GatewayAPI.com Documentation

Release 1.0

OnlineCity

September 10, 2020

Our documentation is broken into a few parts based on protocol. All new customers are encouraged to use the new *REST API*. However we also provide a few legacy APIs based on SOAP and HTTP POST/GET, which are still maintained although deprecated.

If you are reading this online at https://gatewayapi.com/docs/ and require any assistence, please use the live chat feature at the bottom right of your screen.

CHAPTER

ONE

REST API

This is our new API which is available for GatewayAPI.com and based predominately on HTTP POST calls and JSON.

To use this API you must be a customer on the GatewayAPI.com platform, and run on "modern" SSL/TLS software (support SHA-2, ie. OpenSSL 0.9.8+, NSS 3.11+, Win2k8/Vista+, Java 7+).

1.1 Authentication

Use either *API Token*, *HTTP Basic Authentication* or *OAuth*. We encourage the use of *API Token* to most users.

1.1.1 API Token

Your API keys are expressed as a key+secret combo, and as an API token. The key+secret is used for *OAuth* while the token can be used for a simpler scheme with better compatibility.

You can send the token as the username via *HTTP Basic Authentication*, or you may send the token as a query argument or form value. This means that if you can send a HTTP request, you can use Token Authentication.

Example of JSON body and API token as a query argument.

```
POST /rest/mtsms?token=Go-Create-an-API-token HTTP/1.1
Host: gatewayapi.com
Accept: application/json, text/javascript
Content-Type: application/json

{ "message": "Hello World", "recipients": [ { "msisdn": 4512345678 } ] }
```

1.1.2 OAuth

The OAuth specification exists in two versions; 1 and 2, each having little to do with the other. OAuth 1.0a is suitable for API usage without a user present and provides protection against replay attacks.

No user interaction is required for the authentication, so this part is skipped from OAuth. There are only two parties: consumer and service provider. Thus this is a special two-legged variant of OAuth 1.0a. The signing process is identical to the normal three-legged OAuth, but we simply leave the token and secret as empty strings.

The oauth parameters can be sent as the OAuth Authorization header or as URL params. Your framework should take care of all the details for you, however if you are fiddling with it yourself, it's important that the nonce is unique and the timestamp is correct.

Header example

```
POST /rest/mtsms HTTP/1.1
Host: gatewayapi.com
Authorization: OAuth oauth_consumer_key="Create-an-API-Key",
  oauth_nonce="128817750813820944501450124113",
  oauth_timestamp="1450124113",
  oauth_version="1.0",
  oauth_signature_method="HMAC-SHA1",
  oauth_signature="t9w86dddubh4XofnnPgH%2BY6v5TU%3D"
Accept: application/json, text/javascript
Content-Type: application/json

{ "message": "Hello World", "recipients": [ { "msisdn": 4512345678 } ] }
```

URL Params

```
POST /rest/mtsms?oauth_consumer_key=CreateAKey&oauth_nonce=12345&oauth_signature_method=HMAC-SHA1&oau
Host: gatewayapi.com
Accept: application/json, text/javascript
Content-Type: application/json

{ "message": "Hello World", "recipients": [ { "msisdn": 4512345678 } ] }
```

1.1.3 HTTP Basic Authentication

HTTP Basic auth must only be used with HTTPS connections (SSL encrypted), since the credentials are sent as base64 encoded plaintext.

Support is built-in on most networking frameworks, but it's also simple to do yourself. The credentials are sent as "Authorization: Basic `basic-cookie`". basic-cookie is `username ":" password` which is then base64 encoded.

You can use basic auth with credentials (deprecated: ie. username + password), or with an API Token. The API Token is sent as the username with password left empty. You can find and create a set of credentials under "Settings", "Credentials (deprecated)", the API Token is available under API Keys.

```
POST /rest/mtsms HTTP/1.1
Host: gatewayapi.com
Authorization: Basic Zm9vOmJhcg==
Accept: application/json, text/javascript
Content-Type: application/json

{ "message": "Hello World", "recipients": [ { "msisdn": 4512345678 } ] }
```

If you can't use/specify an Authorization header, you can provide the username and password as form or query arguments. The username is sent as 'user', and the password as 'password'.

1.2 Sending SMS'es

Also known as *MT SMS*, short for Mobile Terminated SMS, is when you want to deliver a SMS to a users mobile device.

1.2.1 Message Filtering

Some messages contain links that due to phishing attacks and generally unwanted spam cannot be accepted. Each account has a whitelist of links that are allowed, unique to that account, and approved by our staff. Any links found in the messages are checked against the whitelist, using the following method:

- A bare domain (such as `gatewayapi.com`) allows all links pointing to that domain.
- A specific link (such as `gatewayapi.com/docs`) only allows exactly that link to be allowed through the whitelist check.

Some certain accounts are marked as especially trusted and are excempt from having their messages checked.

You can submit new links, as well as check the current whitelist on the dashboard under Settings.

To learn more about the our efforts to stopping malicious messages, go read the blog post about stopping illegal sms trafic.

1.2.2 Basic usage

Also see *Advanced usage* for a complete example of all features.

`POST /rest/mtsms`

The root element can be either a dict with a single SMS or a list of SMS'es. You can send data in JSON format, or even as http form data or query args.

Request JSON Object

- **`class`** (*`string`*) – Default "standard". The message class to use for this request. If specified it must be the same for all messages in the request.
- **`message`** (*`string`*) – The content of the SMS, *always* specified in UTF-8 encoding, which we will transcode depending on the "encoding" field. The default is the usual *GSM 03.38* encoding. *required*
- **`sender`** (*`string`*) – Up to 11 alphanumeric characters, or 15 digits, that will be shown as the sender of the SMS. See *SMS Sender*
- **`userref`** (*`string`*) – A transparent string reference, you may set to keep track of the message in your own systems. Returned to you when you receive a *Delivery Status Notification*.
- **`callback_url`** (*`string`*) – If specified send status notifications to this URL, else use the default webhook.
- **`recipients`** (*`array`*) – Array of recipients, described below. The number of recipients in a single message is limited to 10.000. *required*

Request JSON Array of Objects

- **`msisdn`** (*`string`*) – *MSISDN* aka the full mobile phone number of the recipient. *required*

Response JSON Object

- **`ids`** (*`array`*) – If successful you receive a object containing a list of message ids.
- **`usage`** (*`dictionary`*) – If successful you receive a usage dictionary with usage information for you request.

Status Codes

- 200 OK – Returns a dict with an array of message IDs and a dictionary with usage information on success

- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

```
POST /rest/mtsms HTTP/1.1
Host: gatewayapi.com
Authorization: OAuth oauth_consumer_key="Create-an-API-Key",
  oauth_nonce="128817750813820944501450124113",
  oauth_timestamp="1450124113",
  oauth_version="1.0",
  oauth_signature_method="HMAC-SHA1",
  oauth_signature="t9w86dddubh4XofnnPgH%2BY6v5TU%3D"
Accept: application/json, text/javascript
Content-Type: application/json

{
    "message": "Hello World",
    "recipients": [
        { "msisdn": 4512345678 },
        { "msisdn": 4587654321 }
    ]
}
```

```
POST /rest/mtsms?token=Go-Create-an-API-token HTTP/1.1
Host: gatewayapi.com
Content-Type: application/x-www-form-urlencoded

message=Hello World&recipients.0.msisdn=4512345678&recipients.1.msisdn=4587654321
```

The two examples above do the exact same thing, but with different styles of input. You can even send it all using just a GET url

GET /rest/mtsms
You can use GET requests to send your SMS'es as well. Just pass the parameters you need as query parameters.

https://gatewayapi.com/rest/mtsms?token=Go-Create-an-API-token&message=Hello+World&recipients.0.msisdn=4512345678&

1.2.3 Connection limit

Our API has a limit of 40 open connections per IP address, if you have more than 40 open connections our web server will reject your requests. If you need to send lots of smses consider bulking your requests with multiple recipients, you can use tags and tagvalues to add unique data per recipient, bulking your requests will also increase your delivery speed compared to making a single request per recipient.

1.2.4 Code Examples

Since sending SMS'es is a central part of most customers' use cases we'll list the code examples in full. These examples are also available preconfigured with your own API keys on the dashboard at https://gatewayapi.com/app/.

Since the OAuth bits are the same for all API calls, these examples can easily be modified for other calls.

Python

For this example you'll need the excellent Requests-OAuthlib. If you are using pip, simply do `pip install requests_oauthlib`.

```
from requests_oauthlib import OAuth1Session
key = 'Create-an-API-Key'
secret = 'GoGenerateAnApiKeyAndSecret'
gwapi = OAuth1Session(key, client_secret=secret)
req = {
    'recipients': [{'msisdn': 4512345678}],
    'message': 'Hello World',
    'sender': 'ExampleSMS',
}
res = gwapi.post('https://gatewayapi.com/rest/mtsms', json=req)
res.raise_for_status()
```

PHP

For a really simple integration, the following will suffice:

```
<?php
// Query args
$query = http_build_query(array(
    'token' => 'Go-Create-an-API-token',
    'sender' => 'ExampleSMS',
    'message' => 'Hello World',
    'recipients.0.msisdn' => 4512345678,
));
// Send it
$result = file_get_contents('https://gatewayapi.com/rest/mtsms?' . $query);
// Get SMS ids (optional)
print_r(json_decode($result)->ids);
```

The above example is good for trying to get a quick sms through to your number as a test, but is not recommened for production use, you should consider the below examples, using composer or cURL.

```
<?php
$recipients = ['4527128516', '4561856583'];
$url = "https://gatewayapi.com/rest/mtsms";
$api_token = "Go-Create-An-API-token";
$json = [
   'sender' => 'ExampleSMS',
   'message' => 'Hello world',
   'recipients' => [],
];
foreach ($recipients as $msisdn) {
   $json['recipients'][] = ['msisdn' => $msisdn];
}
$ch = curl_init();
curl_setopt($ch,CURLOPT_URL, $url);
curl_setopt($ch,CURLOPT_HTTPHEADER, array("Content-Type: application/json"));
curl_setopt($ch,CURLOPT_USERPWD, $api_token.":");
curl_setopt($ch,CURLOPT_POSTFIELDS, json_encode($json));
curl_setopt($ch,CURLOPT_RETURNTRANSFER, true);
$result = curl_exec($ch);
curl_close($ch);
print($result); // print result as json string
```

```
$json = json_decode($result); // convert to object
print_r($json->ids); // print the array with ids
print_r($json->usage->total_cost); // print total cost from 'usage' object
```

However if you are using composer, then you'll want to use our Guzzle example. Install the deps with `composer require "guzzlehttp/oauth-subscriber 0.3.*"`.

```
<?php
require_once 'vendor/autoload.php';
$stack = \GuzzleHttp\HandlerStack::create();
$oauth_middleware = new \GuzzleHttp\Subscriber\Oauth\Oauth1([
    'consumer_key'    => 'Create-an-API-Key',
    'consumer_secret' => 'GoGenerateAnApiKeyAndSecret',
    'token'           => '',
    'token_secret'    => ''
]);
$stack->push($oauth_middleware);
$client = new \GuzzleHttp\Client([
    'base_uri' => 'https://gatewayapi.com/rest/',
    'handler'  => $stack,
    'auth'     => 'oauth'
]);

$req = [
    'sender'     => 'ExampleSMS',
    'recipients' => [['msisdn' => 4512345678]],
    'message'    => 'Hello World',
];
$client->post('mtsms', ['json' => $req]);
```

