Rich Messaging Multichannel API

LEKAB Communication Systems AB

Version 7.0.36, 2026-11-02

Table of contents

Introduction . . . 1
Authentication methods available for requests . . . 2
How to obtain an oauth token from our oauth server . . . 3
POST request examples . . . 3
Explanation of parameters . . . 4
1. The `/send` endpoint . . . 5
1.1. POST request example: a text type message without suggestions . . . 5
1.2. POST request example: single card type message, suggestions in the card and after . . . 6
1.3. POST request example: thread of one text and two media messages (pictures) . . . 7
1.4. POST request example: a text type message with replace . . . 8
1.5. POST request example: sending a template . . . 8
1.6. Explanation of parameters for /send . . . 8
1.7. HTTP response to /send when sending single-part rich message (or only SMS) . . . 10
1.8. HTTP response to /send when sending multi-part rich message thread . . . 11
1.9. HTTP response to /send when sending was not successful . . . 11
1.10. Explanation of response to /send . . . 12
1.11. Example Python 3 code . . . 12
2. Idempotency . . . 14
3. Details of how to construct the `richMessage` json object . . . 15
3.1. Structure of the four message types . . . 15
3.1.1. Text messages . . . 15
3.1.2. Media messages . . . 16
3.1.3. Single card messages . . . 17
3.1.4. Carousel messages . . . 18
3.1.5. Available suggestions for the `suggestion` list parameters . . . 21
4. Opting in and out and blocking of numbers . . . 27
5. Callback of receipts and incoming . . . 28
5.1. The `/seturl` endpoint for setting callback webhooks . . . 28
5.1.1. POST request example: setting both endpoints . . . 28
5.1.2. Explanation of parameters for /seturl . . . 29
5.1.3. HTTP response to /seturl . . . 29
5.2. Format of callback to user webhook of status and incoming . . . 30
5.2.1. Callback of status . . . 30
5.2.2. Callback of incoming . . . 31
5.2.3. Explanation of the different incoming message types . . . 33
6. Use of templates . . . 35
6.1. Substitution in templates . . . 35
6.1.1. Example placeholders . . . 36

6.1.2. Example non-placeholders . . . 36
6.2. The `/template/*` endpoints . . . 36
6.2.1. The `/template/get` endpoint . . . 36
6.2.2. The `/template/list` endpoint . . . 38
6.2.3. The `/template/save` endpoint . . . 40
6.2.4. The `/template/delete` endpoint . . . 41
6.3. Answer with template when suggestion button is pressed . . . 42

Introduction

© 2021-2026 LEKAB Communication Systems AB. Version 7.0.36, 2026-11-02.

This is a Web Service API with the primary purpose of sending **Rich Communication Service** (RCS) messages over the **Google RCS for Business (formerly RBM Rich Business Messaging) network**. Currently, the API contains the basic sending functionality, as well as callback notification of status of messages and callback delivery of incoming messages to a user-defined webhook url, and basic handling of message templates for sending pre-composed messages.

There are two main advantages of RCS messages over SMS messages. First, the messages are "**rich**" in the sense that they can also **contain media (image, video, sound)** and certain combinations of media and text known as cards and carousels of cards. They can also contain **suggestion buttons** which can be clicked to generate **replies** or start **actions**, like opening a url, calling a number or opening the map at a position. Secondly, the messages are always sent from a **registered agent**, which has been vetted by Google and the user's operator for use purpose and cannot be easily "spoofed", so that the end-user knows that the message is from a **trusted**, known source.

Rich Communication Service messages according to the Universal Profile 2.x RCS standard are currently supported, and Universal Profile 3.0 standard features will be implemented as they are rolled out by Google RCS for Business.

Similar to SMS, an RCS message is **received in the standard Messages app** and does not need the recipient to download a dedicated app (toggling a setting may be needed). Message reception is today limited to Android phones, but iOS phones are expected to get RCS capability soon. With RCS currently only supported on Android/Google handsets, our service has **fallback functionality to the SMS channel**, where sending goes through our own SMS gateway and not through Google. We have plans to expand to other channels in the future.

User **accounts** and **billing** are **integrated with our SMS gateway**, and we handle communication with Google and facilitate registration of sending agents with the telephone operators.

Each RCS message can only be of one content type. The service supports the message content types offered by Google at the time of writing: a **text** of up to 3000 bytes, one piece of **media**, a single **card** or a **carousel** of 2 to 10 cards. Each message can also have a number (currently up to 11) of **suggestions**.

To achieve **combinations of the RCS rich message types** (e.g. a long text block, then a picture, then another text block), the sequence can be given as a list of messages (a **message thread**). Google does not guarantee the order of messages sent within a short time period, so this API will handle the ordering by first sending the first part of the message thread, and then send the next part only when Google signals a delivery of the first message, and so on, which will mean a short time interval (seconds) between the sending of the parts. There is no limit to the number of parts in a message, other than each part incurring its own cost for the sender (depending on the contract and agent type). In case of fallback to a non-threadable channel (SMS), the entire text is sent as a single message.

If a message is sent with instructions to first try the RCS channel, and **RCS is not available for that agent to that phone, fallback happens**, i.e. the next channel on the list of channels is tried (currently, this will be SMS if present). The user may supply a dedicated **SMS message text which will be sent instead of the RCS message** upon fallback. If an SMS text is not present, the text-type message content will be used, and if that is also not present due to the RCS message not being of the text type, sending will fail. Note that SMS sending requires specification of a sender id (alphanumeric or phone number) or borrowing a sender id from one of our number pools (Two-way SMS).

We also offer automatic **"up-lift" of SMS** messages where a user's messages sent from any of our standard SMS APIs with a specific sender id are sent as RCS text messages from the user's agent, with fallback to the original SMS if RCS did not work, today probably due to an i(ncompatible) Phone. Up-lift setup and terms of sale are outside the scope of this manual.

Google RCS for Business defines **"Basic RCS"** as a text-type message without any suggestion buttons, and with a text that fits into **160 bytes** of UTF-8 encoding. In that universal standard encoding of the Internet, English letters take 1 byte, European umlauts take 2 bytes, Chinese characters 3 bytes and emojis (as well as Ancient Egyptian hieroglyphs) 4 bytes. The main feature of a Basic RCS is that it is delivered at a lower cost than other RCS for Business messages, and is perfect for the kind of applications where an SMS would be used in the old days of yore. The advantage here with RCS is the spoof-proof **sender agent**, which lends greater **trust** to security related messages, for instance, a one-time password login sequence where the message content is a short text string. Recent information from Google RCS for Business indicates that Basic RCS will be discontinued in the near future.

The implementation so far includes the endpoint `/send`, with the obvious purpose, the endpoint `/seturl` which is used to set the urls to which receipts and incoming messages should be forwarded, and the `/template/*` endpoints for handling message templates.

The /send, /seturl and /template/save endpoints support only POST calls where the `HTTP POST` request body is a **JSON** document and returns responses in the `HTTP` response body as JSON documents. **The standard character encoding of JSON is UTF-8**, and the API does not recognize any other encodings (ISO-8859-1, Windows-1252 or other).

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

The /template/list, /template/get and /template/delete endpoints instead support `HTTP GET` calls, with url query parameters. The /template/delete also supports `HTTP DELETE` calls with the same parameters. For `HTTP GET` and `HTTP DELETE` calls, any request body is ignored, but headers can be used for authentication as usual. Output is in a JSON documents in the `HTTP` response body.

Authentication methods available for requests

Every request to the service must include authentication, i.e. username and password (or equivalent). For `POST` requests these can be given in the (automatically `HTTPS = SSL/TLS` encrypted) `HTTP` headers.

We offer four different alternative ways of supplying these login credentials:

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=`, and the header `Authorization` should have the value `Basic dGVzdHVzZXI6dGVzdHBhc3M=` with a single space between the word Basic and the `Base64` token.
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 can be used as a query parameter `key` or as the value of the header `X-API-Key`. The API key is created in the web portal when editing a user. 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.

It is possible to disallow the username+password based authentication methods by setting the role `DISALLOW_BASIC` for the API user, forcing the use of either API-key or Oauth authentication.

