Lekab Sms Rest Batch Web Service

Lekab Communication Systems AB

Version 7.0.36, 2026-11-02

Lekab Sms Rest Batch Web Service

Introduction . . . 1
Different authentication methods available for requests . . . 2
Supported character sets in the resulting SMS and pricing issues . . . 2
1. The /batchsend/json and /batchsend/list endpoints . . . 4
1.1. POST /batchsend/json request example . . . 4
1.2. POST /batchsend/list request example . . . 4
1.2.1. Explanation of parameters for /batchsend/json and /batchsend/list . . . 5
1.2.2. HTTP response to /batchsend/json and /batchsend/list . . . 8
1.2.3. Explanation of response to /batchsend/json and /batchsend/list . . . 8
1.2.4. Batch status . . . 9
1.2.5. Format of the list for /batchsend/list . . . 9
1.2.6. Example Python 3 code for /batchsend/json . . . 11
1.2.7. Example Python 3 code for /batchsend/list . . . 12
1.2.8. Use of saved text templates and substitution of template style place holders . . . 13
1.3. Idempotency . . . 14
2. The /batchinfo endpoint . . . 16
2.1. GET request example . . . 16
2.2. POST request example . . . 16
2.2.1. Explanation of parameters for /batchinfo . . . 16
2.2.2. HTTP response to /batchinfo . . . 17
2.2.3. Explanation of response to /batchinfo . . . 17
2.2.4. Batch status . . . 17
2.2.5. Example Python 3 code . . . 18
3. The /batchmessageid endpoint . . . 19
3.1. GET request example . . . 19
3.2. POST request example . . . 19
3.2.1. Explanation of parameters for /batchmessageid . . . 19
3.2.2. HTTP response to /batchmessageid . . . 20
3.2.3. Explanation of response to /batchmessageid . . . 20
3.2.4. Example Python 3 code . . . 20
4. The /batchmessagestatus endpoint . . . 21
4.1. GET request example . . . 21
4.2. POST request example . . . 21
4.2.1. Explanation of parameters for /batchmessagestatus . . . 22
4.2.2. HTTP response to /batchmessagestatus . . . 22
4.2.3. Explanation of response to /batchmessagestatus . . . 23
4.2.4. Message status codes . . . 24
4.2.5. Non-final statuses . . . 24

4.2.6. Successful final status . . . 25
4.2.7. Failing final statuses . . . 25
4.2.8. Unclear . . . 25
4.2.9. Example Python 3 code . . . 25
5. The `/batchstatuscount` endpoint . . . 26
5.1. GET request example . . . 26
5.2. POST request example . . . 26
5.2.1. Explanation of parameters for /batchstatuscount . . . 27
5.2.2. HTTP response to /batchstatuscount . . . 27
5.2.3. Explanation of response to /batchstatuscount . . . 27
5.2.4. Message statuses . . . 28
5.2.5. Non-final statuses . . . 28
5.2.6. Successful final status . . . 28
5.2.7. Failing final statuses . . . 29
5.2.8. Unclear . . . 29
5.2.9. Example Python 3 code . . . 29

Introduction

© 2006 - 2026 Lekab Communication Systems AB. Version 7.0.36, 2026-11-02.

This is a Web Service using `HTTP POST` requests of two different formats to send batches of, potentially, many SMS messages: tens or hundreds of thousands.

Where our non-batch web service returns a result only when all messages have been processed and queued for sending, this batch web service makes a quick validation of the input while writing a temporary file and returns a batch id already when the validated batch has been submitted to a background batch processing system. This serves to avoid time-out conditions for large requests; users of Lekab's older SOAP Batch web service will be familiar with the concepts.

If the batch does not pass validation (e.g. due to the presence of invalid phone numbers, lack of mandatory parameters or invalid credentials), none of the messages in the batch will be sent, and the service returns a relevant HTTP error response code and a json document with an error message.

We offer `HTTP GET` and `HTTP POST` versions of endpoints for retrieving batch information and SMS delivery status for messages in a batch. The batch id, received upon sending, is used here to check the status. Many of the syntax and authentication details for these endpoints are very similar to the related Lekab Sms Rest Web Service (cf. that documentation).

Since different REST client software libraries attain varying degrees of sophistication, the aim has been to make the demands on the sending `HTTP POST` body complexity as low as possible. Accordingly, multi-part MIME input formats are not demanded (or supported), because of this uncertainty whether a customer's `HTTP` client can supply any given version.

Instead the `/batchsend HTTP POST` endpoint has two branches: `/batchsend/json` and `/batchsend/list`, where the `/json` endpoint takes a single json document in the `HTTP` body and all overall parameters and all individual recipient data are included as fields in this json object, while the `/list` endpoint accepts the overall parameters as query parameters in the URL and the individual recipient data in a line based semicolon separated list (essentially a .csv file with ';' separators) in the `HTTP` body. In the simplest case, this list is a list of recipient phone numbers, one per line.

`UTF-8` encoding is assumed everywhere (the standard for json), and where URL-encoding of input data is necessary (e.g. for URL query parameters), the underlying encoding is `UTF-8` (the standard for url encoding).

Every endpoint returns a response in the `HTTP` response body as a json document.

The formats of the input and output json and list documents and the input url parameters are described below.

Different authentication methods available for requests

Every request to the service must include authentication, i.e. username and password (or equivalent). For `POST` requests (except `/batchsend/list`) these can be given in in the corresponding fields in the `JSON` document which is sent in the (automatically `HTTPS = SSL/TLS` encrypted) `HTTP` body. For `POST /batchsend/list` and all `GET` requests, username and password can instead be supplied in the `U` and `P` URL parameters. Note that everything in the URL after the host name is also part of the encrypted request, so URL parameters are equally secure during transfer as parameters in the body or headers. However, calling web services from a web browser URL line may lead to authentication credentials being stored in the browser history, which can potentially be unsafe.