It's also possible to do oauth signing using only the built-in PHP functions. Although it's not going to look as nice as guzzle, this one won't require composer or any other dependencies.

```
<?php
// Variables for OAuth 1.0a Signature
$nonce = rawurlencode(uniqid());
$ts = rawurlencode(time());
$key = rawurlencode('Create-an-API-Key');
$secret = rawurlencode('GoGenerateAnApiKeyAndSecret');
$uri = 'https://gatewayapi.com/rest/mtsms';
$method = 'POST';

// OAuth 1.0a - Signature Base String
$oauth_params = array(
    'oauth_consumer_key' => $key,
    'oauth_nonce' => $nonce,
    'oauth_signature_method' => 'HMAC-SHA1',
    'oauth_timestamp' => $ts,
    'oauth_version' => '1.0',
);
$sbs = $method . '&' . rawurlencode($uri) . '&';
$it = new ArrayIterator($oauth_params);
while ($it->valid()) {
    $sbs .= $it->key() . '%3D' . $it->current();$it->next();
    if ($it->valid()) $sbs .= '%26';
}

// OAuth 1.0a - Sign SBS with secret
$sig = base64_encode(hash_hmac('sha1', $sbs, $secret . '&', true));
```

```php
$oauth_params['oauth_signature'] = rawurlencode($sig);

// Construct Authorization header
$it = new ArrayIterator($oauth_params);
$auth = 'Authorization: OAuth ';
while ($it->valid()) {
    $auth .= $it->key() . '="' . $it->current() . '"';$it->next();
    if ($it->valid()) $auth .= ', ';
}

// Request body
$req = array(
    'recipients' => array(array('msisdn' => 4512345678)),
    'message' => 'Hello World',
    'sender' => 'ExampleSMS',
);


// Send request with cURL
$c = curl_init($uri);
curl_setopt($c, CURLOPT_HTTPHEADER, array(
    $auth,
    'Content-Type: application/json'
));
curl_setopt($c, CURLOPT_POSTFIELDS, json_encode($req));
curl_exec($c);
```

cURL

API Tokens and the support for form data is a great match for cURL integration, since sending an SMS becomes as easy as:

```
curl -v https://gatewayapi.com/rest/mtsms \
  -u Go-Create-an-API-token: \
  -d sender="ExampleSMS" \
  -d message="Hello World" \
  -d recipients.0.msisdn=4512345678
```

C#

This example uses RestSharp. and NewtonSoft. If you're using the NuGet Package Manager Console: `Install-Package RestSharp`, `Install-Package Newtonsoft.Json`.

```
var client = new RestSharp.RestClient("https://gatewayapi.com/rest");
var apiToken = "GoGenerateAnApiToken";

client.Authenticator = new RestSharp.Authenticators
   .HttpBasicAuthenticator(apiToken, "");
var request = new RestSharp.RestRequest("mtsms", RestSharp.Method.POST);
request.AddJsonBody(new
{
   sender = "ExampleSMS",
   message = "Hello World",
   recipients = new[] { new { msisdn = 4512345678 } }
});
var response = client.Execute(request);
```

```
// On 200 OK, parse the list of SMS IDs else print error
if ((int)response.StatusCode == 200)
{
   var res = Newtonsoft.Json.Linq.JObject.Parse(response.Content);
   foreach (var i in res["ids"])
   {
      Console.WriteLine(i);
   }
}
else if (response.ResponseStatus == RestSharp.ResponseStatus.Completed)
{
   Console.WriteLine(response.Content);
}
else
{
   Console.WriteLine(response.ErrorMessage);
}
```

Ruby

Install the deps with `gem install oauth`.

```
# encoding: UTF-8
require 'oauth'
require 'json'

consumer = OAuth::Consumer.new(
  'Create-an-API-Key',
  'GoGenerateAnApiKeyAndSecret',
  :site => 'https://gatewayapi.com/rest',
  :scheme => :header
)
access = OAuth::AccessToken.new consumer
body = JSON.generate({
  'recipients' => [{'msisdn' => 4512345678}],
  'message' => 'Hello World',
  'sender' => 'ExampleSMS',
})
response = access.post('/mtsms', body, {'Content-Type'=>'application/json'})
puts response.body
```

Node.js

Install the deps with `npm install request`.

```
var request = require('request');
request.post({
  url: 'https://gatewayapi.com/rest/mtsms',
  oauth: {
    consumer_key: 'Create-an-API-Key',
    consumer_secret: 'GoGenerateAnApiKeyAndSecret',
  },
  json: true,
  body: {
    sender: 'ExampleSMS',
```

```
    message: 'Hello World',
    recipients: [{msisdn: 4512345678}],
  },
}, function (err, r, body) {
  console.log(err ? err : body);
});
```

Java

Using nothing but standard edition java, you can send a SMS like so.

```
import java.io.DataOutputStream;
import java.net.URL;
import java.net.URLEncoder;
import javax.net.ssl.HttpsURLConnection;

public class HelloWorld {
  public static void main(String[] args) throws Exception {
    URL url = new URL("https://gatewayapi.com/rest/mtsms");
    HttpsURLConnection con = (HttpsURLConnection) url.openConnection();
    con.setDoOutput(true);

    DataOutputStream wr = new DataOutputStream(con.getOutputStream());
    wr.writeBytes(
      "token=Go-Create-an-API-token"
      + "&sender=" + URLEncoder.encode("ExampleSMS", "UTF-8")
      + "&message=" + URLEncoder.encode("Hello World", "UTF-8")
      + "&recipients.0.msisdn=4512345678"
    );
    wr.close();

    int responseCode = con.getResponseCode();
    System.out.println("Got response " + responseCode);
  }
}
```

However we expect many of you are using OkHttp or similar, which gives you a nice API. Combine this with your favorite JSON package. Install the dependencies with.

```
compile 'com.squareup.okhttp3:okhttp:3.4.1'
compile 'se.akerfeldt:okhttp-signpost:1.1.0'
compile 'org.json:json:20160810'
```

```
final String key = "Create-an-API-Key";
final String secret = "GoGenerateAnApiKeyAndSecret";

OkHttpOAuthConsumer consumer = new OkHttpOAuthConsumer(key, secret);
OkHttpClient client = new OkHttpClient.Builder()
        .addInterceptor(new SigningInterceptor(consumer))
        .build();
JSONObject json = new JSONObject();
json.put("sender", "ExampleSMS");
json.put("message", "Hello World");
json.put("recipients", (new JSONArray()).put(
        (new JSONObject()).put("msisdn", 4512345678L)
));
```

```
RequestBody body = RequestBody.create(
        MediaType.parse("application/json; charset=utf-8"), json.toString());
Request signedRequest = (Request) consumer.sign(
        new Request.Builder()
                .url("https://gatewayapi.com/rest/mtsms")
                .post(body)
                .build()).unwrap();

try (Response response = client.newCall(signedRequest).execute()) {
    System.out.println(response.body().string());
}
```

Httpie

For quick testing with a pretty jsonified response in your terminal you can use *Httpie*. It can be done simply using your token as follows.

```
http --auth=GoGenerateAnApiToken: \
https://gatewayapi.com/rest/mtsms \
sender='ExampleSMS' \
message='Hello world' \
recipients:='[{"msisdn": 4512345678}]'
```

Or you can install the httpie-oauth library and use your API key and secret.

```
# install httpie oauth lib, with pip install httpie-oauth
http --auth-type=oauth1 \
--auth="Create-an-API-Key:" \
"GoGenerateAnApiKeyAndSecret" \
https://gatewayapi.com/rest/mtsms \
sender='ExampleSMS' \
message='Hello world' \
recipients:='[{"msisdn": 4512345678}]'
```

1.2.5 Advanced usage

`POST /rest/mtsms`

The root element can be either a dict with a single SMS or a list of SMS'es.

Request JSON Object

- **`class`** (*`string`*) – Default 'standard'. The message class, 'standard', 'premium' or 'secret' to use for this request. If specified it must be the same for all messages in the request. The secret class can be used to blur the message content you send, used for very sensitive data. It is priced as premium and uses the same routes, which ensures end to end encryption of your messages. Access to the secret class will be very strictly controlled.
- **`message`** (*`string`*) – The content of the SMS, *always* specified in UTF-8 encoding, which we will transcode depending on the "encoding" field. The default is the usual *GSM 03.38* encoding. Required unless payload is specified.
- **`sender`** (*`string`*) – Up to 11 alphanumeric characters, or 15 digits, that will be shown as the sender of the SMS. See *SMS Sender*
- **`sendtime`** (*`integer`*) – Unix timestamp (seconds since epoch) to schedule message sending at certain time.

- **tags** (*array*) – A list of string tags, which will be replaced with the tag values for each recipient.
- **userref** (*string*) – A transparent string reference, you may set to keep track of the message in your own systems. Returned to you when you receive a *Delivery Status Notification*.
- **priority** (*string*) – Default 'NORMAL'. One of 'BULK', 'NORMAL', 'URGENT' and 'VERY_URGENT'. Urgent and Very Urgent normally require the use of premium message class.
- **validity_period** (*integer*) – Specified in seconds. If message is not delivered within this timespan, it will expire and you will get a notification. The minimum value is 60. Every value under 60 will be set to 60.
- **encoding** (*string*) – Encoding to use when sending the message. Defaults to 'UTF8', which means we will use *GSM 03.38*. Use *UCS2* to send a unicode message.
- **destaddr** (*string*) – One of 'DISPLAY', 'MOBILE', 'SIMCARD', 'EXTUNIT'. Use display to do "flash sms", a message displayed on screen immediately but not saved in the normal message inbox on the mobile device.
- **payload** (*string*) – If you are sending a binary SMS, ie. a SMS you have encoded yourself or with speciel content for feature phones (non-smartphones). You may specify a payload, encoded as Base64. If specified, message must not be set and tags are unavailable.
- **udh** (*string*) – UDH to enable additional functionality for binary SMS, encoded as Base64.
- **callback_url** (*string*) – If specified send status notifications to this URL, else use the default webhook.
- **label** (*string*) – A label added to each sent message, can be used to uniquely identify a customer or company that you sent the message on behalf of, to help with invoicing your customers. If specied it must be the same for all messages in the request.
- **max_parts** (*int*) – A number between 1 and 255 used to limit the number of smses a single message will send. Can be used if you send smses from systems that generates messages that you can't control, this way you can ensure that you don't send very long smses. You will not be charged for more than the amount specified here. Can't be used with Tags or BINARY smses.
- **extra_details** (*string*) – To get more details about the number of parts sent to each recipient set this to 'recipients_usage'. See example response below.
- **recipients** (*array*) – Array of recipients, described below. The number of recipients in a single message is limited to 10.000. *required*

Request JSON Array of Objects

- **msisdn** (*string*) – *MSISDN* aka the full mobile phone number of the recipient. Duplicates are not allowed in the same message. *required*
- **charge** (*object*) – Charge data. See *Overcharged SMSes*.
- **tagvalues** (*array*) – A list of string values corresponding to the tags in message. The order and amount of tag values must exactly match the tags.

Response JSON Object

- **ids** (*array*) – If successful you receive a object containing a list of message ids.