If any of the alternative methods of authentication are used, parameter values pertaining to other authentication methods should be omitted or set to the empty string "".

How to obtain an oauth token from our oauth server

The `/auth/api/v1/token` endpoint is used to request an OAuth 2.0 Bearer Token.

POST request examples

Using Basic authentication (username and password in a Base64 from UTF-8 encoded header)

```
curl -X POST --location "https://secure.lekab.com/auth/api/v1/token" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d 'grant_type=client_credentials' \
    --basic --user username:password
```

Using username and password in body (client_id and client_secret url-encoded from UTF-8)

```
curl -X POST --location "https://secure.lekab.com/auth/api/v1/token" \
    -H "Content-Type: application/x-www-form-urlencoded" \
```

```
    -d 'grant_type=client_credentials&client_id=username&client_secret=password'
```

Note that both these uses of username:password authentication are available for the call even if the user has the role `DISALLOW_BASIC` set. We do not support refresh tokens for the oauth server.

Explanation of parameters

POST param	query param value	Description
grant_type	client_credentials (string)	The requested grant type. Only `client_credentials` is supported.
client_id	user name (string)	User name if Basic authentication is not used. Url encoded from utf-8 if necessary
client_secret	user password (string)	Password if Basic authentication is not used. Url encoded from utf-8 if necessary

Chapter 1. The /send endpoint

Used for sending rich messages over the available channels (currently RCS and SMS).

All sending is from the sending user account's **RCS for Business agent**, registered and approved by Google and the telephone operators in the destination country. We will assist in the registration and approval process. We can also assign a test agent (where a small number of phone numbers are pre-registered as allowed destinations) for testing. Terms of sale, contracts, approval, credit check etc. are not covered by this manual. **We will associate the user account with the RCS agent, so the API sender needs only know the login credentials of the sending user**, while we handle all communication with Google.

Channels are specified as a comma (",") separated list starting with the channel to try first. Channels cannot be repeated. Therefore, the currently possible channel specifications are "RCS", "SMS", "RCS,SMS" and "SMS,RCS". It is likely, that **"RCS,SMS"** (RCS with fallback to SMS) and **"RCS"** (RCS without fallback) will be of primary interest to API users.

A message can be addressed to a number of **recipients**, and the channels will be tried as specified for each recipient until a successful sending is achieved. Recipient addresses (addresses are phone numbers for both RCS and SMS) are given **separated by semi-colons (";")**. In the event that a given recipient has different numbers for different channels, and to later accommodate channels that do not use phone numbers, the address to use in each channel can be given separated by commas. If the number of addresses given for a recipient are fewer than the number of channels specified, the last address is repeated (This is the most common case, where the RCS number given is also used for SMS). **Phone numbers must include a country code**. We allow up to 400 recipients per `HTTP POST` call to `/send`, but recommend substantially fewer. Functionality for accommodating large batch sending is under consideration.

While most parameters for the `/send` endpoint need only a short description, the actual rich message is a `JSON` structure with strict rules, and therefore the `richMessage` field (in the top level of the `JSON` document which makes up the body of the `HTTP POST` call) will be explained in its own section below. Note that the API allows the `richMessage` field to be a single RCS message (see below) or a thread of RCS messages to be sent sequentially in order. A thread is given as a JSON list (array) with square brackets "[" and "]" around the list and commas in between the messages.

1.1. POST request example: a text type message without suggestions

```
https://secure.lekab.com/rime/send
```

Sending to one recipient with authentication in the headers and the following as contents of the HTTP body. A typical Basic RCS for Business.

```
{
   "channels" : "RCS",
```

```
   "address" : "46701234567",
   "richMessage" : {
      "text":"Your access key is 12345678"
   }
}
```

1.2. POST request example: single card type message, suggestions in the card and after

```
https://secure.lekab.com/rime/send
```

Sending to two recipients with authentication in the headers and the following as contents of the HTTP body. This example message is not a probable use-case.

```
{
   "channels" : "RCS,SMS",
   "address" : "46701234567;46702345678",
   "smsText" : "This SMS would have been a RCS card message if your phone allowed
it",
   "smsSender" : "MYCOMPANY",
   "richMessage" : {
      "richCard" : {
         "standaloneCard" : {
            "cardOrientation" : "HORIZONTAL",
            "thumbnailImageAlignment" : "LEFT",
            "cardContent" : {
               "title": "A question of zoology",
               "description": "Is this a cat?",
               "media": {
                  "height" : "MEDIUM",
                  "contentInfo": {
                     "fileUrl" :
"https://upload.wikimedia.org/wikipedia/commons/2/2b/Mainecoon1.png"
                  }
               },
               "suggestions": [
                  { "reply" : { "text" : "It is a cat!", "postbackData" : "CAT YES
"} },
                  { "reply" : { "text" : "No, it is not!", "postbackData" : "CAT
NO"} }
               ]
            }
         }
      },
      "suggestions":[
         { "action" : {
            "text" : "Show famous place on map",
```

```
            "postbackData" : "TOUR EIFFEL",
            "viewLocationAction" : { "latLong" : {"latitude" : 48.858093,
"longitude" : 2.294694}, "label" : "Voila"} }
         }
      ]
   }
}
```

1.3. POST request example: thread of one text and two media messages (pictures)

The second picture has two reply suggestion buttons, for answering the question in the text block. Note that every suggestion must belong to one of the message parts in the thread - suggestions cannot live alone as a message part of their own. When no smsText is given, the SMS text is extracted from all text type parts of the message.

```
https://secure.lekab.com/rime/send
```

Sending to two recipients with authentication in the headers and the following as contents of the HTTP body.

```
{
   "channels" : "RCS,SMS",
   "address" : "46701234567;46702345678",
   "smsSender" : "MYCOMPANY",
   "richMessage" : [
      {
         "text":"Please answer about these two pictures: are they cats?"
      },
      {
         "contentInfo": {
            "fileUrl" : "https://pictureserver.com/dogs/Great_Dane.jpg"
         }
      },
      {
         "contentInfo" : {
            "fileUrl" : "https://pictureserver.com/dogs/German_Shepherd.jpg",
         },
         "suggestions": [
           { "reply" : { "text" : "They are cats!", "postbackData" : "CAT YES"} },
           { "reply" : { "text" : "No, they are not!", "postbackData" : "CAT NO"} }
         ]
      }
   ]
}
```

1.4. POST request example: a text type message with replace

```
https://secure.lekab.com/rime/send
```

Sending to one recipient with authentication in the headers and the following as contents of the HTTP body. The message can be re-used, and the substitution parameter varied for every recipient.

```
{
   "channels" : "RCS",
   "address" : "46701234567",
   "richMessage" : {
      "text":"Your access key is {access_key}"
   },
   "replace" : {"access_key":"12345678"}
}
```

1.5. POST request example: sending a template

```
https://secure.lekab.com/rime/send
```

Sending to two recipients with authentication in the headers and the following as contents of the HTTP body. Note that a template may have smsText, but not smsSender, so the required sender id must be given (or smsTwoway requested) if the SMS channel is requested in the channels list.

```
{
   "channels" : "RCS,SMS",
   "address" : "46701234567;46702345678",
   "smsSender" : "MYCOMPANY",
   "templateid" : "@question-card-with-cat"
}
```

1.6. Explanation of parameters for /send

POST json key	json value	
appname	string	Selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.