We also offer three different alternative ways of supplying these username and password credentials available in both the `GET` and `POST` cases:

1. Username and password can be given as standard Basic authentication, in which the header `"Authorization"` should have the value `"Basic " + token`, where the token is the `Base64` encoding of (a `UTF-8` byte array representation of) `username:password`. Here `testuser:testpass` will be encoded as `dGVzdHVzZXI6dGVzdHBhc3M=`, so the header with key `"Authorization"` should have the value `"Basic dGVzdHVzZXI6dGVzdHBhc3M="`
2. Username and password can be given in the HTTP headers, `X-Lekab-Userid` and `X-Lekab-Password`, respectively. The values have to be the `Base64` encoding of (a `UTF-8` byte array representation of) the username or password to allow non-US-ASCII characters. Here testuser will be encoded as `dGVzdHVzZXI=` and testpass as `dGVzdHBhc3M=`
3. An API key obtained from Lekab can be used as a query parameter `key`, as the value of the header `X-API-Key` or as the JSON field `apikey` in the `POST` request body (except for `/batchsend/list`). The length of the key varies depending on the length of the username (which is contained within the key). `TUxVOmRHVnpkSFZ6WlhJOjlKUUczTXU2TVZVU1Exd3Y` is a possible example key for the username `testuser`. The key is independent of the account password.
4. Oauth authentication is available from a separate endpoint By supplying the username and password as Basic authentication to the `/token` endpoint, in an `HTTP POST` call of type `application/x-www-form-urlencoded` containing the body "grant_type=client_credentials", one obtains a token valid for 10 minutes. This token can then be supplied as a `Bearer` token in the HTTP request, so that the header `Authorization` shall have the value `Bearer + one space + token`. More information about this in the **AuthenticationAPI** documentation.

Whichever method of authentication is used, parameter values pertaining to other authentication methods should be omitted or set to the empty string "".

Supported character sets in the resulting SMS and pricing issues

The character set used in the input to the API does not affect the resulting SMS. Instead, the

characters that are requested to be sent will determine the size and pricing of the message.

Most network operators globally support all printable characters in SMS messages. They also support the sending of longer SMS messages by dividing the content into several SMS message parts (with some exceptions, like South Korea). An SMS message part is limited in size to 140 bytes; longer messages will require more parts. The character set used, not the language, will determine how many parts are needed. The pricing is based on the number of SMS message parts sent. Normally, the recipient's mobile handset will automatically reassemble the parts of a divided message in the correct order, without any action needed by the recipient.

The GSM standard regulates SMS communication globally. While all characters are supported, some characters used in Western European languages are given special treatment. Two character encodings (character sets) are supported by almost all network operators in the world, and therefore by us: GSM-7 and UCS-2.

By GSM-7 we mean the internationally common GSM-7 alphabet according to GSM 03.38 / 3GPP 23.038 without any national language shift table (only the basic character extension). See for example https://en.wikipedia.org/wiki/GSM_03.38 for details.

The GSM-7 encoding uses 7 bits per character to encode only characters from these Western European languages (including numbers and some punctuation), while the UCS-2 encoding can encode characters from any Unicode alphabet and uses 16 bits per character. If every character in a message can be encoded with GSM-7, the message will employ the more compact GSM-7 encoding. If any character in a message cannot be encoded with GSM-7, UCS-2 will be used for the entire message.

Since the GSM-7 is a 7-bit character set, 160 characters will fit into one 140 bytes SMS part. The UCS-2 is a 16-bit character set and can accommodate only 70 characters in a 140 bytes SMS part. If a GSM-7 encoded message is longer than 160 characters, or a non-GSM-7 encoded message is longer than 70 characters, it will be divided into more than one part. Due to how such parts are constructed, multi-part SMS messages can only fit 153 GSM-7 characters or 67 UCS-2 characters per part. All parts of a message will use the same encoding, so they will all be GSM-7 or all UCS-2.

The inclusion, even accidentally, of a non-GSM-7 encodable character in a message will therefore transform it into a UCS-2 encoded message. This message will have a larger number of shorter parts, and the send-out will be more costly. Therefore, when sending SMS messages in Western European languages, it is often advisable to be on the lookout for any such characters that may be included in the message. For instance, some characters automatically generated in some circumstances by Microsoft Word, like curved citation marks (curved quotes) and unbreakable spaces, are not included in the GSM-7 character set. French vowels with the accent circonflexe also do not fit into GSM-7. Unsurprisingly, Cyrillic, Arabic, Chinese, Thai, Japanese and many other alphabets are not included, and neither are emojis, while ordinary text in English, Swedish, Finnish, German and similar languages will fit into the GSM-7 encoding. The Euro-sign (European Union currency symbol) is included in the GSM-7 basic character extension, which we support.

Chapter 1. The /batchsend/json and /batchsend/list endpoints

Used for sending a batch of SMS

1.1. POST /batchsend/json request example

probably from an application

```
https://secure.lekab.com/restsms/api/batchsend/json?DB=Y&DNN=Y&DNM=Y
```

With the contents of the HTTP body:

```
{"username":"testuser","password":"testpass","from":"LEKAB",
 "message":"Hallå där!","batchconversation":"Sendout 123",
 "batch":[
    {"t":"46701223344"},
    {"t":"447711223344","m":"Hello there!"},
    {"t":"46702345678"},
    {"t":"46703344556"},
    {"t":"46704421232"}
 ]
}
```

1.2. POST /batchsend/list request example

probably from an application

```
https://secure.lekab.com/restsms/api/batchsend/list?U=testuser&P=testpass&F=LEKAB&M
8=Hall%C3%A5+d%C3%A4r!&BX=Sendout+123&DB=Y&DNN=Y&DNM=Y
```

With the contents of the HTTP body:

```
# this is Sendout 123
46701223344
447711223344;Hello there!
46702345678
46703344556
46704421232
```