Status Codes

- 200 OK – Returns a dict with message IDs on success
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Fully fledged request

This is a bit of contrived example since, but it shows most of the possible fields in the API like multiple recipients to the same message using tags and various extra options.

```
POST /rest/mtsms HTTP/1.1
Host: gatewayapi.com
Authorization: OAuth oauth_consumer_key="Create-an-API-Key",
  oauth_nonce="128817750813820944501450124113",
  oauth_timestamp="1450124113",
  oauth_version="1.0",
  oauth_signature_method="HMAC-SHA1",
  oauth_signature="t9w86dddubh4XofnnPgH%2BY6v5TU%3D"
Accept: application/json, text/javascript
Content-Type: application/json

  {
      "class": "premium",
      "message": "Hello World, regards %Firstname%, %Lastname%",
      "label": "Deathstar",
      "recipients": [
          {
              "msisdn": 1514654321,
              "tagvalues": [
                  "Vader",
                  "Darth"
              ]
          },
          {
              "msisdn": 1514654322,
              "tagvalues": [
                  "Maul",
                  "Darth"
              ]
          }
      ],
      "sender": "Empire",
      "sendtime": 915148800,
      "tags": [
          "%Lastname%",
          "%Firstname%"
      ],
      "userref": "1234",
      "priority": "VERY_URGENT",
      "validity_period": 86400,
      "encoding": "UTF8",
      "destaddr": "MOBILE",
      "callback_url": "https://example.com/cb?foo=bar"
```

```
    }
```

Example response

If the request succeed, the internal message identifier are returned to the caller like this:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "ids": [
        421332671
    ],
    "usage": {
        "countries": {
            "DK": 2
        },
        "currency": "DKK",
        "total_cost": 0.30
    }
}
```

Please note that this response is subject to change and will continually, be updated to contain more useful data.

If the request fails, the response will look like the example below:

```
HTTP/1.1 403 FORBIDDEN
Content-Type: application/json

{
  "code": "0x0213",
  "incident_uuid": "d8127429-fa0c-4316-b1f2-e610c3958f43",
  "message": "Unauthorized IP-address: %1",
  "variables": [
    "1.2.3.4"
  ]
}
```

`code` and `variables` are left out of the response if they are empty. For a complete list of the various codes see *API Errors*.

If the `extra_details` option is set to `recipients_usage` the response will look like:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "ids": [
        421332671, 4421332672
    ],
    "usage": {
        "countries": {
            "DK": 3
        },
        "currency": "DKK",
        "total_cost": 0.36
    },
    "details": {
       "messages": [
          {
```

```
                "id": 421332671,
                "recipients": [
                    {
                        "country": "DK",
                        "msisdn": 1514654321,
                        "parts": 1
                    },
                    {
                        "country": "DK",
                        "msisdn": 1514654322,
                        "parts": 1
                    }
                ]
            },
            {
                "id": 421332672,
                "recipients": [
                    {
                        "country": "DK",
                        "msisdn": 4512345678,
                        "parts": 1
                    }
                ]
            }
        ]
    }
}
```

1.2.6 Overcharged SMSes

Overcharged SMSes are only possible in Denmark for the moment. Contact our support if you are interested in using this feature.

An overcharged SMS is sent like a normal SMS, with a few extra parameters and restrictions.

Only one recipient per message is allowed. Messageclass *charge* must be used. Sendername is limited to `1204` or your own shortcode.

The `charge` object in recipient takes the following. See *Advanced usage* for full list of parameters.

POST /rest/mtsms

Request JSON Object

- **amount** (*float*) – The amount to be charged including VAT. *required*
- **currency** (*string*) – Currency used in ISO 4217. *required*
- **code** (*string*) – Product code. P01-P10. *required*
- **description** (*string*) – Description of the charge to appear on the phonebill for the MSISDN owner. *required*
- **category** (*string*) – Service category category. SC00-SC34. *required*

Full example

```
POST /rest/mtsms HTTP/1.1
Host: gatewayapi.com
Authorization: OAuth oauth_consumer_key="Create-an-API-Key",
  oauth_nonce="128817750813820944501450124113",
```

```
      oauth_timestamp="1450124113",
      oauth_version="1.0",
      oauth_signature_method="HMAC-SHA1",
      oauth_signature="t9w86dddubh4XofnnPgH%2BY6v5TU%3D"
    Accept: application/json, text/javascript
    Content-Type: application/json

    [
        {
            "message": "Thank you for your purchase",
            "class": "charge",
            "sender": 1204,
            "recipients": [
                {
                  "msisdn": 4512345678,
                  "charge": {
                    "amount": 50.75,
                    "currency": "DKK",
                    "code": "P01",
                    "category": "SC29",
                    "description": "Nokia tune",
                  }
                }
            ]
        }
    ]
```

See *Charge status* for info about status reports and *Refund charged sms* for info about refunding a charged sms.

1.3 Get SMS and SMS status

You can use http get requests to retrieve a message based on its id, this will give you back the original message that you send, including delivery status and error codes if something went wrong. You get the ID when you send your message, so remember to keep track of the id, if you need to retrieve a message. This is only possible after the message has been sent, since only then is it transferred to long term storage.

Please note we strongly recommend using *Webhooks* to get the status pushed to you when it changes, rather than poll for changes. We do not provide the same guarantees for this particular API endpoint as the others, since it runs on the reporting infrastructure.

GET /rest/mtsms/<message_id>

Parameters

- **id** (*integer*) – A SMS ID, as returned when sending the SMS

Status Codes

- 200 OK – Returns a dict that represents the SMS on success
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 404 Not Found – SMS is not found, or is not yet transferred to datastore.
- 422 Unprocessable Entity – Invalid json request body

- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Example response

```
HTTP/1.1 200 OK
Content-Length: 729
Content-Type: application/json
Date: Thu, 1 Jan 1970 00:00:00 GMT
Server: Werkzeug/0.11.15 Python/3.6.0

 [
     {
         "class": "standard",
         "message": "Hello World, regards %Firstname, --Lastname--",
         "payload": null,
         "id": 1,
         "label": "Deathstar inc.",
         "recipients": [
             {
                 "country": "DK",
                 "csms": 1,
                 "dsnerror": null,
                 "dsnerrorcode": null,
                 "dsnstatus": "DELIVERED",
                 "dsntime": 1498040129.0,
                 "mcc": 302,
                 "mnc": 720,
                 "msisdn": 1514654321,
                 "senttime": 1498040069.0,
                 "tagvalues": [
                     "Vader",
                     "Darth"
                 ]
             }
             {

                "country": "DK",
                "csms": 1,
                "dsnerror": null,
                "dsnerrorcode": null,
                "dsnstatus": "DELIVERED",
                "dsntime": 1498040129.0,
                "mcc": 238,
                "mnc": 1,
                "msisdn": 4512345678,
                "senttime": 1498040069.0,
                "tagvalues": null
            },

         ],
         "sender": "Test Sender",
         "sendtime": 915148800,
         "tags": [
             "--Lastname--",
             "%Firstname"
         ],
         "userref": "1234",
         "priority": "NORMAL",
```

```
            "validity_period": 86400,
            "encoding": "UTF8",
            "destaddr": "MOBILE",
            "udh": null,
            "callback_url": "https://example.com/cb?foo=bar"
        }
    ]
```

1.4 Delete scheduled SMS

If you send smses using the sendtime parameter to schedule the sms for a specific time. You can send us DELETE requests for the id of the schudeled message and remove it from, the queue.

DELETE /rest/mtsms/{id}

You can only delete smses that have been added to the queue using the sendtime parameter.

Parameters

- **id** (*integer*) – A SMS ID, as returned when sending the SMS

Status Codes

- 204 No Content –
- 410 Gone – Message is already gone, either deleted or has been sent.
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 404 Not Found – SMS is not found, or is not yet transferred to datastore.
- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

1.5 Check account balance

You can use the /me endpoint to check your account balance, and what currency your account is set too.

GET /rest/me

Response JSON Object

- **credit** (*float*) – The remaining credit.
- **currency** (*string*) – The currency of your credit.
- **account id** (*integer*) – The id of your account.

Status Codes

- 200 OK – Returns a dict with an array containing information on your account.
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address

- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json

 {
    "credit": 1234.56,
    "currency": "DKK",
    "id": 1
 }
```

1.6 Get prices

You can use the prices endpoint to get our price as csv, xlsx or json.

GET /api/prices/list/sms/<type>

Status Codes

- 200 OK – Returns a dict with an array containing information on your account.
- 403 Forbidden – Ie. unauthorized ip address
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json

 {
    "premium": [
      {
        "country": "AD",
        "country_name": "Andorra",
        "dkk": 0.33000,
        "eur": 0.04430,
        "prefix": 376
      },
      {
        "country": "AE",
        "country_name": "United Arab Emirates",
        "dkk": 0.19000,
        "eur": 0.02600,
        "prefix": 971
      }
    ],
    "standard": [
      {
        "country": "AD",
        "country_name": "Andorra",
        "dkk": 0.31000,
        "eur": 0.04160,
        "prefix": 376
      },
```

```
            {
              "country": "AE",
              "country_name": "United Arab Emirates",
              "dkk": 0.16000,
              "eur": 0.02100,
              "prefix": 971
            }
          ]
      }
```


1.7 Webhooks

Although the REST API supports polling for the message status, we strongly encourage to use our simple webhooks for getting Delivery Status Notifications, aka DSNs.

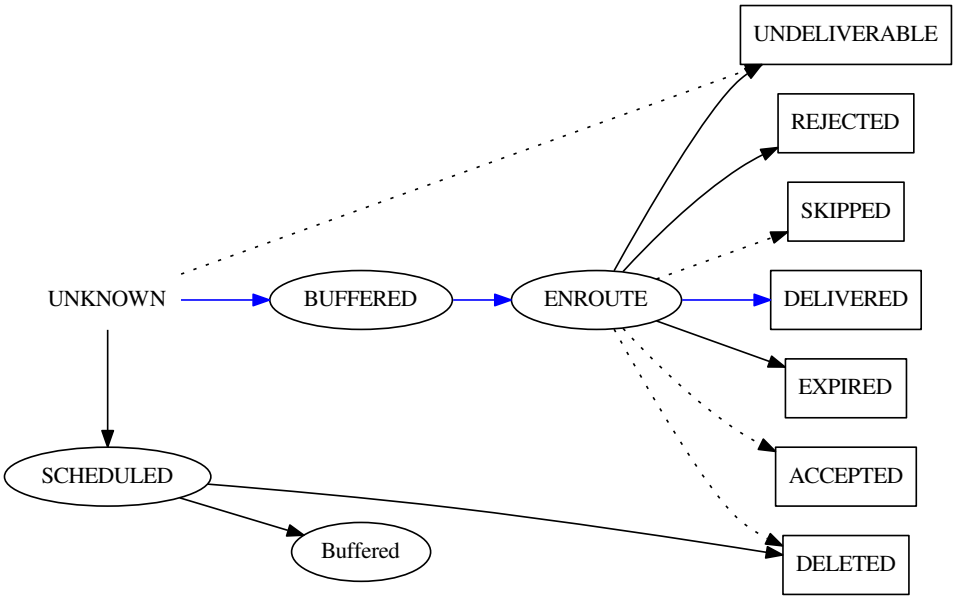
In addition webhooks can be used to react to enduser initiated events, such as MO SMS (Mobile Originated SMS, or Incoming SMS).

If you filter IPs, note that we will call your webhook from the IP range 35.241.147.191/32 and 35.233.1.105/32. In the future we may add IPs but for now this is the range.

1.7.1 Delivery Status Notification

States and status codes

By adding a URL to the callbackurl field, or setting one of your webhooks as the default for status notifications, you can setup a webhook that will be called whenever the current status (state) of the message changes, ie. goes from a transient state (Circles, ie. Enroute) to final state (Boxes, ie. DELIVERED) or an other transient state. Once a final state is reached we will no longer call your webhook with updates for this particular message and recipient.



The normal path for messages are marked in blue above. The dotted lines are very rare events not often used and/or applicable only to specific use cases.

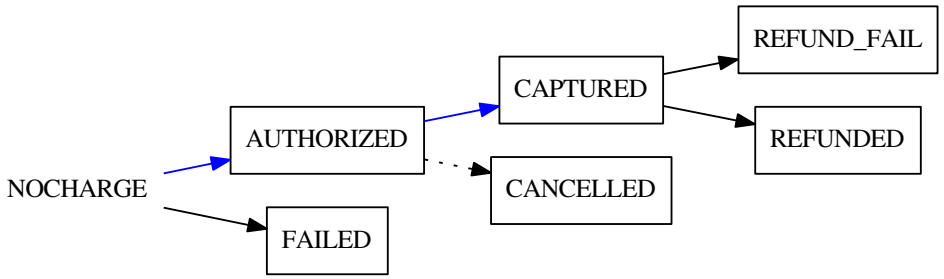
We try to deliver DSNs in a logical order, but they may not always arrive at your webhook in order and sometimes you may receive a transient state after already having received a final state. In this case you should ignore the transient state.

Status	Description
UNKNOWN	Messages start here, but you should not encounter this state.
SCHEDULED	Used for messages where you set a sendtime in the future.
BUFFERED	The message is held in our internal queue and awaits delivery to the mobile network.
ENROUTE	Message has been sent to mobile network, and is on it's way to it's final destination.
DELIVERED	The end user's mobile device has confirmed the delivery, and if message is charged the charge was successful.
EXPIRED	Message has exceeded it's validity period without getting a delivery confirmation. No further delivery attempts.
DELETED	Message was canceled.
UNDELIVERABLE	Message is permanently undeliverable. Most likely an invalid *MSISDN*.
ACCEPTED	The mobile network has accepted the message on the end users behalf.
REJECTED	The mobile network has rejected the message. If this message was charged, the charge has failed.
SKIPPED	The message was accepted, but was deliberately ignored due to network-specific rules.