POST json key	json value	
channels	string	channel try order: "RCS", "SMS", RCS,SMS" or "SMS,RCS"
address	string	Recipient id, for RCS and SMS channels: international phone number (country+area+subscriber) no intl prefix like 00 etc To send to several recipients, separate with semicolons. To use different addresses for different channels, separate with commas within each recipient.
smsText	string	text to use instead of rich message if the SMS channel is used
smsSender	string	SMS sender id, required if not two-way SMS, cannot be reserved for other company
smsTwoway	boolean (true or false)	SMS sender is taken from user's number pool for receiving answers, default false
richMessage	json object or list of json objects, see below	message or thread of messages to send through rich channel, see below
templateid	string	id or "@" + slug of template (see below). The template will replace the smsText and richMessage fields. All {key} present must be replaced
replace	json object/dictionary	{"key1":"value1", "key2":"value2", ...} All {key1} will be replaced by value1 etc. in `richMessage` and `smsText` or in those fields in the template `templateid`
postbackSuffix	string	User defined string that is sent back to agent when any of the suggestion buttons in this message is pressed
trafficType	string	May be required in the future when sending messages via a multi-use agent, to define which use is selected. Currently one of "AUTHENTICATION", "TRANSACTION", "PROMOTION", "SERVICEREQUEST", "ACKNOWLEDGEMENT". Subject to change according to Google RCS for Business.
ttl	integer	Message time to live in minutes, forwarded to Google or SMS operator when sending
conversationid	string	Feature of our SMS service, where receipts and two-way answers to this message are marked with this id
costcenter	string	For billing, used to separate bill for one account into portions e.g. by department

POST json key	json value	
billingid	string	For billing, used to separate bill for one account into portions e.g. by department. Do not set this unless you are certain, only used in some countries.
metadata	string	identifying data not sent to phone but echoed back with status reports and replies
properties	json object/dictionary	{"key1":"value1", "key2":"value2", ...} for future use
Idempotency-Key	HTTP header string	Unique id for rejecting duplicate API calls (see "Idempotency" below)
X-Request-ID	HTTP header string	Unique id id for rejecting duplicate API calls (see "Idempotency" below)

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.

The `replace` parameter is a mapping of placeholders "{key}" to be substituted by text "value" with a dictionary item `"key":"value"`. A key cannot contain any spaces or new lines, or the characters single quote, double quote, colon or left or right curly bracket.

When sending templates, if there are any replace placeholders (of the form {xxx} in the template, it must be substituted by the `replace` parameter. If the `replace` parameter is non-empty, all placeholders in the text (template or not) must be substituted.

If either of the unique id `HTTP` headers is supplied as a non-blank string of up to 255 characters, a second API call with the same id will be rejected. See "Idempotency" below.

1.7. HTTP response to /send when sending single-part rich message (or only SMS)

A successful request will return `200 OK` and a json document of the following format. Note that the request for sending is successful even if the send status of the message sending turns out to be failed. A successfully sent message may sometimes not be delivered due to external factors (phone turned off, phone subscription expired, no such subscriber, phone lacking capability of receiving channel, opt-out from channel), but such requests will be reported as successful by this endpoint, and the later fortune of the message can instead be followed via the delivery status callbacks to the status webhook url. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "resultText" : "At least one message sent OK",
  "sent" : [ {
    "result" : "OK",
    "id" : "1225831041592336384",
```

```
    "channels" : "RCS,SMS",
    "addresses" : "46701234567,46701234567"
  }, {
    "result" : "OK",
    "id" : "1225831041684611072",
    "channels" : "RCS,SMS",
    "addresses" : "46702345678,46702345678"
  } ]
}
```

1.8. HTTP response to /send when sending multi-part rich message thread

A successful request will return `200 OK` and a json document of the following format. Note that the request for sending is successful even if the send status of the message sending turns out to be failed. A successfully sent message may sometimes not be delivered due to external factors (phone turned off, phone subscription expired, no such subscriber, phone lacking capability of receiving channel, opt-out from channel), but such requests will be reported as successful by this endpoint, and the later fortune of the message can instead be followed via the delivery status callbacks to the status webhook url. The `Content-Type` header of the response is `application/json` for all responses.

The difference from the single-part message is the `partid` field, which lists the ids of the parts in order of sending.

```
{
  "resultText" : "At least one message sent OK",
  "sent" : [ {
    "result" : "OK",
    "id" : "1225831041592336384",
    "partids" : "1225831041592336384,1225831041592336387",
    "channels" : "RCS,SMS",
    "addresses" : "46701234567,46701234567"
  }, {
    "result" : "OK",
    "id" : "1225831041684611072",
    "partids" : "1225831041684611072,1225831041684611076",
    "channels" : "RCS,SMS",
    "addresses" : "46702345678,46702345678"
  } ]
}
```

1.9. HTTP response to /send when sending was not successful

An unsuccessful request will give an appropriate `4xx` or `5xx HTTP` result code, and a json

document of the following form. Any result which is not "OK" is an error and means no message was sent.

```
{
  "result" : "ERROR",
  "error" : "Unauthorized"
}
```

In this example, the result code is `401`, as appropriate for "Unauthorized". Syntax errors in the message specification will yield `400` and "Validation error".

1.10. Explanation of response to /send

POST json key	json value (strings quoted)	
resultText	string	"At least one message sent OK" if successful
sent	json list	One list element per recipient
result	string	OK if successful
id	string	id of the message used for future references to this message, e.g. status callbacks
partids	string	ids of the message parts in a message thread, used for future references to this message, e.g. status callbacks. Only present for multi-part threads. Ids are listed in order of sending
channels	string	Comma separated channels to try sending this message
addresses	string	Comma separated addresses to use for each channel
error	string	error description if not successful

1.11. Example Python 3 code

```
import json
import requests
import base64

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

sendreq = {}
sendreq["channels"] = "RCS,SMS"
sendreq["address"] = "46701234567;46702345678"
sendreq["richMessage"] = {
```

```
   "text":"Can you work extra hours on Christmas Day?",
   "suggestions":[
      { "reply" : { "text" : "Yes" } },
      { "reply" : { "text" : "No" } }
   ]
}
sendreq["metadata"] = 'Xmas 2026 employee 1234'
sendreq_json = json.dumps(sendreq)

url = 'https://secure.lekab.com/rime/send'
response = requests.post(url, data=sendreq_json, headers=headers)
sendresp = response.json()
if sendresp["sent"]:
    for m in sendresp["sent"]:
        print("Successful message id "+ m['id'] + " to " + m['addresses'] + " via "
+  m['channels'])
else:
    print("Send error: " + sendresp["error"])
```

will output

```
Successful message id 1225831041592336384 to 46701234567,46701234567 via RCS,SMS
Successful message id 1225831041684611072 to 46702345678,46702345678 via RCS,SMS
```

Chapter 2. 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 sending, **two calls to this API, even with the same text to the same number, will result in sending two messages** (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 `/send` endpoint are **not automatically idempotent**.

If there are any doubts whether a certain message 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 send 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.

There is no check on the message content, always allowing conversations to repeat earlier messages without them being suppressed. Even the SMS duplicate suppression user role, available in some of our other APIs, will not work with this API, in order to allow repeat messages.

The full responsibility for using unique ids is 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.

Chapter 3. Details of how to construct the `richMessage` json object

The rules for RCS Business Messages say that each message must have exactly one content type. This service supports the message content types offered by Google at the time of writing: a **text** of up to 3000 bytes, one piece of **media**, a single **card** or a **carousel** of 2 to 10 cards. Each message can also have a number (currently up to 11) of **suggestions**.

This means that a `richMessage` object can have a maximum of two top level fields, where one gives the content type of the message. This field may be **"text"** for a text message, **"contentInfo"** for a media message or **"richCard"** for a message with a single **card** or **carousel** of cards. The optional second field **"suggestions"** is a list of json objects where each object is a suggestion, i.e. a **button to press** to **reply** to the sending agent with a pre-determined text or to **start an action** on the recipient's phone, like opening the browser to a url or opening the phone calling app with a phone number filled in. The top level list of suggestions can have a maximum of 11 suggestions, but for most RCS-compatible mobile handsets, we do not recommend more than about 3-4 suggestions due to graphical awkwardness of longer lists. The maximum of 11 suggestions does not include suggestions inside cards, where each card can have up to 4 suggestions. Messages containing no content type but only a sugggestions list are not allowed by the standard.

This API extends the Google RCS for business possibilities by also allowing the `richMessage` object to be a JSON list (JSON array) of RCS message objects, to be sent in sequence as a **message thread**. When sending a message thread, each part is only sent when a report of successful delivery for the preceding part is received by the API, so a short delay will be expected between parts. No action is needed after sending from the API user for sending the messages in a thread in this way, it is all handled by the API functionality. In a message thread, each object in the list must adhere to the rules for single messages, and there can be no other entries in the list. For instance, if a list of suggestions is required, that list must be inside one of the objects.

3.1. Structure of the four message types

3.1.1. Text messages

A text message consists of a text of up to 3000 bytes, and an (optional) list of suggestions. If the message is 160 bytes or shorter and there are no suggestions, it is sent at a lower cost, so called "Basic RCS", if it is longer or has at least one suggestion, the standard message price applies. Note that the text can contain any UTF-8/Unicode characters, including emojis, but the current RCS message standard (Universal Profile 2.x) does not include any formatting (often, confusingly, called "rich text") for specifying font, bold, italic or underline.