1.2.1. Explanation of parameters for /batchsend/json and /batchsend/list

`/json` **json key**	`/list` **query param**	**json value (strings quoted)**	**query param value (strings without quotes)**	
username	U	string	string	username of the LEKAB SMS sending account
password	P	string	string	password of the LEKAB SMS sending account
apikey	key	string	string	API key of the LEKAB SMS sending account
from	F	string	string	sender id of the SMS (ignored for two-way SMS)
message	M8	string (UTF-8)	string URL-encoded	Message to send. May contain template style place holders or other place holders or not
templateid	TPID	string containing integer	string	Id of saved template to use as text instead of message parameter. May contain template style place holders or other place holders or not
batchconversation	BX	string	string URL-encoded	conversation id identifying the batch (max 100 characters)
costcenter	C	string	string	grouping in billing (max 255 characters)
billingid	BILL	string	string	grouping in billing (max 255 characters). Use only for users in companies where this is enabled.
validminutes	V	string containing integer	string	validity time of SMS (default is 1440 i.e. 24h)
defaultcountrycode	D	string containing digits	string containing digits	replaces leading zero in **t** numbers from file starting with single zero
"check mobile number" use as query in url	CMN	n/a	T, TRUE, Y or YES	Check that numbers are in probable mobile number series according to Google PhoneNumberUtil library, raise an error if not

`/json` **json key**	`/list` **query param**	**json value (strings quoted)**	**query param value (strings without quotes)**	
"drop blocked" use as query in url	DB	n/a	T, TRUE, Y or YES	Silently drop numbers blocked for the user (if not set, sending will not go through if blocked number is found)
"drop non mobile" use as query in url	DNM	n/a	T, TRUE, Y or YES	Silently drop numbers not in a probable mobile number series according to Google library (if not set, sending will be attempted to non-mobile number, leading to a cost)
"drop non number" use as query in url	DNN	n/a	T, TRUE, Y or YES	Silently drop numbers that are obviously not phone numbers (if not set, sending will not go through if e.g. "46MONKEY" is found)
replacecharacters	R	boolean (default false)	T, TRUE, Y or YES	replace certain characters not in the GSM-7 standard character set with similar characters in that set (e.g. non-breaking space u00A0 replaced by ordinary space u0020). Only applied if enabled for the account; contact Lekab sales for details.
holders	H	json list of string	comma separated URL-encoded strings	place holder strings in the message, that can be replaced for each individual recipient: 1st place holder with 1st substituent etc.
templateHolders	TPH	json list of string	comma separated URL-encoded strings	saved template style curly bracket place holder labels in the message, e.g., abc is label for "{Text:abc}" or "{DateTime:abc:Scheduled at:}", that can be replaced: 1st place holder with 1st substituent etc.
batch		json list of objects containing **t**, **m**, **i**, **s** fields (see below)		individual recipient list for /json endpoint

`/json` **json key**	`/list` **query param**	**json value (strings quoted)**	**query param value (strings without quotes)**	
t		string		to number: phone number in international format: plusses, spaces, hyphens, full stops and parentheses will be disregarded
m		string		individual message (only to this recipient)
i		string		conversation id for this recipient (if omitted, recipient shares the batch conversation id)
s		json list of string		text substitutions for the corresponding placeholders in message (see **holders** parameter)
scheduletime	AT	string containing time	string URL-encoded UTF-8 containing time	Schedule sendout for sending in the future. ISO 8601 standard format like "2022-02-14T15:00:00Z" or "2022-02-14T15:00Z" or "2022-02-14T17:00:00+02:00 or "2022-02-14T13:00-02:00"
Idempotency-Key	(same)	HTTP header string		Unique id for rejecting duplicate API calls (see "Idempotency" below)
X-Request-ID	(same)	HTTP header string		Unique id id for rejecting duplicate API calls (see "Idempotency" below)

There is a duplicate check, so that the exact same message will not be sent to the exact same number within a batch. No check on duplicates from other batches is performed.

Note that the four error handling flags `CMN`, `DB`,`DNM`,`DNN` have to be on the command line, not inside the json document, for technical reasons having to do with the possibility of very large sendouts. Mobile number check via the Google PhoneNumberUtil library is offered as an option, since users may want to send to some number series, especially the extra long numbers sometimes used for Internet-of-Things devices, that are not recognized as possible mobile numbers. Number checks against the user's blocked list and against obvious non-numbers (e.g. containing letters) are always active and the choice there is whether to edit the phone number list or to drop the numbers silently.

When scheduling a batch to a future send time, the format of the time string is strictly constrained. The date must be in the format `YYYY-MM-DD` then there has to be a `T` capital letter T,

then `HH:MM:SS` in a 24-hour scheme, and at last, a time zone marker, which may be either `Z` or `+01:00` or `-01:00` (with any reasonable value instead of `01:00`). The `Z` means the same as `+00:00` i.e. UTC/Zulu/GMT time with no time zone difference or summer time and the plus or minus deviations are deviations from Z. Swedish winter time is `+01:00` and Swedish summer time is `+02:00`. Scheduling to a time in the past or the next 3 minutes is not allowed.

Note that the `billingid` parameter should only be set by users who both have the use of billingid enabled for their company, and have a role set that they may change the billingid to a list of allowed values for their company. For other users, the use of a non-blank billingid will result in an error and no message is sent.

1.2.2. HTTP response to /batchsend/json and /batchsend/list

A successful request will return 200 OK and a json document of the following format. Note that the request for sending may be successful even if the send status of some or all SMS sendings are failed.

The subsequent overall fate of the batch can be found with the `/batchinfo` endpoints.

A list of message IDs belonging to the batch can be obtained with the `/batchmessageid` endpoints. This list can be used when querying for the status of individual messages.

A successfully sent message may sometimes not be delivered due to external factors (phone turned off, phone subscription expired, no such subscriber). The later fortune of the messages in the batch can be followed via the `/batchmessagestatus` endpoints.

