Lekab Rich Rest Web Service

Lekab Communication Systems AB

Version 5.1.168, 2024-09-02

Lekab Rich Rest Web Service

Introduction

This is a Web Service for sending Rich Communication Service (RCS) messages over different channels, receiving callback of status of messages and receiving callback of incoming messages.

The service supports six types of messages from the Universal Profile 2.0 RCS standard: `text_message`, `media_message`, `choice_message`, `card_message`, `carousel_message` and `location_message`. In addition, full Whatsapp business messaging for sending unsolicited messages to Whatsapp users, using pre-approved message templates, is supported via the `template_message` subtype `whatsapp_template`. Likewise, all-hours business messaging to Kakaotalk users, using pre-approved message templates, is supported via the `template_message` subtype `kakaotalk_template`.

Currently available channels are RCS (via Google Messages), SMS, Whatsapp, Facebook Messenger, Viber bot, Viber Business Messages, Kakaotalk and MMS.

If a message is sent with instructions to first try the RCS channel, and if RCS is not available then try the SMS channel, and RCS indeed fails, there will be automatic fallback transcoding of the message to a form that can be sent via the SMS channel, but the user experience is obviously degraded. For example, a picture will be represented by a URL which can be clicked to see the picture.

Note that fallback between template messages is not supported (e.g. trying first a `whatsapp_template` and if that fails a `kakaotalk_template`).

The implementation so far includes the endpoint `/send`, with the obvious purpose, and subendpoints of the endpoint `/callbackurl` used for controlling callback of delivery status and incoming messages to a service at a webhook url where the sender can receive them. There is furthermore the endpoints `/opt/optin` and `/opt/optout` for registering signups and withdrawals for Whatsapp messages.

The endpoints support only POST calls where the `HTTP POST` request body is a JSON document and return responses in the `HTTP` response body as json documents.

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

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 corresponding fields in the `JSON` document which is sent in the (automatically `HTTPS = SSL/TLS` encrypted) `HTTP` body.

We also offer three different alternative ways of supplying these username and password 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 obtained from Lekab or by self-service in the web portal can be used as a query parameter `key`, as the value of the header `X-API-Key` or as the field `apikey` in the `POST` request body. 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.

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 "".

Chapter 1. The /send endpoint

Used for sending Rich Communication Services messages over the available channels. Currently, "RCS", "SMS", "WHATSAPP", "MESSENGER", "VIBER" (Bot), "VIBERBM", "KAKAOTALK" and "MMS" are available, within the limitations set by each channel (see below).

Each request sends to a single recipient, but can list several channels and the address to use in each channel. The channels will be tried in the given order and at most one successful message will be sent. For SMS and RCS, usually the same address will be given, because both channels use phone number as address. The same goes for Whatsapp, Viber Business Messages, Kakaotalk and MMS, while Facebook Messenger and Viber Bot use different identifiers to indicate the intended recipient.

1.1. POST request example

```
https://secure.lekab.com/rich/api/send
```

With the contents of the HTTP body:

```
{
  "username":"testuser","password":"testpass",
  "to":[{"channel":"RCS","address":"46701234567"}],
  "message": {
    "choice_message":{
      "text_message":{"text":"Are you available for emergency work?"},
      "choices":[ {"text_message":{"text":"Yes"}}, {"text_message":{"text":"No"}} ]
    }
  },
  "metadata":"emergency 07734"
}
```

1.1.1. Explanation of parameters for /send

POST json key	json value	
username	string	username of the LEKAB message sending account, see authentication methods above
password	string	password of the LEKAB message sending account, see authentication methods above
apikey	string	API key of the LEKAB message sending account, see authentication methods above
appname	string	Name of an app connected to the LEKAB message sending account. If LEKAB has not specified an appname to use, omit this to use the default appname "" (empty string)

POST json key	json value	
to	json list of objects with channel and address fields	addresses in different channels for a single recipient in order of trying
channel	string	"RCS", "SMS", "WHATSAPP", "MESSENGER", "VIBER", "VIBERBM", "KAKAOTALK" or "MMS"
address	string	Recipient id, e.g. for RCS and SMS channels: international phone number (country+area+subscriber) no intl prefix like 00 etc
message	json object, see below	message to send, see below
channel_properties	json object/dictionary	{"key1":"value1", "key2":"value2", ...} available keys listed for the channels below
statusurl	string	url (starting with http:// or https://) replacing pre-set default url for HTTP POST callback of status of this message
incomingurl	string	url (starting with http:// or https://) replacing pre-set default url for HTTP POST callback of reply to this message
metadata	string	identifying data not sent to phone but echoed back with status reports and replies

1.1.2. HTTP response to /send

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.

```
{
  "result" : "OK",
  "id" : "644535162964586496"
}
```

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".