```
{
   "text":"Yes or no, that is the question?",
   "suggestions":[
      { "reply" : { "text" : "👍 YES", "postbackData" : "YES"} },
      { "reply" : { "text" : "NO 👎", "postbackData" : "NO"} }
```

```
    ]
}
```

Fields and subfields:

json key	json value	
text	string	Message text UTF-8, max 3000 bytes (Basic RCS 160 bytes)
suggestions	Optional JSON list of replies or actions or mixed	0 to 11 suggestions, see below

3.1.2. Media messages

A media message contains a url reference to a server where the piece of media is served. A local media server will be added to this API in the near future. The RCS for Business standard actually includes three different ways of specifying a media message, but two of them refer to messages that are pre-uploaded to Google, and we do not support that uploading process. Therefore, the media url will be contained in a `contentInfo` object. In addition, an (optional) list of suggestion buttons can be included, but no title text or description text. For such combination messages, the card types are appropriate.

The media in a media file may be a picture (jpeg, png etc.), a sound file (mp3 etc.), a video clip (mp4, etc.) or a PDF file. See separate documentation for allowed types, data and pixel sizes and other limitations. Note that while RCS sending is fully encrypted, Google can read all text and all files sent, with no way of sending files that are encrypted so that Google cannot read them.

```
{
   "contentInfo" : {
      "fileUrl" : "https://pictureserver.com/dogs/German_Shepherd.jpg",
      "thumbnailUrl" :
"https://pictureserver.com/dogs/thumbnails/German_Shepherd_tn.jpg",
      "forceRefresh" : true;
   },
   "suggestions":[
      ...
   ]
}
```

Fields and subfields:

json key	json value	
contentInfo	json object	Contains the media reference
fileUrl	string	Url where the media file is served
thumbnailUrl	string	Optional url where a thumbnail file is served

json key	json value	
forceRefresh	boolean	Default false, instruct Google to reload the media file if cached from before
suggestions	Optional JSON list of replies or actions or mixed	0 to 11 suggestions, see below

3.1.3. Single card messages

A single card message contains a single card, and an optional list of suggestions. The card can have a title string of up to x characters, a description text of up to y characters, a piece of media specified as a url in a `contentInfo` object, and an (optional) list of up to 4 internal suggestions inside the card. Depending on the size and type of the media, it is usually displayed in the card as a thumbnail representation which can be clicked to display the media in full-window mode. Exact display geometry is a function of the Messages app version on the phone and is also subject to change.

```
{
   "richCard" : {
      "standaloneCard" : {
         "cardOrientation" : "HORIZONTAL",
         "thumbnailImageAlignment" : "LEFT",
         "cardContent" : {
            "title": "A nice picture",
            "description": "Is this a cat?",
            "media": {
               "height" : "MEDIUM",
               "contentInfo": {
                  "fileUrl" :
"https://upload.wikimedia.org/wikipedia/commons/2/2b/Mainecoon1.png"
               }
            },
            "suggestions": [
               { "reply" : { "text" : "It is a cat!", "postbackData" : "CAT YES"}
},
               { "reply" : { "text" : "No, it is not!", "postbackData" : "CAT NO"}
}
            ]
         }
      }
   },
   "suggestions":[
      ...
   ]
}
```

Fields and subfields:

json key	json value	
richCard	json object	Wraps the card or carousel
standaloneCard	json object	Wraps the single card
cardOrientation	string	Geometry of card display: "HORIZONTAL" or "VERTICAL"
thumbnailImage Alignment	string	Where in the card is media thumbnail: "LEFT" or "RIGHT"
cardContent	json object	Wraps the content of one card (the only one here)
title	string	Title of the card
description	string	Text of the card
media	json object	Wraps media part of content of this card
height	string	Media height "SHORT" (112 DP), "MEDIUM" (168) or "TALL" (264)
contentInfo	json object	Wraps the media urls like in a media message
fileUrl	string	Url where the media file is served
thumbnailUrl	string	Optional url where a thumbnail file is served
forceRefresh	boolean	Default false, instruct Google to reload the media file if cached from before
suggestions	In-card JSON list of replies or actions or mixed	0 to 4 suggestions, see below
suggestions	Optional JSON list of replies or actions or mixed	0 to 11 suggestions, see below

3.1.4. Carousel messages

A carousel message contains 2-10 cards which can have the same types of content as the card in the single card message, as well as an (optional) list of suggestions. The carousel displays the cards in a scrollable fashion so that one of the cards is "on top" and visible. Depending on the size and type of the media, it is usually displayed in the card as a thumbnail representation which can be clicked to display the media in full-window mode. Exact display geometry is a function of the Messages app version on the phone and is also subject to change. The Lovecraft-inspired example below has two cards in the list.

```
{
   "richCard" : {
      "carouselCard" : {
         "cardWidth" : "MEDIUM",
         "cardContents" : [
            {
```

```
                "title":"In picturesque Arkham",
                "description":"Cthulhu Hotel service on call",
                "media":{
                   "height" : "MEDIUM",
                   "contentInfo" : {
                      "fileUrl":
"https://upload.wikimedia.org/wikipedia/commons/6/63/Octopus_marginatus.jpg"
                   }
                },
                "suggestions":[
                   {
                      "action" : {
                         "text" : "Book a room",
                         "postbackData" : "BOOK CTHULHU",
                         "openUrlAction" : { "url" :
"https://creepyhotels.com/cthulhu" }
                      }
                   },
                   {
                      "action" : {
                         "text" : "Call of Cthulhu",
                         "postbackData" : "CALL CTHULHU",
                         "dialAction" : { "phoneNumber" : "+46701234567" } # the ca
lled phone does NOT have to be Android
                      }
                   }
                ]
             },
             {
                "title":"Close to Miskatonic U",
                "description":"Tentacle Inn creature comforts",
                "media":{
                   "height" : "MEDIUM",
                   "contentInfo" : {
                      "fileUrl":
"https://upload.wikimedia.org/wikipedia/commons/d/dd/Loligo_vulgaris.jpg"
                   }
                },
                "suggestions":[
                   {
                      "action" : {
                         "text" : "Book a room",
                         "postbackData" : "BOOK TENTACLE",
                         "openUrlAction" : { "url" :
"https://creepyhotels.com/tentacle" }
                      }
                   },
                   {
                      "action" : {
                         "text" : "Call the hotel",
                         "postbackData" : "CALL TENTACLE",
```

```
                        "dialAction" : { "phoneNumber" : "+46702345678" } # the ca
lled phone does NOT have to be Android
                     }
                  }
               ]
            }
         ]
      }
   },
   "suggestions" : [
      {
        "reply" : { "text" : "Andover Area", "postbackData" : "MENU ANDOVER"}
      },
      {
        "reply" : { "text" : "Haverhill Area", "postbackData" : "MENU HAVERHILL"}
      },
      {
        "reply" : { "text" : "Salem Area", "postbackData" : "MENU SALEM"}
      }
   ]
}
```

Fields and subfields:

json key	**json value**	
richCard	json object	Wraps the card or carousel
carouselCard	json object	Wraps the carousel
cardWidth	string	Geometry of card display: "SMALL" or "MEDIUM"
cardContents	list of json objects	List of contents of 2-10 cards in carousel
title	string	Url where the media file is served
description	string	Url where the media file is served
media	json object	Wraps media part of content of this card
height	string	Media height "SHORT" (112 DP), "MEDIUM" (168) or "TALL" (264), where TALL is not allowed for carousels with card width SMALL
contentInfo	json object	Wraps the media urls like in a media message
fileUrl	string	Url where the media file is served
thumbnailUrl	string	Optional url where a thumbnail file is served
forceRefresh	boolean	Default false, instruct Google to reload the media file if cached from before

json key	json value	
suggestions	In-card JSON list of replies or actions or mixed	0 to 4 suggestions, see below
suggestions	Optional JSON list of replies or actions or mixed	0 to 11 suggestions, see below

3.1.5. Available suggestions for the `suggestion` list parameters

Suggestions can be either of type `reply` or of type `action`, and can be mixed as desired within a suggestions list. A message has a suggestion list of 0 to 11 suggestions at the end of the message, while each card contains a suggestion list of 0 to 4 suggestions. When a user selects a suggestion, an incoming message is sent to the sender agent, where it is stored in our database and forwarded to the sender account by callback to a webhook url (under implementation). The `postbackData` field may be a template call, resulting in a reply sendout to the phone user (see templates section below).

Suggestion `reply`

Sends an answering message with a preset content to the sender agent.