All successful responses will have status `Received`

The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "batchid" : "490922766334210048",
  "batchconversation" : "Sendout 123",
  "batchstatuscode" : 1,
  "batchstatusdescription" : "Received"
}
```

1.2.3. Explanation of response to /batchsend/json and /batchsend/list

json key	json value (strings quoted)	
batchid	string	id of this batch, used in the status endpoints to refer to the batch
batchconversation	string	conversation id for the batch supplied when sending to identify the batch
batchstatuscode	integer	the status of the batch (see Batch status table) as an integer

json key	json value (strings quoted)	
batchstatusdescription	string	description of the status of the batch (see Batch status table)

1.2.4. Batch status

A batch has been fully validated, processed and queued for sending to the operator when the returned status is **0 (Ok)**. Single-digit codes are still in progress, while double-digit codes are error conditions.

Status Code	Description
0	Ok
1	Received
2	Processing
3	Validating
7	Scheduled
10	Unexpected error
11	Quota exceeded
12	Maximum batch size exceeded
13	Access Denied
14	Validation error
15	Dropped due to send time restrictions
99	Batch Aborted

1.2.5. Format of the list for /batchsend/list

The supplied list should have lines of the following format:

```
<phone number>;<message>;<conversation>;<substitution 1>;<substitution 2>; ...
```

where the phone number is mandatory and all other fields are optional.

The semicolons must be present up until right before the last supplied optional field. Fields that are not supplied will have the value "" (empty string) in substitutions.

1. The phone number should preferably be in international format without any long-distance prefix (like 00). Alternatively, a beginning zero will be replaced by the default country code supplied in the call. Punctuation, namely, hyphen, parentheses, plus sign and full stops, will be removed. So a format like **46701234567** is preferred, but **+46(70)123.45.67** will work and **0701-234567** will also work if "**46**" is the default country code.
2. The message is an individual message to be used for this recipient phone number only. It

will run through a URL decoding function, so characters that are replaced upon URL decoding will have to be URL encoded to survive. This is especially important for "+", which will be replaced by space, "%" which has a special meaning in URL encoding and ";" which is the separator between fields on a line in this file. Use %2B for "+", %3B for ";" and %25 for "%". Other UTF-8 characters will survive, but it is perfectly fine to URL encode all non-URL characters, with %C3%96 for "Ö" etc. if that is preferred (e.g. by an automatic function). If no message is supplied, the common message for the batch will be used.

3. The conversation is a reference (max 100 characters) to the individual message, to be used at the sender's convenience. It will run through a URL decoding function, like the message, so similar precautions are needed for special characters. If no individual conversation id is supplied, the common conversation id for the batch is used for this recipient.
4. The substitutions are texts that will replace the corresponding placeholder strings in the (common) message. They will each run through a URL decoding function, like the message, and similar precautions are needed. Omitted substitution fields have the value "" (the empty string), so the corresponding placeholder will disappear. Excessive substitution fields (more than the number of placeholders) will be ignored.
5. Lines that are empty or only contain spaces are ignored.
6. Lines where the first non-space character is # (hash mark) will be ignored and can be used for comments.
7. We recommend that the last line of the file end with a line break.
8. There are no provisions for using any other order of the fields on the line. The proper order must be arranged by the sending software.

Example file with valid lines:

```
# This is an example file
46701234567;;conversation001;Karin;Stockholm City
46701223344;;conversation002;Sven;G%C3%B6teborg+C

46701223355;Special message: go back to bed;conversation003

46702112233;;conversation004
46702112266;;conversation005;Björn+Borg;Södertälje Syd
```

which could e.g. be used with the message:

```
Hello NAME! Your train leaves in one hour from the station STATION. Love, National
Train Service
```

and the placeholders

```
H=NAME,STATION
```

The simplest possible list example (everyone gets the same message) is a list of phone numbers separated by line breaks:

```
46701234567
46701223344
46701223355
46702112233
46702112266
```

1.2.6. Example Python 3 code for /batchsend/json

```
import json
import requests
import base64

sendurl = 'https://secure.lekab.com/restsms/api/batchsend/json'
jsondata = {}

jsondata['from']='Lekab'
jsondata['message']='Spidey Sense Members get 20% off everything in the web shop on
Wednesday! Love, Spiderman'
jsondata['batchconversation']='wed_campaign_feb'
b = []
b.append({"t":"46701223344"})
b.append({"t":"447711223344", "m":"You only get 10% off, because you are
British!"})
b.append({"t":"46702345678"})
b.append({"t":"46703344556"})
b.append({"t":"46704421232"})
jsondata['batch'] = b

json_data = json.dumps(jsondata)

authstringarray="testuser:testpass".encode('utf-8')
authbase64=base64.b64encode(authstringarray).decode('utf-8')
headers={'Authorization':'Basic '+ authbase64, 'Content-Type':'application/json'}

response = requests.post(sendurl, data=json_data, headers=headers)

sendresp = response.json()
print("Batch " + sendresp["batchid"] + "(" + sendresp["batchconversation"] + ")
gets status " + sendresp["batchstatusdescription"])
```

will output

```
Batch 490922766334210048(wed_campaign_feb) gets status Received
```

1.2.7. Example Python 3 code for /batchsend/list