Charge status

For overcharged smses there is an extra status for the charging. The 'NOCHARGE' state is a placeholder for the start of the charging flow.

The 'REFUND_FAIL' state is just a notification, the actual state will still be 'CAPTURED'.



The normal path for messages are marked in blue above. The dotted lines are very rare events not often used and/or applicable only to specific use cases.

Status	Description
NOCHARGE	Messages start here, but you should not encounter this state.
AUTHORIZED	The transaction is authorized
CANCELLED	The transaction is cancelled or timed out
CAPTURED	The transaction is captured and the amount will be charged from the recipients phone bill
FAILED	The transaction failed. Usually because the phone number has blocked for overcharged sms
REFUNDED	A previously captured transaction has been successfully refunded to the phone owner
REFUND_FAIL	The refund procedure failed.

HTTP Callback

If you specify a callback url when sending your message, or have a webhook configured as your default webhook for status notification, we will perform a http request to your webhook with the following data.

`POST /example/callback`

Example of how our request to you could look like.

Request JSON Object

- **`id`** (*`integer`*) – The ID of the SMS/MMS this notification concerns
- **`msisdn`** (*`integer`*) – The *MSISDN* of the mobile recipient.
- **`time`** (*`integer`*) – The UNIX Timestamp for the delivery status event
- **`status`** (*`string`*) – One of the states above, in all-caps, ie. DELIVERED
- **`error`** (*`string`*) – Optional error decription, if available.
- **`code`** (*`string`*) – Optional numeric code, in hex, see *SMS Errors*, if available.
- **`userref`** (*`string`*) – If you specified a reference when sending the message, it's returned to you
- **`country_code`** (*`string`*) – Optional country code of the msisdn.
- **`country_prefix`** (*`integer`*) – Optional country prefix of the msisdn.

Status Codes

- 200 OK – If you reply with a 2xx code, we will consider the DSN delivered successfully.
- 500 Internal Server Error – If we get a code >= 300, we will re-attempt delivery at a later time.

Callback example

```
POST /example/callback HTTP/1.1
Host: example.com
Accept: */*
Content-Type: application/json

{
    "id": 1000001,
    "msisdn": 4587654321,
    "time": 1450000000,
    "status": "DELIVERED",
    "userref": "foobar",
    "charge_status": "CAPTURED",
    "country_code": "DK",
    "country_prefix": 45
}
```

If we can't reach your server, or you reply with a http status code >= 300, then we will re-attempt delivery of the DSN after a 60 second delay, then 120 seconds, 360 seconds, 24 minutes, 2 hours and lastly after 12 hours. We expect you to reply with a 2XX status code within 15 seconds, or we consider it a failed attempt.

The *charge_status* is only present for overcharged smses.

1.7.2 MO SMS (Receiving SMS'es)

Web hooks are also used to receive SMS'es. We call this MO SMS (Mobile Originated SMS).

Prerequisites

In order to receive a SMS, you'll need a short code and/or keyword to which the user sends the SMS. This short code and keyword is leased to you, so when we receive a SMS on the specific short code, with the specific keyword, we know where to deliver the SMS.

You can either lease a keyword on a shared short code, such as +45 1204, or you can lease an entire short code, such as +45 60575797. Contact us via the live chat if you need a new short code and/or keyword.

If you lease the keyword "foo" on the short code 45 1204, a Danish (+45) user would send ie. "foo hello world" to "1204", and you'll receive the SMS.

Once you have a keyword lease, you'll need to assign the keyword to a webhook. You can do this from the dashboard. * If you do not have a webhook, add one. * Click the webhook you want to receive SMS'es. * Click the tab pane "Keywords" * Make sure the checkbox next to "Assign" is checked for the keywords you want to assign to this webhook.

If you have any questions, please contact us using the live chat found ie. in the lower right when reading the documentation online.

HTTP Callback

`POST /example/callback`
Example of how our request to you could look like. The many optional fields are rarely used.

Request JSON Object

- **id** (*integer*) – The ID of the MO SMS
- **msisdn** (*integer*) – The *MSISDN* of the mobile device who sent the SMS.
- **receiver** (*integer*) – The short code on which the SMS was received.
- **message** (*string*) – The body of the SMS, incl. keyword.
- **senttime** (*integer*) – The UNIX Timestamp when the SMS was sent.
- **webhook_label** (*string*) – Label of the webhook who matched the SMS.
- **sender** (*string*) – If SMS was sent with a text based sender, then this field is set. *Optional.*
- **mcc** (*integer*) – *MCC*, mobile country code. *Optional.*
- **mnc** (*integer*) – *MNC*, mobile network code. *Optional.*
- **validity_period** (*integer*) – How long the SMS is valid. *Optional.*
- **encoding** (*string*) – Encoding of the received SMS. *Optional.*
- **udh** (*string*) – User data header of the received SMS. *Optional.*
- **payload** (*string*) – Binary payload of the received SMS. *Optional.*
- **country_code** (*string*) – Optional country code of the msisdn.
- **country_prefix** (*integer*) – Optional country prefix of the msisdn.

Status Codes

- 200 OK – If you reply with a 2xx code, we will consider the DSN delivered successfully.
- 500 Internal Server Error – If we get a code >= 300, we will re-attempt delivery at a later time.

Callback example

```
POST /example/callback HTTP/1.1
Host: example.com
Accept: */*
Content-Type: application/json

{
    "id": 1000001,
    "msisdn": 4587654321,
    "receiver": 451204,
    "message": "foo Hello World",
    "senttime": 1450000000,
    "webhook_label": "test",
    "country_code": "DK",
    "country_prefix": 45
}
```

If we can't reach your server, or you reply with a http status code >= 300, then we will re-attempt delivery of the DSN after a 60 second delay, then 120 seconds, 360 seconds, 24 minutes, 2 hours and lastly after 12 hours. We expect you to reply with a 2XX status code within 15 seconds, or we consider it a failed attempt.

1.7.3 Authentication token

When setting up your webhook you have an option to add an authentication token if you add text to this field we will use it to make a JWT token, which we will send back to your server in the `X-Gwapi-Signature` header.

JWT is widely supported and you can find libraries for mostly any programming language on https://jwt.io, that will show you how to verify the token.

To verify you need the token we send in the `X-Gwapi-Signature` header and the authentication token that you chose when setting up your webhook.

Code Examples

How to verify JWT tokens in differnt languages. More examples can be found on https://jwt.io.

In the following examples the secret shared between you and GatewayAPI are written directly in the code, in production environments, the shared secret should be part of your configuration, so it is better protected.

- PHP

```php
<?php
require_once 'vendor/autoload.php';
use \Firebase\JWT\JWT;
/*
  Token is extracted from the X-Gwapi-Signature header in the post request
  received on your webserver.
*/
$token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MjM4MTcwMywibXNpc2RuIjo0NTQyNjA5MDQ1LCJ0aW1lI
// secret is the secret token you have chosen when setting up your webhook.
$secret = "secret";
// Verify.
$decoded = JWT::decode($token, $secret, array('HS256'));
print_r($decoded);
?>
```

- Python

```python
# The token variable contains the jwt token
# extracted from X-Gwapi-Signature header from the post request received.
# on your webserver
token = (
  'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MjM4MTcwMywibXNpc2RuIjo'
  '0NTQyNjA5MDQ1LCJ0aW1lIjoxNTIyNzY0MDYyLCJzdGF0dXMiOiJERUxJVkVSRUQiLCJlcnJ'
  'vciI6bnVsbCwiY29kZSI6bnVsbCwidXNlcnJlZiI6bnVsbCwiY2FsbGJhY2tfdXJsIjoiaHR'
  '0cDovL2JiYWY3MTQyLm5ncm9rLmlvIiwiYXBpIjo0fQ.KdfDH65bnQtgxEkFnpAQodOciAJ'
  'edZFB13r9wEo8t3Y')
# The secret chosen by you when setting up your webhook
secret = 'secret'
# Verify
decoded = jwt.decode(token, secret, algorithms=['HS256'])
print(decoded)
```

- NodeJS

```javascript
var jwt = require('jsonwebtoken');
// var secret is the secret that you chose and entered on gatewayapi.com
// when setting up your webhook.
var secret = 'secret'
var auth = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MjM4MTcwMywibXNpc2RuIjo0NTQyNjA5MDQ1LCJ0aW1l
```

```
var decoded = jwt.verify(auth, secret);
console.log(decoded);
```

- Ruby

```
require 'jwt'
token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MjM4MTcwMywibXNpc2RuIjo0NTQyNjA5MDQ1LCJ0aW1lIjo
secret = 'secret'
decoded = JWT.decode token, secret
puts decoded_token
```

1.8 Get MOSMS by API

A webhook is required to receive incoming messages, but in addition messages can also be retreived using a GET request.

GET /rest/mosms

Query Parameters

- **from** – The from date, in YYYY-MM-DD format *required*
- **to** – The to date, in YYYY-MM-DD format *required*
- **page** – Page number
- **per_page** – Results per page (max 200)

Response JSON Array of Objects

- **results** (*int*) – Total results
- **pages** (*int*) – Pages available
- **per_page** (*int*) – Results per page
- **page** (*int*) – Current page
- **messages** (*array*) – Array of messages

Example request

```
GET /rest/mosms?from=2019-01-01&to=2019-01-14
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "messages": [
      {
        "anonymized": null,
        "encoding": null,
        "message": "test",
        "mosms_id": 2398517,
        "msisdn": 4512345678,
        "receiver": 451204,
        "sender": null,
        "senttime": 1554465205.0,
        "webhook": "Rukikab"
```

```
        },
        {
          "anonymized": null,
          "encoding": null,
          "message": "test2",
          "mosms_id": 2398518,
          "msisdn": 4512345678,
          "receiver": 451204,
          "sender": null,
          "senttime": 1554465459.0,
          "webhook": "Fibotfus"
        }
      ],
      "page": 1,
      "pages": 1,
      "per_page": 50,
      "results": 2
    }
```


1.9 Get usage by label

You can get the account usage for a specific date range, sub divided by label and country. This can be used for billing your own customers (specified by label) if you do not keep track of each sms sent yourself.

POST /api/usage/labels

Request JSON Object

- **from** (*string*) – The from date, in YYYY-MM-DD format *required*
- **to** (*string*) – The to date, in YYYY-MM-DD format *required*
- **label** (*string*) – Optional label you want to look for.