```
{
   "reply" : {
      "text" : "Salem Area",
      "postbackData" : "MENU SALEM"
   }
}
```

Fields and subfields:

json key	json value	
reply	json object	Wraps the reply
text	string	Button text of the reply suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below).

Suggestion `dialAction`

Opens the telephone app on the recipient's mobile handset with a given number pre-filled. The phone to call does not have to be Android or even a mobile phone. Also informs the sending

agent with an incoming action message.

All actions have the button `text` field, the `postbackData` field and the `fallbackUrl` field where the action is replaced with opening the mobile handset browser at the given url if the mobile handset does not support the originally intended action. Probably the fallbackUrl will not be of much use, since most actions are things mobile handsets usually do. In addition each action has its own `xxxAction` field defining what action it is and containing the parameters for the specific action.

```
{
   "action" : {
       "text" : "✊ Call Erik",
       "dialAction" : { "phoneNumber" : "+46701234567" }
   }
}
```

Fields and subfields:

json key	json value	
action	json object	Wraps an action suggestion
text	string	Button text of the action suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below)
fallbackUrl	string	Url to open if mobile phone does not support this action (unlikely)
dialAction	json object	Wraps a dialAction suggestion
phoneNumber	string	Number to present in the dialing app

Suggestion `viewLocationAction`

Opens the user's default map app and selects the specified location. Also informs the sending agent with an incoming action message.

All actions have the button `text` field, the `postbackData` field and the `fallbackUrl` field where the action is replaced with opening the mobile handset browser at the given url if the mobile handset does not support the originally intended action. Probably the fallbackUrl will not be of much use, since most actions are things mobile handsets usually do. In addition each action has its own `xxxAction` field defining what action it is and containing the parameters for the specific action.

```
{
   "action" : {
      "text" : "Famous place",
```

```
      "postbackData" : "TOUR EIFFEL",
      "viewLocationAction" : {
         "latLong" : {
            "latitude" : 48.858093,
            "longitude" : 2.294694
         },
         "label" : "Voila"
      }
   }
}
```

Fields and subfields:

json key	**json value**	
action	json object	Wraps an action suggestion
text	string	Button text of the action suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below)
fallbackUrl	string	Url to open if mobile phone does not support this action (unlikely)
viewLocationAction	json object	Wraps a viewLocationAction suggestion
latLong	json object	Wraps a latitude+longitude object
latitude	float	Latitude -90.0000 to +90.0000 Southern hemisphere is negative
longitude	float	Longitude -180.0000 to +180.0000 West of Greenwich is negative
label	string	Map marker label

Suggestion `createCalendarEventAction`

Opens user's default calendar app and starts the new calendar event flow with the event data pre-filled. Also informs the sending agent with an incoming action message.

All actions have the button `text` field, the `postbackData` field and the `fallbackUrl` field where the action is replaced with opening the mobile handset browser at the given url if the mobile handset does not support the originally intended action. Probably the fallbackUrl will not be of much use, since most actions are things mobile handsets usually do. In addition each action has its own `xxxAction` field defining what action it is and containing the parameters for the specific action.

```
{
```

```
    "action" : {
       "text" : "Save the date",
       "createCalendarEventAction" : {
          "startTime": "2026-04-30T17:00:00Z",
          "endTime": "2026-04-30T21:00:00Z",
          "title": "Valpurgis night bonfire",
          "description": "A good time will be had by all"
       }
    }
 }
```

Fields and subfields:

json key	json value	
action	json object	Wraps an action suggestion
text	string	Button text of the action suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below)
fallbackUrl	string	Url to open if mobile phone does not support this action (unlikely)
createCalendarEventAction	json object	Wraps a calendarEventAction suggestion
startTime	string	Timestamp in RFC3339 UTC "Zulu" format, with nanosecond resolution and up to nine fractional digits. Examples: "2014-10-02T15:01:23Z" and "2014-10-02T15:01:23.045123456Z"
endTime	string	Timestamp in RFC3339 UTC "Zulu" format, as startTime
title	string	Event title
description	string	Event description

Suggestion `openUrlAction`

Opens the user's default web browser app to the given URL. Also informs the sending agent with an incoming action message.

All actions have the button `text` field, the `postbackData` field and the `fallbackUrl` field where the action is replaced with opening the mobile handset browser at the given url if the mobile handset does not support the originally intended action. Probably the fallbackUrl will not be of much use, since most actions are things mobile handsets usually do. In addition each action has its own `xxxAction` field defining what action it is and containing the parameters for the specific action.

```
{
```

```
   "action" : {
      "text" : "Book the hotel",
      "postbackData" : "BOOK CTHULHU",
      "openUrlAction" : {
         "url" : "https://creepyhotels.com/cthulhu"
      }
   }
}
```

Fields and subfields:

json key	json value	
action	json object	Wraps an action suggestion
text	string	Button text of the action suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below)
fallbackUrl	string	Url to open if mobile phone does not support this action (absurdly unlikely)
openUrlAction	json object	Wraps an openUrlAction suggestion
url	string	Url to open in browser

Suggestion `shareLocationAction`

Opens the RCS app's location chooser so the user can pick a location to send to the agent. Also informs the sending agent with an incoming action message, when the suggestion button is pressed. The location data arrives as a second incoming message of type LOCATION, when the user shares it.

All actions have the button `text` field, the `postbackData` field and the `fallbackUrl` field where the action is replaced with opening the mobile handset browser at the given url if the mobile handset does not support the originally intended action. Probably the fallbackUrl will not be of much use, since most actions are things mobile handsets usually do. In addition each action has its own `xxxAction` field defining what action it is and containing the parameters for the specific action.

```
{
   "action" : {
      "text" : "Where are you?",
      "shareLocationAction" : { }
   }
}
```

Fields and subfields:

json key	json value	
action	json object	Wraps an action suggestion
text	string	Button text of the action suggestion. Also incoming message text if postbackData is empty
postbackData	string	Incoming message text if present. The postback data may be a template call, resulting in a reply sendout to the phone user (see templates section below).
fallbackUrl	string	Url to open if mobile phone does not support this action (absurdly unlikely)
shareLocationAction	json object	Wraps an openUrlAction suggestion, has no parameters

Chapter 4. Opting in and out and blocking of numbers

The phone user can always send in a text message to the service with the text "**STOP**" to get added to the blocklist of that specific sender account. If the user later sends in a text message with the text "**START**", the blocking is removed.

Google RCS for Business has announced that there will be unsubscribe and resubscribe buttons in the Messages app to achieve the same purpose, but currently any use of such buttons is not forwarded to our system in the Google RCS for Business announced format (or any other format), so we expect that in later versions. The unsubscribe and resubscribe buttons in the current version seem to affect settings inside Google RCS for Business which we currently cannot control or access. So a customer that unsubscribes with a Google button cannot yet resubscribe by sending in "START", but will need to resubscribe with the corresponding Google button.

Opting in and out does not affect our security-related blocking of country codes to which an account cannot send. These blocking rules will be set up so that sending is only attempted to countries where the sending RCS agent is registered, verified and launched. This by default includes the SMS channel, but user accounts could possibly relax the country code blocking rules, to achieve fallback to SMS when RCS sending fails to a foreign country.

Chapter 5. Callback of receipts and incoming

In order to track status of sent messages and receive incoming responses, the API user can set up notification of these events through callback (pushing, event-driven reporting) to a user-specified callback URL (webhook) where the user's system is listening for `HTTP` requests. The user's system must respond with a `200 OK` (or similar) `HTTP` response, or we will try to resend the callback.

When our system receives a status receipt from the sending channel, and that receipt does not imply trying another channel for sending, we forward the status receipt to the webhook URL set by the API user. Also, if an incoming message arrives to the RCS agent or to an SMS two-way number, we forward the incoming message to another webhook URL (which may, of course be the same URL as for receipts).

There is currently only one callback notification format, called "JSON". The user can set callback type "JSON" for receipts and "NONE" for incoming, or vice versa, with the expected result. There is an `HTTP POST` API endpoint for setting the webhook urls.

5.1. The `/seturl` endpoint for setting callback webhooks

The endpoint must be called with authentication for the user account that owns the agent. This will be the same authentication as for sending from the agent with the present API. One account is not allowed to set webhooks for another account.

Currently, the endpoint writes the new URL directly in the database, and for every callback event, the database is read at that time. This is subject to change when API traffic increases, when some kind of caching of callback URLs will probably be implemented. We discourage depending on instant URL changes. To distinguish between different message batches, we offer the `metadata` and `conversationid` fields in the sent message, which will be returned with the receipt, and with incoming responses which are estimated to be answers to a given message.

5.1.1. POST request example: setting both endpoints