1.1.3. Explanation of response to /send

POST json key	json value (strings quoted)	
result	string	OK if successful
id	string	id of the message used for future references to this message, e.g. status callbacks
error	string	error description if not successful

1.1.4. 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["to"] = [{"channel":"RCS","address":"46701234567"}]
sendreq["message"] = {
   "choice_message": {
       "text_message":{
         "text":"Can you work extra hours on Christmas Day?"
       },
       "choices":[
         {"text_message":{"text":"Yes"}},
         {"text_message":{"text":"No"}}
       ]
   }
}
sendreq["statusurl"] = 'https://my.company.com/comms/lekab/status'
sendreq["incomingurl"] = 'https://my.company.com/comms/lekab/incoming'
sendreq["metadata"] = 'Xmas 2021 employee 1234'
sendreq_json = json.dumps(sendreq)

url = 'https://secure.lekab.com/rich/api/send'
response = requests.post(url, data=sendreq_json, headers=headers)
sendresp = response.json()
if sendresp["result"] == 'OK':
    print("Successful message to " + sendreq["to"][0]["address"] +
```

```
            " channel " + sendreq["to"][0]["channel"] +
            " got id " + sendresp["id"])
else:
    print("Send error: " + sendresp["error"])
```

will output

```
Successful message to 46701234567 channel RCS got id 644535162964586496
```

1.2. Channels and their specific features

In the following listing, special parameters (`channel_properties`) available for each channel are listed below. They may be added as a json object / string-to-string dictionary in the send request and will apply to the respective channel.

Example:

```
{
  "username":"testuser","password":"testpass",
  "to":[{"channel":"SMS","address":"46701234567"}],
  "message": {
      "text_message":{"text":"This is an example message where no reply is expected"}
  },
  "channel_properties":{"SMS_SENDER":"MYCOMPANY"}
}
```

The SMS in the example will have the sender id MYCOMPANY

1.2.1. RCS

The RCS channel with implementation of six message types from the Universal Profile is the cornerstone of the Rich Web service, and is the base from which exceptions and special features deviate, as described below. The phone number is used as recipient id.

Your RCS bot must be registered so that the recipient can trust that your company is truly the sender of messages. This will include submitting a graphic profile with e.g. your company logo.

1.2.2. SMS

SMS is obviously inherently a text-only channel, so any rich content must be transcoded. For instance URLs for pictures, sound and video clips will be shown instead of the media itself. As usual, the phone number is used as recipient id.

Currently, two `channel_properties` are defined for SMS messages:

`SMS_FLASH_MESSAGE` Whether this is flash SMS message. Flash SMS messages are shown on screen without user interaction while not saving the message to the inbox. Possible values are "true" and

"false". The default is "false".

`SMS_SENDER` The sender to use when sending a message on SMS channel. Can be a valid MSISDN, short code or alphanumeric sender. If the sender is rejected by the underlying SMS channel you get the error information in the delivery report. If an answer is expected to the SMS, this property should not be set, and the system should assign a number to which the answer is expected as the sender id, so the user can easily reply.

1.2.3. Whatsapp

You need to have a registered Whatsapp Bot which is the sender of the messages.

Whatsapp uses the phone number as recipient id.

There is a special opt-in/opt-out procedure that must be followed for unsolicited messages to be sent, and the /opt subendpoints (see below) are to be used when the intended recipient requires opt-in or opt-out to messages. The recipient may notify you in any way convenient, and you must then call the appropriate endpoint.

There are also strict rules about sending unsolicited messages. These have to be derived from pre-registered templates, as described under message type `template_message`, sub-type `kakaotalk_template`, in the section on message specification below. Whatsapp decide whether to approve or reject a proposed template, depending on how certain they are that the template will be used to send a message that the recipient explicitly wants from your business ("you can pick up your order number X at location Y"), and not a typical unsolicited business message ("Now 10% sale in all our stores", "You are pre-approved for a credit card"). There are quite a few configuration options at send time for Whatsapp templates, which can make the API call somewhat complicated (see below).

1.2.4. Kakaotalk

You need to have a registered Kakaotalk channel as sender of the messages.

Kakaotalk uses the phone number as recipient id.

Any kind of messages (within reason) can be sent during Korea daytime hours, between 8:00 AM and 9:00 PM Korea Standard Time (KST). Messages sent outside that time are rejected. Before sending those messages the receiver must subscribe to your channel. Kakaotalk does not support carousel messages and location messages, so these are transcoded to text, but the other message types described here are fully supported, with the following limitation:

Media content is limited to .png and .jpg image media types. Media included in the Media message will be rejected by the channel if:

- the width is below 500px, or if the 'width:length' ratio is below 2:1, or above 3:4 or if the wide image dimension is not equal to [800px x 600px]
- image file format is different than JPG or PNG
- file size is bigger than 500KB or for image with dimension [800px x 600px] is bigger than 2MB

To send messages outside these hours, messages have to be derived from pre-registered templates, as described under message type `template_message`, sub-type `kakaotalk_template`, in the section on message specification below. Kakaotalk decide whether to approve or reject a proposed template, depending on how certain they are that the template will be used to send a message that the recipient explicitly wants from your business ("you can pick up your order number X at location Y"), and not a typical unsolicited business message ("Now 10% sale in all our stores", "You are pre-approved for a credit card").

A template derived message that is used to send out user credentials, like passwords, can be sent with priority (i.e. faster) by setting the `channel_properties` value `"KAKAOTALK_AUTHENTICATION": "true"`. Such a message must contain one of the strings: "auth", "password", "verif", "□□□□□", "□□", "□□□□" or "□□".

Example (where the template referenced contains one of the strings):