Response JSON Array of Objects

- **amount** (*float*) – Amount of SMSes
- **cost** (*float*) – Cost of the SMSes
- **country** (*string*) – The country the SMSes was sent to
- **currency** (*string*) – Either DKK or EUR
- **label** (*string*) – The label specified when the SMSes was sent
- **messageclass_id** (*string*) – The class specified when the SMSes was sent

Status Codes

- 200 OK – Returns a array with a dict containing usage info.
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Request example

```
curl -v https://gatewayapi.com/api/usage/labels
-u 'Your_API_Token_Here':
-d '{"from":"2019-08-01","to":"2019-08-24"}'
-H 'Content-Type: application/json'
```

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "amount": 29,
    "cost": 3.48,
    "country": "DK",
    "currency": "DKK",
    "label": null,
    "messageclass_id": "standard"
  },
  {
    "amount": 6,
    "cost": 1.5,
    "country": "IT",
    "currency": "DKK",
    "label": null,
    "messageclass_id": "standard"
  }
]
```

1.10 Sending emails

You can send emails through gatewayapi using our email endpoint. Contact sales@gatewayapi.com to request access.

If you need SPF on your domain, you will need to include the following in your DNS SPF record: `include:_spf.gatewayapi.com`

`POST /rest/email`

Request JSON Object

- **`html`** (*string*) – The html content of the email.
- **`plaintext`** (*string*) – The plain text content of the email.
- **`subject`** (*string*) – The subject line of the email, tags can be used like in the message to personalise the subject.
- **`from`** (*string*) – The name and email of the sender, can be just the email if no name is specified, see below for format.
- **`reply`** (*string*) – The name and email of the sender, can be just the email if no name is specified, see below for format.
- **`tags`** (*array*) – A list of string tags, which will be replaced with the tag values for each recipient, if used remember to also add tagvalues to all recipients.
- **`attachments`** (*array*) – A list of base64 encoded files to be attached to the email, described below:
- **`data`** (*string*) – The base64 encoded data of the file to attach.

- **filename** (*string*) – The name of the file attached to the email.
- **mimetype** (*string*) – The mimetype of the file, eg. text/csv.
- **recipients** (*array*) – list of email addresses to receive the email, described below:
- **address** (*string*) – The recipient email address.
- **name** (*string*) – The name of the recipient shown in the email client.
- **tagvalues** (*array*) – A list of string values corresponding to the tags in the email. The order and amount of tag values must exactly match the tags.
- **cc** (*array*) – A list of cc recipients, taks an address and optionally a name of the recipient.
- **bcc** (*array*) – A list of cc recipients, taks an address and optionally a name of the recipient.

Status Codes

- 200 OK – Returns a dict with an array of message IDs and a dictionary with usage information on success
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Request example

```
POST /rest/email HTTP/1.1
Host: gatewayapi.com
Authorization: OAuth oauth_consumer_key="Create-an-API-Key",
  oauth_nonce="128817750813820944501450124113",
  oauth_timestamp="1450124113",
  oauth_version="1.0",
  oauth_signature_method="HMAC-SHA1",
  oauth_signature="t9w86dddubh4XofnnPgH%2BY6v5TU%3D"
Accept: application/json, text/javascript
Content-Type: application/json

{
    "html": "<b>Hello %firsname %surname %target is about to be removed.",
    "plaintext": "Hello %firsname %surname %target is about to be removed.",
    "subject": "Annihilation: %target",
    "from": "Darth Vader <darth@example.com>",
    "returnpath": "bounce@example.com",
    "tags": ["%firstname", "%surname", '%target'],
    "recipients": [
        {"address": "l.organa@example.com", "name": "Leia Organa", "tagvalues": ["Leia", "Organa
        {"address": "l.skywalker@example.com", "name": "Luke Skywalker", "tagvalues": ["Luke", "
    ]
}
```

Example response

If the request succeed, the internal message identifiers are returned to the caller like this:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "ids": [
        431332671
    ]
    "usage": {
        "amount": 1,
        "currency": "DKK",
        "total_cost": 0.003
    }

}
```

1.10.1 Email code examples

Code examples for sending emails.

Python

For this example you'll need the excellent Requests-OAuthlib. If you are using pip, simply do `pip install requests_oauthlib`.

```
from requests_oauthlib import OAuth1Session
key = 'Go-Create-an-API-Key'
secret = 'Go-Create-an-API-Key-and-Secret'
gwapi = OAuth1Session(key, client_secret=secret)
req = {
  'html': '<b>Hello %firsname %surname %target is about to be removed.',
  'plaintext': 'Hello %firsname %surname %target is about to be removed.',
  'subject': 'Annihilation: %target',
  'from': 'Darth Vader <darth@galacticempire.com>',
  'reply': 'Count Dokuu <c.dokuu@galacticempire.com>',
  'returnpath': 'bounce@galacticempire.com',
  'recipients': [{'address': 'l.organa@example.com',
                  'name': 'Leia Organa',
                  'tagvalues': ['Leia', 'Organa', 'Alderaan'],
                  'cc': [{'address': 'h.solo@example.com',
                          'name': 'Han Solo'}],
                  'bcc': [{'address': 'chewie@example.com',
                           'name': 'Chewbacca'}],
                  }],
  'tags': ['%firstname', '%surname', '%target'],
  'attachments': [{
  'data': '/9j/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQgKCgkICQkKDA8MCg'
    'sOCwkJDRENDg8QEBEQCgwSExIQEw8QEBD/yQALCAABAAEBAREA/8wABgAQEAX/2gAIAQEA'
    'AD8A0s8g/9k=',
    'filename': 'kyber.jpeg', 'mimetype': 'image/jpeg'}]
}
res = gwapi.post('https://gatewayapi.com/rest/email', json=req)
print(res.json())
res.raise_for_status()
```

1.11 HLR and Number lookup

We are at work on expanding our services with a HLR API, for now we are offering a number lookup API for danish numbers only. This will only be available to selected customer. If you have use of this API talk to us on support and we will figure something out. Requested numbers can be of any of these forms '+4512345678', 004512345678, 4512345678.

POST /rest/hlr

Request JSON Object

- **msisdns** (*array*) – List of numbers to lookup.

Status Codes

- 200 OK – Returns a dict with information for each number in the request.
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address, or account is not authorized to use this API.
- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

Example response

If the requests succeeds information for each valid number passed to the API, will be returned as below.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "currency": "DKK",
   "hlr": {
       "4512345678": {
           "current_carrier": {
               "mcc": "238",
               "mnc": "20",
               "name": "Telia"
           },
           "network_operator": {
               "mcc": "238",
               "mnc": "20",
               "name": "Telia"
           },
           "original_carrier": {
               "mcc": "238",
               "mnc": "20",
               "name": "Telia"
           },
       "ported": false,
       "type": "GSM"
       }
   },
   "lookups": 1,
   "total_cost": 0.06
 }
```

1.11.1 Code examples

Code examples for hlr lookups

Python

For this example you'll need the excellent Requests-OAuthlib. If you are using pip, simply do `pip install requests_oauthlib`.

```
from requests_oauthlib import OAuth1Session
key = 'Go-Create-an-API-Key'
secret = 'Go-Create-an-API-Key-and-Secret'
gwapi = OAuth1Session(key, client_secret=secret)
req = {
    'msisdns': [4512345678]
}
res = gwapi.post('https://gatewayapi.com/rest/hlr', json=req)
print(res.json())
res.raise_for_status()
```

Httpie

For quick testing with a pretty jsonified response in your terminal you can use *Httpie*. It can be done simply using your token as follows.

```
http --auth=GoGenerateAnApiToken: \
https://gatewayapi.com/rest/hlr \
msisdns:='[451234678]'
```

1.12 Refund charged sms

Charged smses that have successfully been captured are eligible for refunds. Sending charged smses requires special setup and permissions. You will not immediately know if the refund is successful, this info will be send to your callback url, or will be visible through the sms log on your backend when updated.

`POST /rest/refund`

Only charged smses with charge status capture, can be refunded.

Request JSON Object

- **`mtsms_id`** (*integer*) – The id of the charged sms to refund.
- **`msisdn`** (*integer*) – The msisdn the charged messages was sent to.
- **`callback_url`** (*string*) – Optional url for getting status of the refund.

Status Codes

- 204 No Content –
- 400 Bad Request – Ie. invalid arguments, details in the JSON body
- 401 Unauthorized – Ie. invalid API key or signature
- 403 Forbidden – Ie. unauthorized ip address
- 404 Not Found – SMS is not found.

- 422 Unprocessable Entity – Invalid json request body
- 500 Internal Server Error – If the request can't be processed due to an exception. The exception details is returned in the JSON body

1.13 Keywords

You can use this API to programmatically add new keywords to your account, for use with value added services (VAS), ie. your services.

Access to this API requires a separate agreement with GatewayAPI, intended for resellers and/or accounts with large needs for two-way messaging.

GET /api/vas
Get a list of keywords

Request Headers

- Authorization – API Token or OAuth bearer token

```
GET /api/vas HTTP/1.1
User-Agent: curl/7.37.1
Host: gatewayapi.com
Accept: */*
authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXV...
```

Example response:

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 148
Date: Mon, 18 May 2015 12:53:59 GMT

[
    {
        'shortcode': 451204,
        'keyword': 'charlie',
        'pushsetting_reference': 'foo'
    }
],
```

POST /api/vas/check
Check if a keyword is available

Request JSON Object

- **keyword** (*string*) – Keyword to search for
- **shortcode** (*integer*) – ie. 451204

Response JSON Object

- **system** (*string*) – System using the keyword (gatewayapi)
- **available** (*boolean*) – Is the keyword available

Request Headers

- Authorization – API Token or OAuth bearer token

Status Codes

- 200 OK – no error, see json output
- 422 Unprocessable Entity – input validation error

```
POST /api/vas/check HTTP/1.1
Host: gatewayapi.com
Accept: */*
Authorization: Basic TzFsa3FtTGhRdHFRMXpHNHp
Content-Type: application/json

{ "shortcode": 451204, "keyword": "foobar" }
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "available": false,
    "system": "gatewayapi"
}
```

POST /api/vas

Add a new keyword

Please note that the price for the keyword will be deducted from your GatewayAPI Credit immediately upon adding the keyword.

When adding a new keyword you are charged the full price, regardless how many days are left in the month until next invoice period.

If you want to assign the new keyword to a webhook right away, set the optional field pushsetting_reference to the "unique label" of the webhook.

Request JSON Object

- **keyword** (*string*) – Keyword to add
- **shortcode** (*integer*) – ie. 451204
- **pushsetting_reference** (*string*) – Optional webhook to assign to.

Request Headers

- Authorization – API Token or OAuth bearer token

Status Codes

- 201 Created – created
- 422 Unprocessable Entity – input validation error

```
POST /api/vas HTTP/1.1
Host: gatewayapi.com
Accept: */*
Authorization: Basic TzFsa3FtTGhRdHFRMXpHNHp
Content-Type: application/json

{ "shortcode": 451204, "keyword": "foobar" }
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
    "account_id": 1,
    "keyword": "foobar",
    "pushsetting_reference": null,
    "shortcode": 451204
}
```

DELETE /api/vas/(int: shortcode)/(*keyword*)

Cancel the keyword.

Note that your keyword will remain active on your account until the end of the subscribtion period.

Query Parameters

- **shortcode** – ie. 451204
- **keyword** – The keyword to delete

Request Headers

- Authorization – API Token or OAuth bearer token

Status Codes

- 202 Accepted – Accepted for deletion at end of payment period.
- 404 Not Found – couldn't find the keyword
- 422 Unprocessable Entity – input validation error

```
DELETE /api/vas/451204/foobar HTTP/1.1
Host: gatewayapi.com
Accept: */*
Authorization: Basic TzFsa3FtTGhRdHFRMXpHNHp
```

Example response:

```
HTTP/1.1 204 OK
Content-Length: 0
```

CHAPTER

TWO

LEGACY HTTP API

For these APIs you need a set of credentials. These are different from the API Keys the REST API uses. You'll find them in the dashboard, under Settings.

2.1 Sending SMS'es

2.1.1 httppost.nimta.com

This API is available at httppost.nimta.com for the old gateway (oc.dk/gateway) and https://gatewayapi.com/legacy/http/v2/sendsms for the new.

It's currently one of the endpoints with the most traffic and it's not going away for the foreseeable future. However we are deprecating it since the REST API is where all future development takes place.

This API is deprecated and only supports sending limited SMS messages. Despite the name you can submit both GET and POST requests to this API endpoint.

`POST /sendsms`

Send a SMS message.

Should only be performed via HTTPS connections since it contains plaintext credentials.

Arguments can be sent as POST (form encoded) or as GET.

If you don't know the mobile network operator or (smsc) then you can set it to ie. "dk.unknown" for danish recipients, as long as your message is not charged.

Form Parameters

- **`user`** – credential username
- **`password`** – credential password
- **`to`** – One or more recipient MSISDNs to send a message to. Ie 4512345678
- **`smsc`** – An ISO 3166-1 country code followed by a period and an ID representing the operator
- **`price`** – A numeric value with two decimals followed by an ISO 4217 currency code, ie. 10.00DKK
- **`text`** – An alphanumeric value representing the content of the SMS.
- **`sessionid`** – Maximum length is 30 characters – and must always be unique. Recommended format is msisdn:time
- **`from`** – Optional alphanumeric sender. Maximum 11 characters

- **callbackurl** – Optional URL for status callbacks
- **class** – Class to use for message delivery. Defaults to 'A'
- **charset** – Charset of inputs, either 'utf-8' or 'iso-8859-1' (default). Only available at gatewayapi.com.