```
https://secure.lekab.com/rime/seturl
```

The example call sets two urls, one for receiving receipts and another for receiving incoming responses.

```
{
  "receipttype" : "JSON",
  "receipturl" : "https://mycompany.com/mess/receipts?receiptkey=klm765",
  "receiptheaders": "{\"Importantheader\":\"Hello\"}",
  "incomingtype" : "JSON",
```

```
  "incomingurl" : "https://mycompany.com/mess/incoming?incomingkey=abc123"
}
```

5.1.2. Explanation of parameters for /seturl

POST json key	json value	
appname	string	Selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.
receipttype	string	The output format for receipt callback. "JSON" or "NONE" or null/not set for no change
receipturl	string	The webhook url for receipt callback. Can have url query parameters e.g. for authentication
receiptheaders	string of JSON	JSON string containing Map of String to String, sent as headers e.g. for authentication, only set when url is set
incomingtype	string	The output format for incoming callback. "JSON" or "NONE" or null/not set for no change
incomingurl	string	The webhook url for receipt callback. Can have url query parameters e.g. for authentication
incomingheaders	string of JSON	JSON string containing Map of String to String, sent as headers e.g. for authentication, only set when url is set

5.1.3. HTTP response to /seturl

A successful request will return `200 OK` and a json document of the following format, containing the current status of the callback webhook URLs for the user. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "userid" : "12345",
  "appname" : "",
  "receipttype" : "JSON",
  "receipturl" : "https://mycompany.com/mess/receipts?receiptkey=klm765",
  "receiptheaders" : "{\"Importantheader\":\"Hello\"}",
  "incomingtype" : "JSON",
  "incomingurl" : "https://mycompany.com/mess/incoming?incomingkey=abc123"
}
```

5.2. Format of callback to user webhook of status and incoming

The only callback type choice that results in any callback is "JSON".

5.2.1. Callback of status

For RCS messages, only three statuses can happen: `SENDFAIL`, `DELIVERED` and `READ`. The `SENDFAIL` status is only called back for the last channel in the list of channels to try. Note that message threads are invisible to the status callback functionality, so a three-part thread of messages to one recipient will generate a `DELIVERED` and a `READ` status callback for each of the parts. SMS messages can receive other failing statuses from the operators, summed into the status `UNDELIVERABLE` that likewise is only called back if SMS is the last channel to try.

Example callback of status DELIVERED


```
{
  "id": "1249385535177367552",
  "userid": "12345",
  "appname": "",
  "time": "2025-09-08T13:32:24Z",
  "channel": "RCS",
  "from": "mycompany_abcdabcd_agent",
  "to": "46701234567",
  "status": "DELIVERED",
  "metadata": "This is the metadata",
  "conversationid": "ABC123"
}
```


Example callback of status READ


```
{
  "id": "1249385535177367552",
  "userid": "12345",
  "appname": "",
  "time": "2025-09-08T13:32:30Z",
  "channel": "RCS",
  "from": "mycompany_abcdabcd_agent",
  "to": "46701234567",
  "status": "READ",
  "metadata": "This is the metadata",
  "conversationid": "ABC123"
}
```

Explanation of fields in status receipt callback

json key	json value	
id	string of long integer	The id of the message, same as was returned when sending
userid	string of long integer	The id of the user account owning the message
appname	string	Selects one of several sender agents for a user account. Will be the default appname "" if we have not supplied an appname to use.
time	string	Date and time in UTC/Greenwich/Zulu zone, ISO 8601 format with second precision, always ends in Z
channel	string	Channel which the status receipt refers to: RCS or SMS
from	string	Sender agent for RCS, sender id for SMS
to	string	Address (phone number) to which the message was sent in the channel
status	string	One of DELIVERED, READ, SENDFAIL, UNDELIVERABLE
statusText	string	Sometimes explanation of failed status (not always present, depends on what we know)
metadata	string	User-defined data associated with the sent message, not sent to/from Google RCS for Business or SMS operator
conversationid	string	(Legacy) Short user-defined data (up to 255 chars) associated with the sent message, not sent to/from Google RCS for Business or SMS operator

5.2.2. Callback of incoming

Incoming messages are generated when the recipient of a message uses one of the suggested reply- or action-buttons in the message, or when the user responds with a text, media item or location in the conversation with the agent in the Messages app on the mobile device. Google RCS for Business services are expected to add buttons that generate opt-out (STOP) and opt-in (START) messages, but we have not yet seen such buttons or messages, so users for now have to generate them themselves by sending text messages containing the word "STOP" or "START".

There are currently 8 different types of incoming messages. The incoming message type is found in the field `"type"`, and will be one of `REPLY`, `ACTION`, `STOP`, `START`, `TEXT`, `USERFILE`, `LOCATION`, `SMS`. Each of the types will have all the general fields `id`, `userid`, `appname`, `time`, `channel`, `from`, `to`, and `type`. If we can make a reasonable guess at which message this message is in response to (either from pressing a suggestion button which transmits that info in the background, or from guessing the last message of the conversation), we will also give that message's id (`responseto`), metadata (`resptometadata`) and conversationid (`resptoconvid`).

There is no concept of threads of messages for incoming RCS messages. If the user responds with a text and then a picture, two incoming messages will be generated.

Example callback of incoming of type `REPLY`

```
{
  "id": "1249385655331594240",
  "userid": "12345",
  "appname": "",
  "time": "2025-09-08T13:32:32Z",
  "channel": "RCS",
  "from": "46701234567",
  "to": "mycompany_abcdabcd_agent",
  "responseto": "1249385535177367552",
  "type": "REPLY",
  "text": "ABCD",
  "buttontext": "YES",
  "resptometadata": "This is the metadata",
  "resptoconvid": "ABC123"
}
```

Explanation of fields in incoming message callback

json key	json value	
id	string of long integer	The id of the message in our incoming message table
userid	string of long integer	The id of the user account owning the incoming message
appname	string	Selects one of several agents for a user account. Will be the default appname "" if we have not supplied an appname to use.
time	string	Date and time in UTC/Greenwich/Zulu zone, ISO 8601 format with second precision, always ends in Z
channel	string	Channel which the status receipt refers to: RCS or SMS
from	string	Address (phone number) from which the message was received in the channel
to	string	Receiving agent for RCS, incoming number for SMS
responseto	string	Best guess of which sent message id this is a response to, always correct for buttons
type	string	One of `REPLY`, `ACTION`, `STOP`, `START`, `TEXT`, `USERFILE`, `LOCATION`, `SMS`

json key	json value	
text	string	Incoming message for most message types, for buttons the postbackData specified when sending (see below)
buttontext	string	Text on reply or action button that was pressed to generate the message (see below)
filepayload	string	Json object with data (url, size etc) of media file sent in by the user (see below)
filethumb	string	Json object with data (url, size etc) of media file thumbnail usually generated by Google RCS for Business (see below)
locationlatlong	string	e.g. 59.606927;16.567048 latitude and longitude semicolon separated in degrees with decimals (see below)
postbacksuffix	string	Additional text that is sent back from every button in message, defined when sending
resptometadata	string	User-defined data associated with the sent message to which this is probably a response, not sent to/from Google RCS for Business or SMS operator
resptoconvid	string	(Legacy) Short user-defined data (up to 255 chars) associated with the sent message to which this is probably a response, not sent to/from Google RCS for Business or SMS operator

