdn; $message->content = $content;
$fileName = tempnam(sys_get_temp_dir(), 'EnterpriseUpload').'.csv'; $myFile = fopen($fileName, 'w');
fputs($myFile, ""
.implode('","', array( 'PHONE', 'MESSAGE', 'MASKS', 'TIMESTAMPS'
))
."
."\n"
$message->mask = $mask == NULL ? $this->mask : $mask;
$message->time = new DateTime($time, new DateTimeZone("Asia/Kolkata")); $this->_messages[] = $message;
return TRUE;
}
/**
*  Sends the response using file upload API
*  @return Boolean
*/
public function sendMsg(){ $rows = array();
foreach ($this->_messages as $message) { $rows[] = array(
$message->msisdn, $message->content, $message->mask,
```

```
$message->time->format('Y-m-d H:i:s')
);
}
);
foreach ($rows as $row) {
fputcsv($myFile, $row, ',', '"');
}
fclose($myFile); $params = array();
$params['method'] = 'xlsUpload'; $params['userid'] = $this->id; $params['password'] = $this->password;
$params['filetype'] = 'csv'; $params['auth_scheme'] = 'PLAIN'; $params['v'] = '1.1';
$params['xlsFile'] = '@'.realpath($fileName);
$response = self::post($this->_url, $params, TRUE, CURL_HTTP_VERSION_1_0); unlink($fileName);
return preg_match('/^success/', $response);
}
public static function post($url, $params, $multipart = FALSE, $version= CURL_HTTP_VERSION_NONE){
if(function_exists('curl_init')){ $ch = curl_init();
$timeout = 60; curl_setopt($ch,CURLOPT_URL,$url); curl_setopt($ch,CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_HTTP_VERSION, $version); curl_setopt($ch, CURLOPT_POST, TRUE); if($multipart){
curl_setopt($ch, CURLOPT_POSTFIELDS, $params); }else{
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($params));
}
curl_setopt($ch,CURLOPT_TIMEOUT,$timeout); $data = curl_exec($ch);
if($data === FALSE){
throw new Exception(curl_errno($ch));
}
curl_close($ch); return $data;
}else{
return FALSE;
}
}
}
?>
```

10.9. Sample C# Code for sending a single message

```
var client = new RestClient("https://enterprise.smsgupshup.com/GatewayAPI/rest");
var request = new RestRequest(Method.POST);
request.AddParameter("application/x-www-form-urlencoded",
"method=sendMessage&send_to=919820XXXXXX&msg=This%20is%20sample%20test%20message%20from%20
GupShup&msg_type=TEXT&userid=XXXXXX&auth_scheme=PLAIN&password=XXXXX&format=JSON",
ParameterType.RequestBody);
IRestResponse response = client.Execute(request);
```

10.10. Sample PHP Code sending a single message (Post method)

```
<?php
$curl = curl_init();
$post_fields = array();
$post_fields["method"] = "sendMessage";
$post_fields["send_to"] = "919820XXXXXX";
$post_fields["msg"] = "This is sample test message from GupShup";
$post_fields["msg_type"] = "TEXT";
$post_fields["userid"] = "XXXXXX";
```

```
$post_fields["password"] = "XXXXX";
$post_fields["auth_scheme"] = "PLAIN";
$post_fields["format"] = "JSON";
curl_setopt_array($curl, array(
CURLOPT_URL => "https://enterprise.smsgupshup.com/GatewayAPI/rest",
CURLOPT_RETURNTRANSFER => true,
CURLOPT_ENCODING => "",
CURLOPT_MAXREDIRS => 10,
CURLOPT_TIMEOUT => 30,
CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
CURLOPT_CUSTOMREQUEST => "POST",
CURLOPT_POSTFIELDS => $post_fields
));
$response = curl_exec($curl);
$err = curl_error($curl);
curl_close($curl);
if ($err) {
  echo "cURL Error #:" . $err;
} else {
  echo $response;
}
```

10.11. Sample Curl command for sending a single message

```
curl --request POST \
--url https://enterprise.smsgupshup.com/GatewayAPI/rest \
--data
'method=sendMessage&send_to=919820XXXXXX&msg=This%20is%20sample%20test%20message%20from%20G
upShup&msg_type=TEXT&userid=XXXXXX&auth_scheme=PLAIN&password=XXXXX&format=JSON'
```

10.12. Sample Ruby Code for sending a single message

```
require 'uri'
require 'net/http'
url = URI("https://enterprise.smsgupshup.com/GatewayAPI/rest")
http = Net::HTTP.new(url.host, url.port)
request = Net::HTTP::Post.new(url)
request.body =
"method=sendMessage&send_to=919820XXXXXX&msg=This%20is%20sample%20test%20message%20from%20
GupShup&msg_type=TEXT&userid=XXXXXX&auth_scheme=PLAIN&password=XXXXX&format=JSON"
response = http.request(request)
puts response.read_body
```

10.13. Sample NodeJS Code for sending a single message

```
var request = require("request");
var options = { method: 'POST',
url: 'https://enterprise.smsgupshup.com/GatewayAPI/rest',
form:
{ method: 'sendMessage',
send_to: '919820XXXXXX',
msg: 'This is sample test message from GupShup',
msg_type: 'TEXT',
userid: 'XXXXXX',
```

```
auth_scheme: 'PLAIN',
password: 'XXXXX',
format: 'JSON' } };
request(options, function (error, response, body) {
if (error) throw new Error(error);
console.log(body);
});
```

10.14. Sample Python Code for sending a single message

```
import requests
url = "https://enterprise.smsgupshup.com/GatewayAPI/rest"
payload =
"method=sendMessage&send_to=919820XXXXXX&msg=This%20is%20sample%20test%20message%20from%20
GupShup&msg_type=TEXT&userid=XXXXXX&auth_scheme=PLAIN&password=XXXXX&format=JSON"
response = requests.request("POST", url, data=payload)
print(response.text)
```