```
import json
import requests
import base64
import urllib.parse

sendurl = 'https://secure.lekab.com/restsms/api/batchsend/list'
jsondata = {}

jsondata['from']='Lekab'
sendurl += '?F=' + jsondata['from']
jsondata['message']='Spidey Sense Members get 20% off everything in the web shop on
Wednesday! Love, Spiderman'
sendurl += '&M8=' + urllib.parse.quote_plus(jsondata['message'])
jsondata['batchconversation']='wed_campaign_feb'
sendurl += '&BX=' + urllib.parse.quote_plus(jsondata['batchconversation'])

authstringarray="testuser:testpass".encode('utf-8')
authbase64=base64.b64encode(authstringarray).decode('utf-8')
headers={'Authorization':'Basic '+ authbase64}

with open('C:/Users/Bob/Documents/WebCampaigns/wednesday_february.csv', 'rb') as f:
    response = requests.post(sendurl, data=f, headers=headers)

sendresp = response.json()
print("Batch " + sendresp["batchid"] + "(" + sendresp["batchconversation"] + ")
gets status " + sendresp["batchstatusdescription"])
```

where the file wednesday_february.csv contains (note the URL-encoding of the percent sign in the second message):

```
# this is wednesday_february
46701223344
447711223344;You only get 10%25 off, because you are British!
46702345678
46703344556
46704421232
```

will output

```
Batch 490922766334210048(wed_campaign_feb) gets status Received
```

1.2.8. Use of saved text templates and substitution of template style place holders

It is sometimes convenient to use a set of standard message templates where only a part of the message text is unique to the individual message. In such a template, some character sequences function as place holders, which are to be substituted with the individualized text pieces. In the web portal, such templates can be edited and stored with an id. The id of the template is shown on the editing page in the newest version of the web portal, and is used in this API with the `templateid` / `TPID` parameter. If a template id is given, any explicit message given in the `message` / `M8` parameters is ignored. Even where no template id is given, a template text with place holders for substitutions can be equally well be supplied as the message. That is, use of a saved template and use of place holder substitution are independent features that may be used alone or together.

In the newest version of the web portal, the feature for editing and saving text templates is improved to allow a use case where logged-in users send single SMS messages by manually adding individual data for slots in the template message into input boxes on the SMS sending web page, allowing a coherent messaging strategy between different recipients.

When editing this type of template in the web portal, convenient buttons add curly bracket style place holders of the form `{Text:customer}` or `{Text:ITEMNO}` or `{Text:City:Location of the store}` or `{DateTime:today}`. Here `customer`, `ITEMNO`, `City` and `today` are labels referring to these place holders. In this API, there is a short-cut to substitute such template derived place holders in in the same way as place holders without curly brackets, taking care to keep the upper-case and lower-case of the label exactly as in the template. Any curly bracket place holders not given a substitution value for a recipient are replaced with the empty string in the output message.

Note that the web portal template editing page can also add place holders where data pertaining to the logged-in user or the address book contact of the recipient is to be extracted into the message, and these place holder types are not supported in the batch API, which is primarily aimed at use without reference to contacts in any address books stored with us, and without a logged-in manual web portal user.

Thus, place holders can be listed exactly as they occur in the template, using the `holders` / `H` parameter. When `"holders":["NAME", "THE CITY"]` or `H=NAME,THE+CITY` are specified, the text:

```
Hello NAME! You can pick up your delivery at our store in THE CITY. Best regards,
Our Company
```

will be replaced for a given recipient, so that `NAME` is replaced by the first substituent in the `"s"` json list, or, in the `/list` endpoint, with the text after the third semicolon on the input line. Likewise for `THE CITY` being replaced by the second substituent in the `"s"` json list, or, in the `/list` endpoint, with the text after the fourth semicolon.

But curly bracket saved template style place holders can also be referred to via their label, using the `templateHolders` / `TPH` parameter. When `"templateHolders":["name", "City"]` or `TPH=name,City` are specified, the text:

```
Hello {Text:name}! You can pick up your delivery at our store in {Text:City}. Best
regards, Our Company
```

will give a similar result as the previous example.

In a case where both plain style `holders` / `H` and curly bracket style `templateHolders` / `TPH` are used, the first substituents refer to the `holders`, followed by the `templateHolders`. This use mode is allowed but may be confusing. If any labels used inside the curly bracket place holders happen to coincide exactly with plain style place holders, results are unpredictable. Do not do that! `H=name&TPH=name` is a bad idea!

1.3. Idempotency

With `HTTP POST` API calls, the issue of **idempotency** arises. A procedure that results in the same final state if called twice with the same parameters, as if called once, is said to be **idempotent**. According to the general standards of `HTTP`, a `POST` call is **not** guaranteed to result in the same final state if called twice with the same parameters, i.e. it is **not guaranteed to be idempotent**. In the case of message batch sending, **two calls to this API, even with the same text to the same number, will result in sending two batches** (which is, of course, **not** the same that would happen if there were only one call to the API). So the API calls to the `/batchsend` endpoints are **not automatically idempotent**.

If there are any doubts whether a certain batch has actually been sent already, something that can happen e.g. with unstable network connections or with client software that erroneously produces duplicates, there is a need for suppression of such duplicate sendout, i.e. **idempotency enforcement**. With such suppression in place, the user may resend the request without fear that double messages (with accompanying double billing) would be sent to the number.

In this API we offer the possibility to supply in the call an `HTTP` header with an id that is supposed to be unique for this API call. If such a header is present, it is checked against a list of (hashed to non-identifiability) calls done recently. If the API call was already recently serviced, it is not repeated.

The two available `HTTP` headers are the most common (semi-standard) **idempotency enforcement headers**: `Idempotency-Key` and `X-Request-ID`, and either may be used with the same effect. The API works just fine without these headers, but one may be supplied for suppressing duplicates if such a risk is likely.

The full responsibility for using unique ids lies on the API calling user. If a malfunctioning client software twice composes the exact same message to the exact same number, but marks it with different unique ids, we have no way of detecting this.

We do not support "replay" of the answer to the original API call in the case of duplicates, instead there is a `400` error specifying that a duplicate id was encountered.

Regardless of whether a header is supplied or not, we also offer a check on the SMS message content, only allowing one SMS message with a given text to a given number within a time period (4 hours). Duplicate messages with the same text to a given number will be suppressed.

This functionality is turned on by setting a certain **duplicate suppression role** on the sending user account profile. If the role is set, there is no way of making exceptions using this api, so duplicate messages to the same phone number will always be suppressed for the user (until the role is turned off or the time period has lapsed). When a message in a batch is suppressed by the role, the api will not be aware that this happened, and there is no instantaneous error message, so message statuses will have to be reviewed later.

Chapter 2. The `/batchinfo` endpoint

Used to retrieve the current status of a batch that was sent earlier. Outputs the same info that is returned when sending, but with the current status (see Batch Status table).

2.1. GET request example

e.g. from web browser or curl, but in real use-cases probably from an application