Request Headers

- Content-Type – application/x-www-form-urlencoded

Status Codes

- 200 OK – with a plaintext body: "Processing:sessionid", with sessionid replaced with the given sessionid
- 400 Bad Request – if the request can't be processed due to an exception. The body contains the exception message

Example request:

```
POST /sendsms HTTP/1.1
Host: httppost.nimta.com
Accept: */*
Content-Length: 139
Content-Type: application/x-www-form-urlencoded

user=myusername&password=mypassword&to=4512345678&smsc=dk.tdc&sessionid=4512345678:2010050715101
```

Example response:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/2.2 (FreeBSD)
Content-Length: 36
Content-Type: text/plain

Processing:4512345678:20100507151010
```

2.1.2 nimta.com

This API is available at both www.nimta.com and nimta.com. The API is one of our first and has been deprecated since 2010. It remains operative, however.

This API is also available at https://gatewayapi.com/legacy/http/v1/sendsms.

Originally this API supported various other features such as: WapPush, Charged Messaging and MMS. These features are now defunct and thus left undocumented.

GET /Gateway/Kunder/Opret/Gateway.aspx
Send a SMS message.

Should only be performed via HTTPS connections since it contains plaintext credentials.

Arguments can only be sent as GET Query Params.

Query Parameters

- **username** – credential username
- **password** – credential password

- **number** – The recipient mobile subscriber number, without country code but including any area code. Ie. 87654321
- **countryCode** – The country code of the mobile subscriber, ie. 45
- **message** – The content of the SMS
- **gatewayclass** – Class to use for message delivery, defaults to 'A'
- **alphatext** – Optional alphanumeric sender. Maximum 11 characters

Status Codes

- 200 OK – with a .NET hidden form or other nonsensical output
- 200 OK – If the request can't be processed it will still return 200, but with an error message

2.1.3 Delivery Status Notification

Callbacks are used to respond to changes in the message delivery status, also known as Delivery Status Notifications or DSNs for short.

By adding a URL to the callbackurl field, you can set up a webhook that will be called so you can keep track of whether the message was delivered successfully or not, and if not then why.

Code	Description	Cause
1	Delivered	All okay. Message delivered, and charged if charge was requested
2	Insufficient funds	The recipient lacks the funds, ie. prepaid, or cannot be charged.
3	Blacklisted	The mobile subscriber is blacklisted by the operator, and cannot receive messages
4	Unknown recipient	The msisdn is not recognized by the operator
5	Unknown status	Message is still enroute or an unknown error occurred
6	Expired	Message has expired according to validity period
7	Undeliverable	Message could not be delivered, typically because of error with content
8	Deleted	Message was deleted and not delivered

If you set a callbackurl when you sent the message, we will call your url with one of these status codes and the sessionid you provided when you sent the message. You can use this sessionid to track the message in your internal systems.

When calling your service, we will perform a GET request, ie. https://example.com/callback?sessionid=4587654321:1234&statuscode=1

Beware that if you specify any query params in your callbackurl they will not be returned to you, only the sessionid and statuscode params will be included.

GET /example/callback

Query Parameters

- **sessionid** – The sessionid you provided when you sent the message. Optional.
- **statuscode** – One of the status codes (integer) described above

Status Codes

- 200 OK – If you reply with exactly 200 (not 204 etc) we consider the DSN delivered successfully. Else we re-attempt later.

2.2 Receiving SMS'es

When we receive a MO SMS (mobile originated SMS), we will look at the first word in the SMS, known as the keyword. The SMS is then routed to the customer who has an active subscription for this keyword.

We then send a HTTP GET request to the URL configured for that keyword, ie. https://example.com/mosms?sender=4512345678&smsc=unknown&sessionid=4512345678%3A9379401&appnr=1204&keyword=test

You must respond with a very specific body, otherwise we'll treat your response as a failure and re-attempt delivery of the MO SMS. It's important that the content type is "text/plain" and your reply body is exactly `cmd=asynch-no-trace`, no extra whitespace or other output except headers is allowed.

GET /example/mosms

Example of what our request to you could look like. The path and hostname are configureable of course.

Query Parameters

- **sender** – The MSISDN of the end user who initiated the MO SMS (sent it)
- **smsc** – The SMSC of the end user, this can be used to later send a charged SMS
- **sessionid** – To enable you to track the message we provide an unique sessionid
- **appnr** – Application number or shortcode, where the user sent the SMS
- **keyword** – The keyword we matched
- **text** – The body of the SMS, excluding the matched keyword. Optional.

Response Headers

- Content-Type – must be "text/plain"

Example request:

```
GET /example/mosms?sender=4512345678&smsc=unknown&sessionid=4512345678%3A9379401&appnr=1204&keyw
Host: example.com
Accept: */*
```

Mandatory response:

```
HTTP/1.1 200 OK
Content-Type: text/plain

cmd=asynch-no-trace
```

CHAPTER

THREE

KANNEL API

For this API you need a set of credentials. These are different from the OAuth API the Rest API uses. You'll find them in the dashboard, under "API".

We use the term Kannel API, because this is an API designed specifically for the Open Source Kannel SMS Gateway. It uses the Kannel–HTTP–Kannel interface, which we re-implemented in gatewayapi based on the Kannel source code.

3.1 Config

To use the API simply set up a new Kannel SMSC:

```
group = smsc
smsc = http
system-type = kannel
smsc-id = oc
port = 13019
send-url = "https://gatewayapi.com/kannel/sendsms/standard"
smsc-username = "{YOUR CREDENTIAL USERNAME}"
smsc-password = "{YOUR CREDENTIAL PASSWORD}"
```

The last path fragment is the message class to use. In this example set to 'standard'

The important part is that you set the `system-type` to `kannel`, which tells Kannel to use it's own protocol for this SMSC.

3.2 Features

Our kannel API is (almost) fully featured. The implemented features are listed below with their variable.

charset We support Windows-1252, UTF-8 and UTF-16BE.

coding You can send a SMS with *GSM 03.38* (default), 8-bit or UCS-2 encoding.

dlr-url Specify an URL and have us call it with delivery status reports.

validity Have the SMS expire X minutes from now, so if not delivered will be discarded.

deferred Have the SMS delivered X minutes in the future.

mclass Send a SMS to the ie. screen directly. Aka flash sms.

account Account name or number to enable tracing in your own systems. Will be set in userref.

udh You may specify your udh and optionally binary SMS body for full control.

from You may set an alphanumeric sender max 11 chars, or up to 15 digits for a phonenumber.

The only feature we don't support yet is MWI (message waiting indicator). If you have a good use case for this feature, let us know in the live chat, and we'll add it to the feature request tracker.

3.3 Delivery Reports

If you are not already an expert on kannels status reports, take a brief look at Chapter 10 SMS Delivery Reports.

You are expected to provide a URL, which contains one or more variables which kannel then replaces with data. Especially `%d` which gives back the report type or 'status code'.

3.3.1 Variables

%d Kannel report type, ie. 1 for delivered.

%A The answer of the SMSC. We give an answer in the form `status / (errorcode)`.

%p The MSISDN of the phone who received the message.

%q The international phone number (*E.164*), who received the message.

%T The time of the delivery report expressed as seconds since unix epoch.

%F The SMS ID used on GatewayAPI.com to track the message.

%o The userref/account for this SMS if specified.

3.3.2 DLR URL Chaining

The normal behavior is for you to provide the dlr-url to your kannel server, when you send an SMS using the /cgi-bin/sendsms interface. Then as the final destination we will call your URL and not your kannel server. However if this is not what you want, maybe you have firewall blocking it, you can have us call your kannel server, which will then forward or chain the DLR to your server as if your kannel server was the final destination. This is explained in some length in the Kannel source code.

Should you have a need for kannel to receive the DLR and then forward it to you as "normal" then add this to your config:

```
dlr-url = "http://203.0.113.1:13019/?username=SAME_AS_SMSC&password=SAME_AS_SMSC"
connect-allow-ip = 77.66.39.*
```

You would have to replace the IP with a public IP where we can reach your server. Also make sure username and password is the same as in the *Config* above.

3.4 Caveats

3.4.1 Concatenated SMS

In order to support concatenated SMS, make sure to set the smsbox config var `sms-length` to something more than the default 140. Kannel will then split long SMS'es into several parts and set an UDH to allow the end user device to concatenate them together. Unfortunately kannel insists on doing this on it's end, so in the GatewayAPI traffic log it is going to show each segment separately.

3.4.2 Feedback appreciated

This is a brand new API and although it's tested extensively, it might not exactly fit your kannel setup. We really appreciate feedback on this API. Please contact us via the livechat, especially if you have an urgent situation.

CHAPTER

FOUR

LEGACY SOAP API

For this API you need a set of credentials. These are different from the API Keys the REST API uses. You'll find them in the dashboard, under settings.

The API is compatible with the old Nimta API, so in order to switch from the OnlineCity NIMTA Messaging Gateway, to GatewayAPI.com, you can simply switch the SOAP endpoint and update your credentials etc, but not otherwise change anything.

We still strongly recommend the new APIs such as the REST API, because that's the API we will focus on going forward.

Compared to the SOAP API, the REST offers

- Improved security, incl. replay attack protection
- SMS Tags
- Message priority selection
- User specified tracking references
- Improved performance
- etc...

4.1 How to switch

If you want to switch from nimta.com to gatewayapi.com, there are just three configuration changes you need to made.

Required changes

- Use the WSDL at https://gatewayapi.com/legacy/soap/api.wsdl or switch location to https://gatewayapi.com/legacy/soap
- Update the username and password with your GatewayAPI credentials (not API keys)
- Update the "gatewayClass" to match your wanted GatewayAPI service level, ie. "standard" or "premium", rather than the old "A" and "B" classes.

4.2 Request example

`POST /legacy/soap`
Send a SMS message.

Example request:

```
POST /legacy/soap HTTP/1.1
Host: gatewayapi.com
Connection: Keep-Alive
Content-Type: application/soap+xml; charset=utf-8; action="http://www.nimta.com/webservices/Gate
Content-Length: 1716

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:ns1="http://www.nimta.co
  <env:Body>
    <ns1:sendSMSes>
      <ns1:username>secret</ns1:username>
      <ns1:password>secret</ns1:password>
      <ns1:gatewayClass>standard</ns1:gatewayClass>
      <ns1:messages>
        <ns1:SMSSendMessage>
          <ns1:message>Lorem ipsum dolor sit amet</ns1:message>
          <ns1:alphatext>Example SMS</ns1:alphatext>
          <ns1:charge>0</ns1:charge>
          <ns1:delayed>false</ns1:delayed>
          <ns1:sendTime>2016-10-26T23:23:47+02:00</ns1:sendTime>
          <ns1:recipients>
            <ns1:Recipient>
              <ns1:countryCode>45</ns1:countryCode>
              <ns1:number>12345678</ns1:number>
              <ns1:operatorCode>3</ns1:operatorCode>
            </ns1:Recipient>
          </ns1:recipients>
        </ns1:SMSSendMessage>
      </ns1:messages>
    </ns1:sendSMSes>
  </env:Body>
</env:Envelope>
```

Example response:

```
HTTP/1.0 200 OK
Content-Length: 340
Content-Type: application/soap+xml; charset=utf-8
Server: Werkzeug/0.11.11 Python/3.5.1
Date: Wed, 26 Oct 2016 21:23:47 GMT