```
{
  "username":"testuser","password":"testpass",
  "to":[{"channel":"KAKAOTALK","address":"46701234567"}],
  "message": {
      "template_message":{
          "kakaotalk_template":{
              "template_id": "password_sendout",
              "parameters": {
                  "password":"abc123",
                  "validday":"January 30"
              }
          }
      }
  },
  "channel_properties":{"KAKAOTALK_AUTHENTICATION":"true"}
}
```

The template derived message in the example will be sent faster than other messages.

1.2.5. Facebook Messenger

With Facebook Messenger, a Facebook page can send messages to contacts. The contact has to initiate the conversation by sending a message to the page. Ordinarily, this allows you to respond within 24 hours, but the time limit can be overridden by giving certain `channel_properties`. Registration of your sender page with Facebook is necessary, and will include your privacy policy and business purpose.

A special id, which is not the phone number, is used as recipient id.

`Facebook Messenger rules for the EU countries:` We have no information whether these rules will change in the future. Facebook does not allow sending sound or video clips within the EU countries. Only pictures are available. Also note that delivery receipts and read receipts are not sent within the EU countries, so there will be no indication whether the recipient received or opened the message. Choice buttons are only available in iOS or Google apps, not on web browser clients, and

the message types choice_message, card_message, carousel_message and location_message are only rendered on such apps, with an error message given if the recipient is not an app.

Currently, three `channel_properties` are defined for Facebook Messenger messages:

`MESSENGER_MESSAGING_TYPE` Messenger messaging type. For more information visit: https://developers.facebook.com/docs/messenger-platform/send-messages/#messaging_types Defaults to MESSAGE_TAG if MESSENGER_MESSAGE_TAG is set.

`MESSENGER_MESSAGE_TAG` Messenger message tag. Enables sending specific updates to users outside the standard messaging window. If you want to send messages after the 24h response window it is required to include the proper message tag. For more information including a list of possible values and applicable policies visit: https://developers.facebook.com/docs/messenger-platform/send-messages/message-tags There is no default value for this property.

`MESSENGER_NOTIFICATION_TYPE` Messenger push notification type. Possible values are REGULAR (sound/vibration), SILENT_PUSH (on-screen notification only) and NO_PUSH (no notification). The default is REGULAR.

1.2.6. Viber Bot

You will need a Viber Bot registered with Viber.

Viber Bot communication has to be initiated by the phone user sending in a message, thereby subscribing to your bot. Unsolicited messages are not allowed, and even not possible since the recipient id is not known before the incoming message is received by your bot.

Viber Bot supports all message types, incoming messages and choice replies.

Currently, two `channel_properties` are defined for Viber Bot messages:

`VIBER_SENDER_NAME` Viber Bot sender’s name to display. Max 28 characters.

`VIBER_SENDER_AVATAR` Viber Bot sender’s avatar URL. Avatar size should be no more than 100 kb. Recommended 720x720.

1.2.7. Viber Business Messages

Viber Business messages use the phone number as recipient id. Of the three types of Viber Business accounts, we support "1way" and "2way", but not "session". You need a contract for such a business account in order to send messages. If answers are required to your messages, a "2way" account is needed.

Viber Business does only support a single choice in each choice list, and choices have to be url, call or location choices. Buttons for sending back suggested text are not supported by Viber Business, but postback data will be delivered when a choice is made. If a text answer is needed, the phone user must enter the text and send to you.

Media messages can only render images, while sound clips and video clips are rendered as urls.

Viber Business does not support carousel messages.

1.2.8. MMS

The classic MMS channel needs only a MMS account with us. MMS, as always, uses the phone number as recipient id.

MMS supports text_message, media_message and card_message, while choice_message, carousel_message and location_message are transcoded into text. Incoming messages and choice responses are accepted, but will likely arrive at the SMS channel if they do not contain any media, because that is what the phone will use to send a text answer.

Currently, one `channel_properties` key is defined for MMS messages:

`MMS_SENDER` Required on MMS channel, specifies the shortcode or longnumber to use as the sender.

1.3. Message specification

The `message` field in the send request will contain a JSON object (JSON document) specifying the message to send. A valid message object will contain exactly one field, where the field name is the type of message and the field value is an inner JSON object specifying a message of that type.

The seven available message types are `text_message`, `media_message`, `choice_message`, `card_message`, `carousel_message`, `location_message` and `template_message`.

Types `choice_message`, `card_message` and `carousel_message` have a `choices` parameter (a JSON list of choice objects), which in rich content channels, like "RCS", creates clickable buttons (up to 3 buttons per choices parameter) which each initiates a user action. The action can be to return a text response (e.g. "Yes"), the initiation of a phone call from the phone, the access to a url from the phone's browser or the display of a location (latitude and longitude) on the phone's map application. Rich content channels callback a choice response when the user selects a choice, also for the call and url choices. The "SMS" channel will mark user responses to a choice-containing message as a choice response, but will callback text written by the user only.

1.3.1. Message type `text_message`

A text message is similar to an SMS in that the entire message consists of a piece of text.

Example:

```
{
  "text_message":{
    "text":"Hello and welcome! Your password is dog123"
  }
}
```

Fields:

json key	**json value**	
text	string	Message text

1.3.2. Message type `media_message`

A media message consists of a url pointing to a picture, sound clip or video clip to be sent in the channel. Obviously, there are various limitations regarding allowed media formats and sizes, as well as recommendations for best results, all described in a separate guideline document. The url must be a public address where the API can access the media data. The API user is responsible for uploading the media data so that it is available on the web, for instance to a cloud service object storage bucket. For the SMS channel, the url given is simply echoed to the message as a link, and the media must be available when the recipient asks for it.

Example:

```
{
  "media_message":{
    "url":"https://upload.wikimedia.org/wikipedia/commons/a/aa/Medieval_harp.jpg"
  }
}
```

Fields:

json key	**json value**	
url	string	Full url where the media data can be accessed, must begin with `http://` or `https://`

1.3.3. Message type `choice_message`

A choice message consists of a text message together with a list of choices (1-3 choices allowed). The available choices are described in a later section in this document.

```
{
  "choice_message":{
    "text_message":{"text":"Are you available for emergency work?"},
    "choices":[
      {"text_message":{"text":"Yes"}},
      {"text_message":{"text":"No"}}
    ]
  }
}
```

Fields and subfields:

json key	**json value**	
text_message	json object	A text message similar to the message type text_message
text	string	Message text

json key	json value	
choices	JSON list of choices	1 to 3 choices, see available choices below

1.3.4. Message type `card_message`

A card message is a media message with a set of 0-3 choices. The card has a title and may have a descriptive text. For picture and video formats suitable for displaying in a card, see the separate document with media limits and style guides. The available choices are described in a later section in this document.