```
curl
https://secure.lekab.com/restsms/api/batchinfo?U=testuser&P=testpass&BI=490922766334210048
```

2.2. POST request example

Probably from an application

```
https://secure.lekab.com/restsms/api/batchinfo
```

with the contents of the HTTP body:

```
{
   "username" : "testuser",
   "password" : "testpass",
   "batchid" : "490922766334210048"
}
```

2.2.1. Explanation of parameters for /batchinfo

POST json key	GET query param	json value (strings quoted)	query param value (strings without quotes)	
username	U	string	string	username of the LEKAB SMS sending account
password	P	string	string	password of the LEKAB SMS sending account
apikey	key	string	string	API key of the LEKAB SMS sending account
batchid	BI	string	string	batch id of the batch for which current status is to be retrieved

2.2.2. HTTP response to /batchinfo

A successful request will return 200 OK and a json document of the following format. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "batchid" : "490922766334210048",
  "batchconversation" : "Sendout 123",
  "batchstatuscode" : 0,
  "batchstatusdescription" : "Ok"
}
```

When a status of **0 (Ok)** is returned, the batch has been successfully processed and queued for sending to the operator. Depending on the batch size and which operators are involved, the actual SMS sending may take less than a second up to hours, but the status 0 is a final status and will not change further. Other single-digit codes are still in progress, and will have changed upon later checking, while double-digit codes are final error conditions (see Batch status table).

2.2.3. Explanation of response to /batchinfo

json key	**json value (strings quoted)**	
batchid	string	id of this batch, used in the status endpoints to refer to the batch
batchconversation	string	conversation id for the batch supplied when sending to identify the batch
batchstatuscode	integer	the status of the batch (see Batch status table) as an integer
batchstatusdescription	string	description of the status of the batch (see Batch status table)

2.2.4. Batch status

A batch has been fully validated, processed and queued for sending to the operator when the returned status is **0 (Ok)**. Single-digit codes are still in progress, while double-digit codes are error conditions.

Status Code	**Description**
0	Ok
1	Received
2	Processing
3	Validating
10	Unexpected error

Status Code	Description
11	Quota exceeded
12	Maximum batch size exceeded
13	Access Denied
14	Validation error
15	Dropped due to send time restrictions
99	Batch Aborted

2.2.5. Example Python 3 code

```
import json
import requests

info = {"username" : "testuser", "password": "testpass", "batchid" :
"490922766334210048"}
info_json = json.dumps(info)
url = 'https://secure.lekab.com/restsms/api/batchinfo'
response = requests.post(url, data=info_json)
inforesp = response.json()
print("Batch " + inforesp["batchid"] + "(" + inforesp["batchconversation"] + ")
gets status " + inforesp["batchstatusdescription"])
```

will output

```
Batch 490922766334210048(wed_campaign_feb) gets status Ok
```

Chapter 3. The /batchmessageid endpoint

Used to retrieve a list of all the message ids in a batch that was sent earlier. It is only when /batchinfo returns a 0 (Ok) status, that the full list of ids will be available.

3.1. GET request example

e.g. from web browser or curl

```
curl
https://secure.lekab.com/restsms/api/batchmessageid?U=testuser&P=testpass&BI=490922
766334210048
```

3.2. POST request example

Probably from an application

```
https://secure.lekab.com/restsms/api/batchmessageid
```

with the contents of the HTTP body:

```
{
   "username" : "testuser",
   "password" : "testpass",
   "batchid" : "490922766334210048"
}
```

3.2.1. Explanation of parameters for /batchmessageid

POST json key	GET query param	json value (strings quoted)	query param value (strings without quotes)	
username	U	string	string	username of the LEKAB SMS sending account
password	P	string	string	password of the LEKAB SMS sending account
apikey	key	string	string	API key of the LEKAB SMS sending account
batchid	BI	string	string	batch id of the batch for which current status is to be retrieved

3.2.2. HTTP response to /batchmessageid

A successful request will return 200 OK and a json document of the following format. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "messageids" : [ "180024553", "180024554", "180024556", "180024558", "180024559"
]
}
```

When a status of **0 (Ok)** is returned when calling `/batchinfo`, the batch has been successfully processed and queued for sending to the operator. At that point forward, a call to /batchmessageid will return the full list of the messageids.

3.2.3. Explanation of response to /batchmessageid

json key	**json value (strings quoted)**	
messageids	json list of string	ids of the messages in the batch

3.2.4. Example Python 3 code

```
import json
import requests

ids = {"username" : "testuser", "password": "testpass", "batchid" :
"490922766334210048"}
ids_json = json.dumps(ids)
url = 'https://secure.lekab.com/restsms/api/batchmessageid'
response = requests.post(url, data=ids_json)
idsresp = response.json()
for s in idsresp["messageids"]:
    print("id=" + s)
```

will output