5.2.3. Explanation of the different incoming message types

- The `REPLY` message comes from the phone user pressing a reply suggestion button. The `text` field value will be the postback data specified when sending. The button text is also given.
- The `ACTION` message comes from the phone user pressing an action suggestion button, for displaying a url in the browser, setting up a call to a phone number, displaying coordinates on a map, etc. The `text` field value will be the postback data specified when sending. The button text is also given. Note that Google RCS for Business does not supply the exact variety of action in the incoming message, so if that is specifically needed, it should be included in the postback message when sending.
- The `STOP` message arises when the phone user sends a text message with the text "STOP" (any case of letters). It should also come when the phone user presses an `Unsubscribe` button, but that is, apparently, not yet implemented by Google RCS for Business.
- The `START` message arises when the phone user sends a text message with the text "START" (any case of letters). It should also come when the phone user presses a `Subscribe` button, but that is, apparently, not yet implemented by Google RCS for Business.
- When the RCS user writes a text in the chat, a `TEXT` incoming message is generated. The `text` field contains the text.

- When the RCS user sends in a media file (picture, video, sound), this gives a `USERFILE` incoming message. The incoming file will be stored in a Google RCS for Business public server, and a url pointing to the file will be found inside the json object of the `filepayload` field while the url to the Google RCS for Business generated thumbnail of the incoming file will be found in the `filethumb` field. We are planning to file away the incoming media from Google to keep it safe and easy to refer to in our own media server, when that is implemented. If the phone user sends both a text and a media file, two messages will be received by the agent. Typical content of the `USERFILE filepayload`, where the `fileUri` can be called to retrieve the media:

```
{"mimeType":"image/jpeg","fileSizeBytes":422754,"fileUri":"https://rcs-copper-
eu.googleapis.com/blob/75b23925-54d6-7654-9515-
e5127833eeb1/J7IO2A14B9PMG_0U7WegoGUqOmtX4f8UWEljx3Lo?ct=aW1hZ2bulnBlZw","fileName"
:"720332000011178328.jpg"}
```

- The `shareLocation` action will generate an `ACTION` incoming message when pressed, and when the user subsequently submits a position (probably from the phone's GPS), a `LOCATION` incoming message with the given location will be generated. The `LOCATION` message `locationlatlong` field will be a semicolon separated string with the latitude in degrees (with decimals) in the interval -90 to 90 degrees (south is negative), and the longitude in degrees (with decimals) in the interval -180 to 180 degrees (west is negative). Example : "59.327402;18.055316".
- If the messages was transmitted through the SMS channel and the phone user responds to the sender number (or uses a reserved incoming SMS number), an `SMS` incoming message will be generated. The SMS message text will be found in the `text` field.

Parameter fields occuring in each type of incoming message

parameter	REPLY	ACTION	STOP	START	TEXT	USERFILE	LOCATION	SMS
text	x	x	x("STOP")	x("START")	x			x
buttontext	x	x						
postbacksuffix	x	x						
filepayload						x		
filethumb						x		
locationlatlong							x	

Chapter 6. Use of templates

Templates are **pre-prepared message contents** (rich message and SMS text) for use when sending messages with this API. As an example, this allows a content specialist to prepare e.g. a marketing message in advance, and at a later point a market specialist selects an appropriate list of recipients to which this message should be sent. A template can only be read and used in the API by its owner account. Templates can be stored with the /template/* endpoints (below) or edited and stored using a graphical web application tentatively named "RCS Conversations" on which work is in progress.

A template always has a system-generated `id` (something like "1257673581400342522") in the database, and this id can be used to refer to the template. It is also possible to give a, so called, `slug` (like "marketing-xmas-2026") to use when referring to the template. This is more readable, and when a template is saved with the same slug, the message is replaced. Therefore a given slug will refer to the latest version of the template.

In the `/send` endpoint of this API, a template is called by setting the `templateid` parameter to either a template id:

```
  "templateid": "1257673581400342522",
```

or a slug preceded by a "@" character:

```
  "templateid": "@marketing-xmas-2026",
```

There is no associated recipient list or SMS sender id inside a template, so it can be reused for multiple recipients and campaigns. When the SMS channel is used, a sender id (or a twoway SMS flag) must be added separately.

If the template contains a multi-part rich message **thread**, that will be handled just like when such a thread is given in the `/send` endpoint `richMessage` field, see above, and the parts will be sent in the given sequence without any additional input from the user.

A further use of templates is for **automatically responding with a message** when the recipient selects a reply suggestion or action suggestion **button**. This is specified in the `postbackData` field of the suggestion, see below. Note that the template started in this way will not be be substituted by the replace parameter, and should therefore not contain placeholders.

6.1. Substitution in templates

Substitution of `placeholders` in the template text (`smsText` and `richMessage`) with personalized content is available using the `replace` parameter. Note that substitution in the `richMessage` is done by substituting in the json serialization string of the message, leaving open the possibility to substitute in e.g. suggestion button text or postback data, media urls etc. If the substitution breaks the json structure, the message will not be sent.

Later, retrieval of such personalized content from e.g. the recipient's contact in the address book is planned.

It is not allowed to send a template where an unsubstituted placeholder remains.

A placeholder has the form "{abcd}" where abcd can be any characters, except right or left curly brackets, single or double quotes, colon or any space-like characters (space, tab, new line).

6.1.1. Example placeholders

```
{name}
{access_key}
{my.little.pony}
```

6.1.2. Example non-placeholders

```
{"name" : "Gustav"}
{access key}
{}
{'hello'}
```

6.2. The `/template/*` endpoints

API users can administer their own templates, using a set of endpoints for retrieving, listing, saving and deleting templates. Work is also in progress on a graphical RCS conversation web application which will allow editing and saving templates for use in the API.

The endpoints must be called with authentication for the user account that owns the agent to be used when sending the template. This will be the same authentication as for sending from the agent with the present API. One account is not allowed to access or save templates belonging to another account.

6.2.1. The `/template/get` endpoint

This endpoint gives the **full data** available regarding a single template, including id, slug, name, owner id, owner appname, richMessage object, smsText and creation/modification history.

GET request example getting a template with a given id

```
https://secure.lekab.com/rime/template/get?I=1257673581400342522
```

With authorization in headers or as an additional `&key=xxxx` apikey query parameter (see beginning of document)

GET request example getting a template with a given slug

```
https://secure.lekab.com/rime/template/get?S=marketing-xmas-2026
```

With authorization in headers or as an additional `&key=xxxx` apikey query parameter (see beginning of document)

Explanation of parameters for /template/get

GET query key	value	
I	string	id : The unique id of the template, string containing a long integer
S	string	slug: A user-given id string, lower case, no spaces for the template. Should not start with "@"
A	string	appname : Selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.

If both id and slug are given, they must correspond to the same template, or no template is found

HTTP response to /template/get

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

```
{
  "id" : "11257673581400342522",
  "slug" : "manual-example-message",
  "name" : "An example message used to illustrate saving templates in the API
manual",
  "userid" : 12345,
  "appname" : "",
  "richMessage" : [ {
     "text":"This is a text RCS message and, as you can see, it is from a trusted
agent"
  } ],
  "smsText" : "You got this SMS message because you cannot receive RCS",
  "createdby" : "testuser",
  "createdtime" : "2025-10-01T10:25:48Z",
  "modifiedby" : "testuser",
  "modifiedtime" : "2025-10-08T10:42:55Z"
}
```

The `modifiedby` and `modifiedtime` will only be present if a message with that slug has been saved

at least twice by the user, corresponding to a modification of the message with that slug. Note that the `richMessage` field is always a list (enclosed by squared bracket), which may have one or more items, depending on the message being a single message or a thread to be sent sequentially when handled internally by the API.