```
{
  "card_message":{
    "title":"Rent a Bard",
    "description":"Spice up your party with a traditional singer of poetry",
    "media_message":{
      "url":"https://upload.wikimedia.org/wikipedia/commons/a/aa/Medieval_harp.jpg"
    },
    "choices":[
      {"text_message":{"text":"Yes"}},
      {"text_message":{"text":"No"}}
    ]
  }
}
```

Fields and subfields:

json key	json value	
title	string	Title of the message card
description	Optional string	Descriptive text of the card
media_message	json object	Optional media message similar to the message type media_message
url	string	full url where the media data can be accessed, must begin with `http://` or `https://`
choices	Optional JSON list of choices	0 to 3 choices, see available choices below

1.3.5. Message type `carousel_message`

A carousel message consists of 1-10 card messages, presented in a device-specific way so that the user can scroll among the cards. Each card may have 0 to 3 choices, as described in the available choices section below. The available choices are described in a later section in this document. Also, the entire carousel can have a set of 0 to 3 top level choices, which should pertain to the entire carousel, rather than any specific card. The RCS specification does not include any top level title or top level description, so each card has to present itself properly.

Note that on some mobile handsets, even when a card in a carousel contains an image which conforms to the RCS style guidelines, still only the two first choices on each card are displayed even if three are specified, and only when the media message is omitted, all three choices are shown. We have no more information about which handsets this applies to and when this will be fixed.

```
{
  "carousel_message":{
    "cards":[
      {
        "title":"In picturesque Arkham",
        "description":"Cthulhu Hotel service on call",
        "media_message":{
          "url":"https://arkhamhotel.com/media/cthulhuhotel.png"
        },
        "choices":[
          {"url_message":
            {"title":"Book a room",
             "url":"https://arkhamhotel.com/book/cthulhu"},
           "postback_data":"BOOK CTHULHU"},
          {"call_message":
            {"title":"Call of Cthulhu", "phone_number":"19786661234"},
           "postback_data":"CALL CTHULHU"}
        ]
      },
      {
        "title":"Close to Miskatonic U",
        "description":"Tentacle Inn creature comforts",
        "media_message":{
          "url":"https://arkhamhotel.com/media/tentacleinn.png"
        },
        "choices":[
          {"url_message":
            {"title":"Book a room",
             "url":"https://arkhamhotel.com/booking/tentacle"},
           "postback_data":"BOOK TENTACLE"},
          {"call_message":
            {"title":"Call the hotel", "phone_number":"19786664321"},
           "postback_data":"CALL TENTACLE"}
        ]
      }
    ],
    "choices":[
      {"text_message":{"text":"Andover Area"},
       "postback_data":"MENU ANDOVER"},
      {"text_message":{"text":"Haverhill Area"},
       "postback_data":"MENU HAVERHILL"},
      {"text_message":{"text":"Salem Area"},
       "postback_data":"MENU SALEM"}
    ]
  }
```

```
}
```

Fields and subfields:

json key	json value	
cards	JSON list of card messages	1-10 card messages similar to the message type card_message
title	string	Title of the message card
description	Optional string	Descriptive text of the card
media_message	Optional json object	media message similar to the message type media_message
url	string	full url where the media data can be accessed, must begin with `http://` or `https://`
choices	Optional JSON list of choices for each card	0 to 3 choices, see available choices below, or 0 to 2 see note above
choices	Optional JSON list of top level choices for the carousel	0 to 3 choices, see available choices below

1.3.6. Message type `location_message`

A location message contains a short text title and a location which is supplied in a format that can be directly read by the phone's map program (e.g. Google maps). The location is sometimes displayed as a "map pin" symbol with an accompanying label. When clicking the location, the map program is started and the location displayed.