```
id=180024553
id=180024554
id=180024556
id=180024558
id=180024559
```

Chapter 4. The `/batchmessagestatus` endpoint

Used to retrieve the statuses of the SMS messages in a batch that was sent earlier.

This endpoint is multi-functional, depending on which of the input parameters are supplied. It can either retrieve unread statuses of SMS messages sent in a batch, or retrieve the statuses for a list of given SMS message ids (independent on whether they were read before or not).

1. You can either give a batch id and/or a batch conversation id, which will request a number of statuses (up to the maxnum parameter, which defaults to 1000 and is maximally allowed to be 10000) of statuses from the specified batch(es). This mode of calling has markasread defaulting to true, and the intended way it is supposed to work is that one reads statuses for chunks of the batch until no new statuses appear.
2. Or you can give a list of message ids and/or message conversation ids to get the statuses for all your batch messages with those ids. Here markasread defaults to false, but you can set it to true if you want to mark the message statuses as read.

If you give any message ids and/or conversation ids, the batch id and batch conversation id parameters will be ignored.

4.1. GET request example

e.g. from web browser or curl

```
curl
https://secure.lekab.com/restsms/api/batchmessagestatus?U=testuser&P=testpass&I=180
024553,180024554,180024556
```

4.2. POST request example

Probably from an application

```
https://secure.lekab.com/restsms/api/batchmessagestatus
```

with the contents of the HTTP body:

```
{
   "username" : "testuser",
   "password" : "testpass",
   "messageids" : [ "180024553", "180024554", "180024556" ]
}
```

4.2.1. Explanation of parameters for /batchmessagestatus

POST json key	GET query param	json value (strings quoted)	query param value (strings without quotes)	
username	U	string	string	username of the LEKAB SMS sending account
password	P	string	string	password of the LEKAB SMS sending account
apikey	key	string	string	API key of the LEKAB SMS sending account
batchid	BI	string	string	batch id of the batch for which current status is to be retrieved
batchconversation	BX	string	string	batch conversation id of the batch for which current status is to be retrieved
messageids	I	json list of string	comma separated strings	list of ids of the SMS for which send status is to be retrieved
messageconversations	X	json list of string	comma separated strings	list of conversation ids of the SMS for which send status is to be retrieved
maxnum	N	integer (default 1000, max 10000)	integer	max number of statuses to retrieve (ignored if ids are listed)
markasread	R	boolean (default true or false)	F, FALSE, N, NO, T, TRUE, Y or YES	flag whether the status should be marked as read (defaults to false if message ids or message conversation ids are given, true if no such ids are given)

4.2.2. HTTP response to /batchmessagestatus

A successful request will return 200 OK and a json document of the following format. The `Content-Type` header of the response is `application/json` for all responses.

```
{
   "statuses" : [ {
     "batchid" : "490922766334210048",
     "batchconversation" : "Sendout 123",
     "to" : "46701223344",
     "from" : "LEKAB",
     "id" : "180024553",
     "status" : "DELIVERED",
```

```
      "statuscode" : "2",
      "conversation" : "Sendout 123",
      "time" : "1577031270000"
    }, {
      "batchid" : "490922766334210048",
      "batchconversation" : "Sendout 123",
      "to" : "447711223344",
      "from" : "LEKAB",
      "id" : "180024554",
      "status" : "DELIVERED",
      "statuscode" : "2",
      "conversation" : "Sendout 123",
      "time" : "1577031272000"
   }, {
      "batchid" : "490922766334210048",
      "batchconversation" : "Sendout 123",
      "to" : "46702345678",
      "from" : "LEKAB",
      "id" : "180024556",
      "status" : "UNDELIVERABLE",
      "statuscode" : "6",
      "conversation" : "Sendout 123",
      "time" : "1577031269000"
   } ]
}
```

In this example, two of the messages were delivered, while one got the status Undeliverable from the operator. Since no individual conversation ids were supplied, all messages are marked with the batch conversation id.

4.2.3. Explanation of response to /batchmessagestatus

POST json key	json value (strings quoted)	
statuses	json list of json documents	list of send statuses retrieved
batchid	string	id of the batch to which the message belongs
batchconversation	string	conversation id for the batch supplied when sending to identify the batch
to	string	recipient phone number
from	string	sender id (number or alphanumeric)
id	string	id of the SMS
status	string	name of the status state of the message (DELIVERED is good)

POST json key	json value (strings quoted)	
statuscode	string containing integer **0** to **13**	integer code of the status state of the message, 2 is good
conversation	string	conversation id given when sending the message
time	string containing long integer	timestamp in milliseconds after the epoch (1970-01-01 00:00:00 Z)

4.2.4. Message status codes

The following message status codes can be received in the message status.

statuscode	status	Description
0	QUEUED	Queued for delivery
1	SENT	Sent to operator
2	DELIVERED	Delivered to the mobile station
3	DELETED	The message was deleted
4	EXPIRED	The message has expired
5	REJECTED	The message was rejected by the operator
6	UNDELIVERABLE	The message could not be delivered
7	ACCEPTED	The message was accepted by the operator
8	ABSENTSUBSCRIBER	The subscribers mobile station is switched off
9	UNKNOWNSUBSCRIBER	The subscriber is not known
10	INVALIDDESTINATION	The destination address is invalid
11	SUBSCRIBERERROR	The mobile station can not receive the message
12	UNKNOWN	The status of the message is unknown
13	ERROR	Internal error when sending the message

4.2.5. Non-final statuses

A later status update from the mobile carrier is quite likely to change the value.

`QUEUED (0)`, `SENT (1)`

Note that when the recipient's phone is turned off, this is where the status is stuck until the phone is turned on or the message expires.

4.2.6. Successful final status

The recipient's phone has supposedly acknowledged receipt of this message.

`DELIVERED (2)`

4.2.7. Failing final statuses

The recipient will not get this message.

`DELETED (3)`, `EXPIRED (4)`, `REJECTED (5)`, `UNDELIVERABLE (6)`, `ABSENTSUBSCRIBER (8)`, `UNKNOWNSUBSCRIBER (9)`, `INVALIDDESTINATION (10)`, `SUBSCRIBERERROR (11)`, `ERROR (13)`

4.2.8. Unclear

Probably failing final statuses (probably the mobile carrier has lost or dumped the message, but it may suddenly turn up).

`ACCEPTED (7)`, `UNKNOWN (12)`

4.2.9. Example Python 3 code