<?xml version="1.0" encoding="UTF-8"?>
<soap12env:Envelope xmlns:soap12env="http://www.w3.org/2003/05/soap-envelope" xmlns:tns="http://
  <soap12env:Body>
    <tns:sendSMSesResponse>
      <tns:sendSMSesResult>
        <tns:int>1641644</tns:int>
      </tns:sendSMSesResult>
    </tns:sendSMSesResponse>
  </soap12env:Body>
</soap12env:Envelope>
```

4.3 API Reference

Unfortunately we have not converted the documentation to this new format yet, but you can still use the old documentation with this API.

You can find it on https://oc.dk/gateway/#api or embedded below if reading online.

CHAPTER

FIVE

FTP API

For this API you need a set of credentials. These are different from the OAuth API the Rest API uses. You'll find them in the dashboard, under Settings.

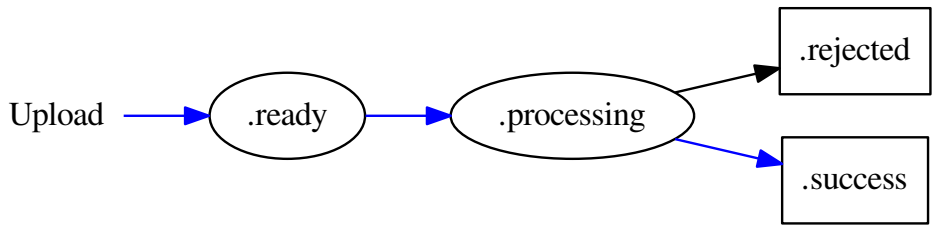
We made this API upon request by our valued customers, but we strongly encourage all customers to use the REST API. You can also batch many SMS'es together in one REST call.

Access to this API is controlled by our support staff, so if you require access please contact our live support and help us understand your use case so we can provide you with the best solution.

We have two active ftp servers, "**a.ftp.gatewayapi.com**" and "**b.ftp.gatewayapi.com**". Both see the same files, if you can't connect to one, try the other server. Ideally have your solution automatically failover if one is temporary unavailable.

5.1 Flow

- After uploading your file, you send the file by adding ".ready" to the end of the file. That is you rename the file. If the file is in excel format, add ".excel.ready".
- The moment you rename the file, we start sending it and ".ready" is changed to a timestamped (Ymd'T'His) ".processing". Beware that most FTP clients cache the file listing, so you might need to click refresh in your FTP client to see this change.
- If the file could not be processed, it will be renamed to ".rejected". The cause of the failure will be added to the bottom of the file.
- When the file is processed and queued for delivery it will be renamed to ".success".



If processing fails due to internal errors, the file batch will be retried up to 5 times, with 30 minutes between each attempt.

5.2 Format

The first row in the CSV must ALWAYS be a list of column names. We understand the following column names, which must match *exactly*.

column	Description
msisdn	*MSISDN* aka the full mobile phone number of the recipient
message	The message to send. If longer than 160 normal chars, it will be split into parts of 153 normal chars.
sender	*Optional*. Up to 11 alphanumeric characters, or 15 digits, that will be shown as the sender of the SMS.
class	*Optional*. Default "standard". The message class to use for this request. If specified it must be the same for all messages in the file.
userref	*Optional*. A transparent string reference, you may set to keep track of the message in your own systems.

We will attempt to auto detect the file encoding (ie. UTF-8 or Windows-1252/cp1252). You can override this by adding ".utf8" or ".cp1252" to the file. Keep in mind the file must *end with* ".ready" in order for processing to start, so to override auto detected and force utf-8, call it ie ".utf8.ready". UTF-16/32 is supported too, but only with auto detection.

Example of a unix encoded file

```
"userref","msisdn","sender","class",message"
"test","4510203040","MyCompany","standard","I'm John Doe, who ""randomly"" quote things."
,"4512345678",,"standard","Optional fields"
```

If you added ".excel" to the file name, excels format is used. The target is what excel outputs on windows, running a Western Language and using the Save As "CSV (Semicolon Delimited)".