If the template was not found, there will be a `404 Not Found` error and the response body

```
{"error":"Unknown template id"}
```

6.2.2. The `/template/list` endpoint

This endpoint gives a **list of all templates** available to the querying userid and appname. The list can be chosen to **include all data** for every template (id, slug, name, userid, appname, richMessage, smsText, createdby, createdtime, modifiedby, modifiedtime) **or only identifying information** (id, slug, name, userid, appname).

Explanation of parameters for /template/list

GET query key	value	
F	string	full : set to TRUE, T, YES or Y to get full data for every template
A	string	appname : Selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.

HTTP response to /template/list with F=false

A successful request will return `200 OK` and a json document of the following format, containing the a list of all templates saved by the user. If the `F` parameter is set to `false`, the message content and creation history is not given. The `Content-Type` header of the response is `application/json` for all responses.

```
[ {
  "id" : "11257673581400342522",
  "slug" : "manual-example-message",
  "name" : "An example message used to illustrate saving templates in the API
manual",
  "userid" : 12345,
  "appname" : ""
}, {
  "id" : "11257673581420512320",
  "slug" : "second-example-message",
  "name" : "A second example message used to illustrate saving templates in the API
manual",
  "userid" : 12345,
  "appname" : ""
```

```
} ]
```

HTTP response to /template/list with F=true

A successful request will return `200 OK` and a json document of the following format, containing the a list of all templates saved by the user. If the `F` parameter is set to `true` all known data about the templates is given. The `Content-Type` header of the response is `application/json` for all responses.

```
[ {
  "id" : "11257673581400342522",
  "slug" : "manual-example-message",
  "name" : "An example message used to illustrate saving templates in the API
manual",
  "userid" : 12345,
  "appname" : "",
  "richMessage" : [ {
     "text":"This is a text RCS message and, as you can see, it is from a trusted
agent"
  } ],
  "smsText" : "You got this SMS message because you cannot receive RCS",
  "createdby" : "testuser",
  "createdtime" : "2025-10-01T10:25:48Z",
  "modifiedby" : "testuser",
  "modifiedtime" : "2025-10-08T10:42:55Z"
}, {
  "id" : "11257673581420512320",
  "slug" : "second-example-message",
  "name" : "A second example message used to illustrate saving templates in the API
manual",
  "userid" : 12345,
  "appname" : "",
  "richMessage" : [ {
     "text":"This is another text RCS message and, as you can see, it is from a
trusted agent"
  } ],
  "smsText" : "You got this other SMS message because you cannot receive RCS",
  "createdby" : "testuser",
  "createdtime" : "2025-10-01T10:25:48Z",
  "modifiedby" : "testuser",
  "modifiedtime" : "2025-10-08T10:42:55Z"
} ]
```

The `modifiedby` and `modifiedtime` will only be present for a message if a message with that slug has been saved at least twice by the user, corresponding to a modification of the message with that slug. Note that the `richMessage` field is always a list (enclosed by squared bracket), which may have one or more items, depending on the message being a single message or a thread to be sent sequentially when handled internally by the API.

6.2.3. The `/template/save` endpoint

This endpoint is used **to save a template**. If a `slug` is given, and an earlier template with that same slug was saved, the new template overrides the old one. Otherwise, a new template with a new id is created.

POST request example: a minimal template with a text rich message

```
https://secure.lekab.com/rime/template/save
```

With authorization in headers or as an additional `&key=xxxx` query parameter (see beginning of document)

Saving a template with authentication in the headers and the following as contents of the HTTP body. Note that the `richMessage` can be as complicated as any message acceptable for sending to the `/send` endpoint. This extra simple message is just to give a brief example in this manual text.

```
{
   "slug" : "manual-example-message",
   "name" : "An example message used to illustrate saving templates in the API
manual",
   "richMessage" : {
      "text":"This is a text RCS message and, as you can see, it is from a trusted
agent"
   },
   "smsText" : "You got this SMS message because you cannot receive RCS"
}
```

Note that if a template with the same `slug` is already present for this user and appname, the older template will be replaced. Adding a slug to a message without one or changing a slug is not possible. Instead delete the message and save a new one with the correct slug. The `richMessage` field can be given as a single message object (as in the example) or as a square bracket ("[" and "]") enclosed JSON list of message objects to denote a thread of messages to be sent in sequence. A single message is saved as a one-item list internally and will appear as a list when retrieved with the `/template/get` and `/template/list` endpoints.

Explanation of parameters for /template/save

POST json key	json value	
slug	string	short readable id (max 127 characters) that can refer to the template e.g, when sending, lower case letters and numbers, hyphen and underscore, no spaces. If a template with the same `slug` is already present for this user, the older template will be replaced.

POST json key	json value	
name	string	longer descriptive text to be read e.g. when selecting a template from a list or when the slug is too cryptic
appname	string	selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.
smsText	string	text to use instead of rich message if the SMS channel is used
richMessage	json object or list of json objects, see above	message or message thread to send through rich channel, see above

HTTP response to /template/save

A successful request will return `200 OK` and a json document of the following format, containing the result of the saving process with id, slug and name of the saved template. If a template with the same `slug` was already present for this user, the older template was replaced. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "userid" : "12345",
  "appname" : "",
  "id" : "1257673581400342522",
  "slug" : "manual-example-message",
  "name" : "An example message used to illustrate saving templates in the API
manual"
}
```

6.2.4. The `/template/delete` endpoint

This endpoint accepts both `HTTP GET` and `HTTP DELETE` calls, to **delete one template** per call. Only templates owned by the user can be deleted.

GET or DELETE request example deleting a template with a given id

```
https://secure.lekab.com/rime/template/delete?I=1257673581400342522
```

With authorization in headers or as an additional `&key=xxxx` apikey query parameter (see beginning of document)

GET or DELETE request example deleting a template with a given slug

```
https://secure.lekab.com/rime/template/delete?S=marketing-xmas-2026
```

With authorization in headers or as an additional `&key=xxxx` apikey query parameter (see beginning of document)

Explanation of parameters for /template/delete

GET or DELETE query key	value	
I	string	id : The unique id of the template, string containing a long integer
S	string	slug: A user-given id string, lower case, no spaces for the template. Should not start with "@"
A	string	appname : Selects one of several sender agents for a user account. Omit this to use default appname "" if we have not supplied an appname to use.

If both id and slug are given, they must correspond to the same template, or no template to delete is found

HTTP response to /template/delete

A successful request will return `200 OK` and a json document of the following format, containing the result of the deletion process with id, slug and name of the deleted template, and the field `delete` having the value `DELETED`. The `Content-Type` header of the response is `application/json` for all responses.

```
{
  "userid" : "12345",
  "appname" : "",
  "id" : "1257673581400342522",
  "slug" : "manual-example-message",
  "name" : "An example message used to illustrate saving templates in the API
manual",
  "delete" : "DELETED"
}
```

If the template is not present (already deleted or never present), there will be a `404 Not Found` error and the response body

```
{"error":"Unknown template id"}
```

6.3. Answer with template when suggestion button is pressed

A further use of templates is for **automatically responding with a message** when the recipient selects a **reply suggestion or action suggestion button**. This allows setting up a complex

navigable tree of messages that the recipient can peruse. Note that sending messages using this functionality will allow the recipient to generate traffic, possibly associated with a cost, on behalf of the original sender.

This is done in the API by setting the `postbackData` field of the reply suggestion or action suggestion to `TEMPLATE125767358140034252`2 or `TEMPLATE@marketing-xmas-2026`.

If more data is required in the postback data, it can be added after a semicolon: `TEMPLATE125767358140034252`2`;EXTRA DATA ADDED HERE`.

Just as when sending a template message from the api using the `templateid`, the template must exist and be owned by the original message sender user account, at the time the recipient presses the button. There is no check at the time of sending such a template reference in the postbackData, whether the template exists at that time.

If the template contains a rich message thread, being a list of several messages, all parts of the thread will be sent, just like when such a list is given in the /send endpoint.