```
import json
import requests

statuses = {"username" : "testuser", "password": "testpass", "messageids" :
["180024553", "180024554", "180024556"]}
statuses_json = json.dumps(statuses)
url = 'https://secure.lekab.com/restsms/api/batchmessagestatus'
response = requests.post(url, data=statuses_json)
statusresp = response.json()
for s in statusresp["statuses"]:
    print("id=" + s["id"] + ", status=" + s["status"])
```

will output

```
id=180024553, status=DELIVERED
id=180024554, status=DELIVERED
id=180024556, status=UNDELIVERABLE
```

Chapter 5. The /batchstatuscount endpoint

Used to retrieve counts of the statuses of the SMS messages in a batch that was sent earlier. The message statuses are not marked as read, so the same values will be retrieved every time unless statuses have changed or old messages have been deleted or archived.

You supply a batch id and/or a batch conversation id and will get the number of messages with each status in the specified batch(es). Probably, one or the other is given to refer to the batch, and the resulting list of status counts will have one json document in it. If both id and conversation id are given, and they refer to different batches, the list will have two json documents in it.

We recommend this endpoint only be called once per minute or less often. Polling every second (or even more often) does not add any value. The database call will be relatively heavy for larger batches, and the values change slowly: at first, when all messages have been sent to the operator, many delivery receipts come back from the operator in the first minute, but there is no inherent value in knowing the progress every second. After the first few minutes, new delivery receipts only arrive when a formerly turned off mobile handset is turned on, triggering delivery, and finally after 24 to 72 hours when messages expire for mobile handsets that were never turned on.

5.1. GET request example

e.g. from web browser or curl

```
curl
https://secure.lekab.com/restsms/api/batchstatuscount?U=testuser&P=testpass&BI=4909
22766334210048
```

5.2. POST request example

Probably from an application

```
https://secure.lekab.com/restsms/api/batchstatuscount
```

with the contents of the HTTP body:

```
{
   "username" : "testuser",
   "password" : "testpass",
   "batchid" : "490922766334210048"
}
```

5.2.1. Explanation of parameters for /batchstatuscount

POST json key	GET query param	json value (strings quoted)	query param value (strings without quotes)	
username	U	string	string	username of the LEKAB SMS sending account
password	P	string	string	password of the LEKAB SMS sending account
apikey	key	string	string	API key of the LEKAB SMS sending account
batchid	BI	string	string	batch id of the batch for which counts of current message statuses are to be retrieved
batchconversation	BX	string	string	batch conversation id of the batch for which counts of current message statuses are to be retrieved

5.2.2. HTTP response to /batchstatuscount

A successful request will return 200 OK and a json document of the following format. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "statuses" : [ {
    "batchid" : "490922766334210048",
    "batchconversation" : "Sendout 123",
    "counts" : {
      "DELIVERED" : 4,
      "UNDELIVERABLE" : 1
    }
  } ]
}
```

In this example, four of the messages were Delivered to the mobile handset, while one got the status Undeliverable from the operator.

5.2.3. Explanation of response to /batchstatuscount

POST json key	json value (strings quoted)	
statuses	json list of json documents	list of batch status counts retrieved, one json document per batch

POST json key	json value (strings quoted)	
batchid	string	id of the batch to which this status count refers
batchconversation	string	conversation id for the batch supplied when sending to identify this batch
counts	map(string → integer)	name of the status of SMS message (DELIVERED is good) mapped to the number of occurances of that status

5.2.4. Message statuses

The following message statuses can occur in the count. The desired, final value is `DELIVERED`.

status	Description
`QUEUED`	Queued for delivery
`SENT`	Sent to operator
`DELIVERED`	Delivered to the mobile station
`DELETED`	The message was deleted
`EXPIRED`	The message has expired
`REJECTED`	The message was rejected by the operator
`UNDELIVERABLE`	The message could not be delivered
`ACCEPTED`	The message was accepted by the operator
`ABSENTSUBSCRIBER`	The subscribers mobile station is switched off
`UNKNOWNSUBSCRIBER`	The subscriber is not known
`INVALIDDESTINATION`	The destination address is invalid
`SUBSCRIBERERROR`	The mobile station can not receive the message
`UNKNOWN`	The status of the message is unknown
`ERROR`	Internal error when sending the message

5.2.5. Non-final statuses

A later status update from the mobile carrier is quite likely to change the value.

`QUEUED`, `SENT`

Note that when the recipient's phone is turned off, this is where the status is stuck until the phone is turned on or the message expires.

5.2.6. Successful final status

The recipient's phone has supposedly acknowledged receipt of this message.

`DELIVERED`

5.2.7. Failing final statuses

The recipient will not get this message.

`DELETED`, `EXPIRED`, `REJECTED`, `UNDELIVERABLE`, `ABSENTSUBSCRIBER`, `UNKNOWNSUBSCRIBER`, `INVALIDDESTINATION`, `SUBSCRIBERERROR`, `ERROR`

5.2.8. Unclear

Probably failing final statuses (probably the mobile carrier has lost or dumped the message, but it may suddenly turn up).

`ACCEPTED`, `UNKNOWN`

5.2.9. Example Python 3 code

```
import json
import requests

statuses = {"username" : "testuser", "password": "testpass", "batchid" :
"490922766334210048"}
statuses_json = json.dumps(statuses)
url = 'https://secure.lekab.com/restsms/api/batchstatuscount'
response = requests.post(url, data=statuses_json)
statusresp = response.json()
for s in statusresp["statuses"]:
    print("id=" + s["batchid"] + ", conv=" + s["batchconversation"])
    print("---")
    for k in s["counts"].keys():
        print(k + " : " + str(s["counts"][k]))
    print("---")
```

will output

```
id=490922766334210048, conv=Sendout 123
---
DELIVERED : 4
UNDELIVERABLE : 1
---
```