```
userref;msisdn;sender;class;message
"test";"4510203040";"MyCompany";"standard";"I'm John Doe, who ""randomly"" quote things."
```

We do not support escaping double quotes by adding backslashes. Double quotes must be escaped by adding an extra double quote, as is the most widely used convention.

You can terminate each row with CR+LF (\r\n) or LF (\n). We recommend LF for unix and CR+LF for excel format, as is standard on the respective platforms. Tab delimited files are not supported.

5.3 Caveats

- You can't upload Excel .xls(x) files, save the file as "CSV (Semicolon delimited)", and remember to add ".excel.ready" when sending the file.
- The encoding/charset of the file is auto detected, consider adding explicit encoding to the file name. Although auto detection is very good, there are a few rare edgecases.
- We recommend you do not create the CSV yourself, but have some standard software do it.
- You can use newlines (ie. \n) as normal, as long as the text is in double quotes " ". Remember that CR+LF (\r\n) is two chars in the SMS (unless you have the new SMS optimization feature enabled which can remove the CR (\r).
- The FTP has strongly limited feature set compared the the REST API, this is by design.
- Many FTP clients cache the directory listing, so they do not show the subsequent ".processing" and ".success" steps unless you force refresh the client.
- You can't delete or rename files in the ".processing" stage. You can however delete .success or .rejected files.

- You can't create new directories.
- You can't modify the file permissions.
- Due to FTPs protocol design, it's not trivial to implement load balancing, so you may experience an outage of one ftp servers during maintenance windows. If possible automatically try the other server if one fails.
- The timestamp added to the files is in UTC
- Existing files will be overwritten, but due to the timestamp this is unlikely.
- Files larger than 30 MB will not be processed, we recommend you split very large batches into smaller files of 10.000 rows each.

CHAPTER

SIX

EMAIL TO/FROM SMS

Documentation of how to send emails to sms and using web hooks to receive messages and have them delivered to your email inbox.

6.1 Email to SMS

To use this API, you need to whitelist one or more emails or an entire domain, for sending smses.

6.1.1 Whitelisting emails

To whitelist emails and domains go to "Settings" "Email Whitelist", add an email like me@example.com or an entire domain e.g. @example.com if your whole organisation should be able to send messages this way.

If you whitelist @example.com no other accounts will be able to use this domain.

Requirements

To successfully deliver your email as a SMS you have two options for proper authentication:

1. Your whitelisted emails or domain, need to pass SPF check and have a DKIM record.
2. You need to put an API token in the Subject field of the e-mail.

For security reasons we highly recommend option 1, since e-mails in most cases are transferred unencrypted over SMTP, an attacker might be able to capture your API tokens. We do provide the second option for customers that are unable to setup SPF and DKIM for their e-mail.

If you use API token and you also use our IP whitelist, then you will need to whitelist the IP-address of your outgoing e-mail server as well.

6.1.2 Sending an SMS

Sending via the Email to SMS api is straightforward, all you need to do is send to the phonenumbers of your recipients and set a default sender id.

Recipients

Sending sms messages is a simple as sending an email to the phonenumber@smtp.gatewayapi.com, for example 4512345678@smtp.gatewayapi.com

Sender ID

Your sender ID is controlled by the default sender setting found under “Settings”, “SMS defaults”.

SMS length

Smses send on this API is limited to 20 sms parts, equal to 3060 characters, this is a measure to try and prevent unintenionally long smses send, when using email clients like e.g. Gmail, where the mails are kept as conversation threads, where responding to the original email, will keep the entrie conversation in the email.

6.2 SMS to Email

Receiving SMS messages from your users in your email inbox, requires you to have a virtual number attached to your account, and to use this number with a SMS to Email web hook.

6.2.1 SMS to Email Webhook

Navigate to “Settings” “Web hooks” and setup a web hook, of the type, SMS to Email Give your web hook a name, and add email adresses that should receive the smses.

Smses will be delivered with from address: 4512345678@smtp.gatewayapi.com for example.

CHAPTER

SEVEN

SMPP

We offer SMPP connection for select customers. Contact sales@gatewayapi.com to get access. We use SMPP version 3.4, which should be backwards compatible with version 3.3.

7.1 Connection

Use the following to connect

Host	smpp1.gatewayapi.com
Port	7777
Port TLS	7778
Bind type	Transceiver or transmitter and receiver.
Maximum sessions	1 transceiver or 1 transmitter/receiver.

7.2 Supported SMPP commands

The following commands are supported

Command	Hex
generic_nack	0x80000000
bind_receiver	0x00000001
bind_receiver_resp	0x80000001
bind_transmitter	0x00000002
bind_transmitter_resp	0x80000002
submit_sm	0x00000004
submit_sm_resp	0x80000004
deliver_sm	0x00000005
deliver_sm_resp	0x80000005
unbind	0x00000006
unbind_resp	0x80000006
bind_transceiver	0x00000009
bind_transceiver_resp	0x80000009
enquire_link	0x00000015
enquire_link_resp	0x80000015

7.3 bind_receiver, bind_transmitter, bind_transceiver

Multiple sessions are supported but must be enabled in SMPP server.

Note that only first active session binded as transceiver or receiver will receive delivery receipts.

Type	Field	Description
COctet	system_id	Account information
COctet	password	Account information
COctet	system_type	Empty or “EX”
Integer	interface_version	0x33 or 0x34
Integer	addr_ton	Ignored
Integer	addr_npi	Ignored
COctet	address_range	Ignored

CHAPTER

EIGHT

ERROR CODE INDEX

You may encounter errors not listed here, but we try to keep this list updated. If you get an error it will always be accompanied with an error description, so this list provides a reference, but you don't need to know all these.

All error codes in gatewayapi.com follow the same pattern. They are a 32-bit unsigned integer, often represented in hex format as four hex chars, ie. 0x104A. The first two hex chars (first byte), represent the origin of the error. The following list of error codes are divided into sections depending on the origin.

8.1 General error codes

Regardless of the error origin, all error codes that end in 00-0F, have the same meaning. These do not map to a specific error, but may be caused by different issues depending on the context. Thus these are left undocumented since the exact cause varies.

Code	Description
00	Success (no error)
01	Unknown error
02	Generic NACK
03	Operation not permitted
04	Input/output error
05	Shutting down
06	Insufficient Resources
07	Permission denied
08	Invalid Argument
09	Resource temporarily unavailable (EAGAIN)

This means that if you get an 0x1001, it's an "Unknown error" from the *SMS Errors*. If you get an 0x0308 it's an "Invalid Argument" from *API Errors* etc.

8.2 API Errors

These cover codes 0x0100-0x07FF. You might encounter these when communicating with one of our API's ie. *REST API*.

Code	HTTP Status Code	Description
0x0210	401	Invalid username: %1
0x0210	401	Invalid username: %1
0x0211	401	Invalid password
0x0212	401	Invalid IP-address: %1
0x0213	403	Unauthorized IP-address: %1
0x0214	403	Temporary blacklist for MSISDN %1 with hash %2
0x0215	403	MSISDN %1 blacklisted
0x0216	403	Insufficient credit
0x0217	403	Unauthorized destination: country %1
0x0218	422	Unknown message class id %1
0x0219	403	E-mail not enabled for account
0x021A	403	HLR not enabled for account
0x021B	403	Phonebook not enabled for account
0x021C	403	MMS not enabled for account
0x021D	403	SMS not enabled for account
0x021E	500	Message class %1 is not configured for %2
0x021F	403	Unauthorized operator: MCC %1, MNC %2
0x0220	400	Unsupported parameter %1
0x0221	400	Unsupported signature method %1
0x0222	400	Missing required parameter %1
0x0223	401	Invalid Consumer Key
0x0224	401	Invalid signature
0x0225	401	Expired timestamp
0x0226	401	Invalid / used nonce
0x0229	401	Invalid token
0x022A	403	Account frozen, contact support
0x0310	422	A message contains the same recipient more than once. MSISDNs %1 is duplicated
0x0311	422	Priority %1 not available
0x0312	422	Messages filtered based on content

8.3 SMS Errors

The following table is an exhaustive list of error codes that might be returned to you in a Delivery Status Notification. We do our best to keep track of all these, but some may be missing. Not all error codes are relevant or expected to occur in normal operations. This section covers codes 0x0100-0x01FF.

These supplement the *States and status codes* to provide additional insight on failures.

Group	Code	Description
HLR	0x1010	Unknown subscriber
HLR	0x1011	Call barred
HLR	0x1012	Teleservice not provisioned
HLR	0x1013	Absent subscriber
HLR	0x1014	Facility not supported
HLR	0x1015	HLR System failure
HLR	0x1016	Unexpected data value
HLR	0x1017	HLR Data missing

Continued on next page

Table 8.1 – continued from previous page

Group	Code	Description
HLR	0x1018	Memory capacity exceeded
HLR	0x1019	Mobile subscriber not reachable
HLR	0x101A	VLR System failure
HLR	0x101B	HLR Internal Error
MSC	0x1020	Unidentified subscriber
MSC	0x1021	Absent subscriber, IMSI detached
MSC	0x1022	Absent subscriber, no page response
MSC	0x1023	Subscriber busy for MT SMS
MSC	0x1024	Facility not supported
MSC	0x1025	Illegal subscriber
MSC	0x1026	Illegal equipment
MSC	0x1027	System failure
MSC	0x1028	Unexpected data value
MSC	0x1029	Data missing
MSC	0x102A	Memory capacity exceeded
MSC	0x102B	Equipment protocol error
MSC	0x102C	Equipment not short message equipped
MSC	0x102D	Illegal error
MSC	0x102E	MSC Internal Error
SCREEN	0x1030	Screening block
SCREEN	0x1031	Terminating IMSI blocked
SCREEN	0x1032	Originating location mismatch
SCREEN	0x1033	Error, originator blocked
SCREEN	0x1034	Error, destination blocked
SCREEN	0x1035	Error, keyword blocked
SCREEN	0x1036	Error, SC address blocked
SCREEN	0x1037	Error, blocked due to exceeded quota
SCREEN	0x1038	Error, loop detected
SCREEN	0x1039	Error, data coding scheme blocked
SCREEN	0x103A	Error, information element identifier blocked
SCREEN	0x103B	Error, personal service barring, MO
SCREEN	0x103C	Error, personal service barring, MT
SMSC	0x1040	Unidentified Subscriber
SMSC	0x1041	Facility not supported
SMSC	0x1042	System failure
SMSC	0x1043	Unexpected data value
SMSC	0x1044	Data missing
SMSC	0x1045	Equipment protocol error
SMSC	0x1046	Unknown service centre address
SMSC	0x1047	Service centre congestion
SMSC	0x1048	Invalid short message entity address
SMSC	0x1049	Subscriber not service centre subscriber
SMSC	0x104A	SMSC Internal Error
ROUTE	0x1050	Internal routing error
ROUTE	0x1051	Unsupported number plan
ROUTE	0x1052	Unsupported type of number
ROUTE	0x1053	Message not deliver
ROUTE	0x1054	Dialling zone not found
ROUTE	0x1055	Not home zone and IMSI not allowed

Continued on next page

Table 8.1 – continued from previous page

Group	Code	Description
ROUTE	0x1056	Not home zone and IMSI fetch failed
ROUTE	0x1057	Destination network type unknown
ESME	0x1060	Invalid destination address
ESME	0x1061	Invalid destination numbering plan
ESME	0x1062	Invalid destination type of number
ESME	0x1063	Invalid destination flag
ESME	0x1064	Invalid number of destinations
ESME	0x1065	Invalid source address
ESME	0x1066	Invalid source numbering plan
ESME	0x1067	Invalid source type of number
ESME	0x1068	ESME Receiver permanent error
ESME	0x1069	ESME Receiver reject error
ESME	0x106A	ESME Receiver temporary error
ESME	0x106B	Invalid command length
ESME	0x106C	Invalid service type
ESME	0x106D	Invalid operation
ESME	0x106E	Operation not allowed
ESME	0x106F	Invalid parameter
ESME	0x1070	Parameter not allowed
ESME	0x1071	Invalid parameter length
ESME	0x1072	Invalid optional parameter
ESME	0x1073	Optional parameter missing
ESME	0x1074	Invalid validity parameter
ESME	0x1075	Invalid scheduled delivery parameter
ESME	0x1076	Invalid distribution list
ESME	0x1077	Invalid message class
ESME	0x1078	Invalid message length
ESME	0x1079	Invalid message reference
ESME	0x107A	Invalid number of messages
ESME	0x107B	Invalid predefined message
ESME	0x107C	Invalid priority
ESME	0x107D	Invalid replace flag
ESME	0x107E	Request failed
ESME	0x107F	Invalid delivery report request
ESME	0x1080	Message queue full
ESME	0x1081	External error
ESME	0x1082	Cannot find information
ESME	0x1081	IMSI lookup blocked
ESME	0x1082	ESME error
ESME	0x1082	ESME Internal error
ESME	0x1083	ESME Unknown external error
ESME	0x1084	Invalid Mobile Subscriber
ESME	0x1085	Short message exceeds maximum
ESME	0x1086	Unable to Unpack GSM message
ESME	0x1087	Unable to convert IRA Alphabet
SP	0x1090	Internal error
SP	0x1091	Network time-out
SP	0x1092	Operation barred - insufficient funds
SP	0x1093	Illegal mobile subscriber - blocked

Continued on next page

Table 8.1 – continued from previous page

Group	Code	Description
SP	0x1094	Refunded by network operator

CHAPTER

NINE

APPENDIX

Throughout the documentation we are going to refer some terms that might not be commonly used outside the mobile gateway business, so you'll find them all explained here.

9.1 Glossary

MT SMS Mobile Terminated SMS. Sent SMS. Outgoing SMS. A SMS going from our gateway to an end users mobile device.

MO SMS Mobile Originated SMS. Received SMS. Incoming SMS. A SMS sent by an end user to one of our application numbers.

MSISDN Mobile Station International Subscriber Directory Number, but you may think of this as the full mobile number, including area code if available and the country code, but without prefixed zeros or '+'.

Examples:

- 4510203040 (In Denmark known as: 10 20 30 40)
- 46735551020 (In Sweden known as: 073-555 10 20)
- 17325551020 (In US known as: (732) 555-1020)

The MSISDN is easily interchangeable with *E.164* numbers, you simply remove or add the leading '+' in *E.164*. It can contain up to 15 digits, so we use an unsigned 64-bit integer.

E.164 The standard format for international phone numbers. Up to 15 digits.

MCC Mobile Country Code, as defined by the ITU-T E.212 standard.

MNC Mobile Network Code, as defined by the ITU-T E.212 standard.

GSM 03.38 The de facto standard encoding in SMS, which supports up to 160 normal characters in a single SMS, and 153 normal characters in each SMS in a chained/concatenated SMS. It achives this by using 7 bits for each character and supports the common characters in most western languages.

The full character set is:

- Basic Latin
  - (Space) ! " # $ % & ' ( ) * + , - . /
  - 0 1 2 3 4 5 6 7 8 9
  - : ; < = > ? @
  - a b c d e f g h i j k l m n o p q r s t u v w x y z
  - A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Special characters
  - (Currency sign) £ ¥
  - § ¿ _
  - (Newline \n), (Carriage Return \r)
- Greek characters
  - Δ Φ Γ Λ Ω Π Ψ Σ Θ Ξ
- Diacritics
  - è é ù ì ò Ç Ø ø Å å Æ æ ß É Ä Ö Ñ Ü ä ö ñ ü à ä ö ñ ü à
- These characters use two GSM 03.38 chars to encode one of the following
  - ^ { } [ ] ~ | €

UCS2 Universal Coded Character Set, an early Unicode implementation. For the most part you can use UTF-16-BE (Big Endian) interchangeably with UCS-2.

9.2 SMS Length

A sms message has a maximum size of 140 bytes, messages larger than 140 bytes are chained together using some of these bytes, leaving 134 bytes for the message on each "page".

Using *GSM 03.38* the length is extended by only using 7 bits per character instead of the normal 8 bits used by UTF-8 this gives 160 characters for one page messages, if a messages is longer than those 160 charcters, 7 bits will be used to chain the messages together as pages giving a total of 153 characters per page.

Using *UCS2* encoding will enable to use a wide range of different characters that is not available in *GSM 03.38*, this is achieved by using 2 bytes per character. giving you 70 characters for one page messages, if you go above 70 characters the messages will be chained giving you 134 bytes or 67 characters for your message.

In short:

- GSM 03.38 (default)
  - Uses 7 bits per character, to extend the 140 bytes max length of a sms
  - 160 normal characters for one page.
  - Messages longer than 160 characters are chained and gives a length of 153 characters per page.
- UCS2
  - Allows the use of a lot of different special characters by using 2 bytes per character
  - 70 characters for one page.
  - Messages longer then 70 characters are chained and gives a leaves 67 characters per page.

9.3 SMS Sender

The originator of the SMS message. The SMS standards limit the length up to 15 digits if it's a number and up to 11 characters if it's a text. You can use spaces in the sender, but most modern smartphones do not display the space.

For some destinations there may be country/network specific restrictions on the senders, and the sender may be automatically replaced or you may need to use a special sender for the destination/network.

We recommend you stick with characters in the range a-zA-Z0-9, however if you do use Latin-1 characters ie. (æøå) we will support it on connections where it is available. If the mobile network connection do not support these characters, we will automatically replace them with basic latin chracters according to the table below. If the replacement results in a too long sender, only the first character of the replacement is used.

Code	Char	Replacement
00A0		
00A1	¡	!
00A2	¢	C/
00A3	£	PS
00A4		$?
00A5	¥	Y=
00A6	¦	
00A7	§	SS
00A8	¨	“
00A9	©	(c)
00AA	ª	a
00AB	«	<<
00AC	¬	!
00AD		
00AE	®	(r)
00AF	¯	-
00B0	°	deg
00B1	±	+-
00B2	²	2
00B3	³	3
00B4	´	‘
00B5	µ	u
00B6	¶	P
00B7	·	*
00B8	¸	,
00B9	¹	1
00BA	º	o
00BB	»	>>
00BC	¼	1/4
00BD	½	1/2
00BE	¾	3/4
00BF	¿	?
00C0	À	A
00C1	Á	A
00C2	Â	A
00C3	Ã	A
00C4	Ä	A
00C5	Å	Aa
00C6	Æ	Ae
00C7	Ç	C
00C8	È	E

Continued on next page

Table 9.1 – continued from previous page

Code	Char	Replacement
00C9	É	E
00CA	Ê	E
00CB	Ë	E
00CC	Ì	I
00CD	Í	I
00CE	Î	I
00CF	Ï	I
00D0	Ð	D
00D1	Ñ	N
00D2	Ò	O
00D3	Ó	O
00D4	Ô	O
00D5	Õ	O
00D6	Ö	O
00D7	×	x
00D8	Ø	Oe
00D9	Ù	U
00DA	Ú	U
00DB	Û	U
00DC	Ü	U
00DD	Ý	Y
00DE	Þ	Th
00DF	ß	ss
00E0	à	a
00E1	á	a
00E2	â	a
00E3	ã	a
00E4	ä	a
00E5	å	aa
00E6	æ	ae
00E7	ç	c
00E8	è	e
00E9	é	e
00EA	ê	e
00EB	ë	e
00EC	ì	i
00ED	í	i
00EE	î	i
00EF	ï	i
00F0	ð	d
00F1	ñ	n
00F2	ò	o
00F3	ó	o
00F4	ô	o
00F5	õ	o
00F6	ö	o
00F7	÷	/
00F8	ø	oe

Continued on next page

Table 9.1 – continued from previous page

Code	Char	Replacement
00F9	ù	u
00FA	ú	u
00FB	û	u
00FC	ü	u
00FD	ý	y
00FE	þ	th
00FF	ÿ	y

We route the traffic to the best connection regardless of their support of special characters, so you may experience the sender is replaced. With that said, if you have a special need for these characters in your sender fields, contact support and we can work something out.

You can also consult the HTTP Routing Table

In addition to our dashboard, which you can find on https://gatewayapi.com/app/, we also offer programmatic access to a few of the functions offered in the dashboard. You will find the documentation for these at https://gatewayapi.com/docs/website/

Writing and improving this documentation is an ongoing effort. Pull requests etc are most welcome. https://github.com/onlinecity/gatewayapi-docs

/Gateway

/api

/legacy

/rest

/sendsms

E

G

M

U