```
{
  "location_message":{
    "title":"Location of the place",
    "label":"The place",
    "coordinates": {
      "latitude":48.858093,
      "longitude":2.294694
    }
  }
}
```

Fields and subfields:

json key	json value	
title	string	message to explain the location
label	Optional string	label displayed beside the map pin marker
coordinates	JSON object	contains latitude and longitude

json key	json value	
latitude	float	latitude of location between -90.0 and 90.0, negative for southern hemisphere
longitude	float	longitude of location between -180.0 and 180.0, negative for western hemisphere

1.3.7. Message type `template_message`

Some messaging channels allow sending at more hours of the day, or to users who have not been heard from recently, only when the message is derived from a message template that has been registered and pre-approved by the channel.

Currently, the subtypes `whatsapp_template` and `kakaotalk_template` are available, for use in the corresponding channels. A `template_message` may only refer to one template subtype. Fallback between channels using different template subtypes is not supported (you can not in one API call specify that the Kakaotalk channel with one template should be tried first, and if that did not succeed, the Whatsapp channel should be tried with a different template). Fallback between a template using channel and another channel is not documented, and probably does not work.

1.3.8. Message type `template_message` sub-type `whatsapp_template`

The subtype `whatsapp_template` is used to send messages through the Whatsapp channel using pre-approved templates (registered and approved by Whatsapp). This is the only kind of message allowed for sending to a Whatsapp user who has not been heard from for 24 hours.

A whatsapp_template has three parts: a header (optional), a text body and a set of response buttons (optional). When you define your template for Whatsapp to approve, you decide which parts are available, and their types and composition.

The header can be one of a short line of text, a document, an image or a video. If the header is of type text, you can define place-holders {{1}}, {{2}}, etc in the header (must be consecutive starting with 1), and you send in the same number of header parameter objects of type `text` to replace the place-holders in the specific message. If the header is instead a media type, the header parameter object list must contain a single header parameter object of one of the types `document`, `image` or `video`, as pre-defined in the template, where a url `link` to the intended media file is specified.

The body text can have its own set of place-holders {{1}}, {{2}}, etc. independent of any header text parameters, (and also consecutive, starting with 1). You send in the appropriate number of text strings to replace the place-holders.

The buttons defined in the template will have a defined order, and you send in the same number of button parameter objects of the defined types in the same order. There are three types of buttons available, `reply` for quick-reply buttons where you specify the `postback_data`, i.e. the payload sent back as a reply, `call` for the option to call a pre-registered number (which goes to your company and cannot be dynamically changed), and `url` for a button to visit a given pre-registered url (owned by your company). You can register a dynamic url, where you can supply a `url_suffix` that is appended to the pre-registered url, e.g. to identify which user answered. Currently, Whatsapp will allow up to three `reply` buttons or up to two buttons of type `call` or `url`, but not both `reply` buttons

and other buttons at the same time. It is unclear whether two `call` or `url` buttons are allowed, or only a max of one of each.

```
{
  "template_message": {
    "whatsapp_template": {
      "template_id": "pick_up_your_order",
      "language_code": "en_US",
      "header_parameters": [
        {
          "type": "image",
          "link": "https://www.mycompany.com/media/images/company_logo_small.jpg"
        }
      ],
      "body_parameters": ["John Smith", "12314258", "Bigtown City Center", "Jan 31"],
      "button_parameters": [
        {
          "type": "call"
        },
        {
          "type": "url",
          "url_suffix": "?order=12314258"
        }
      ]
    }
  }
}
```

Fields and subfields:

json key	**json value**	
whatsapp_template	JSON object	Single allowed top level field, contains all message parameters
template_id	string	The registered name of the template
language_code	string	One of the registered languages for the template name
header_parameters	list of JSON objects	The header parameters (see below), corresponding to template
body_parameters	list of strings	The body parameters, corresponding to template
button_parameters	list of JSON objects	The button parameters (see below), corresponding to template

Fields of the header parameter objects:

json key	json value	
type	string	"text", "document", "image" or "video"
text	string	Required for type "text", otherwise not allowed
link	string	Required for types "document", "video", "image": header media url, not allowed for "text"
provider_name	string	Optional allowed for types "document", "video", "image" on distributed media provider
filename	string	Optional only allowed for type "document"

Fields of the button parameter objects:

json key	json value	
type	string	"reply", "call" or "url"
postback_data	string	Required for type "reply", otherwise not allowed
url_suffix	string	Required for type "url" if registered template has dynamic url, otherwise not allowed

1.3.9. Message type `template_message` sub-type `kakaotalk_template`

The sub-type `kakaotalk_template` is used to send messages through the Kakaotalk channel using pre-approved templates (registered and approved by Kakaotalk). This is the only kind of message allowed for sending to a Kakaotalk user outside normal daytime business hours.

In contrast to the very flexible and configurable Whatsapp templates, where all the configurability forces a rather complicated call structure, the Kakaotalk template needs all configuration to be done in advance when registering the template, and the API call simply substitutes named placeholders in the template with the data pertaining to the specific recipient, making for a very straight-forward API call:

```
{
  "template_message": {
    "kakaotalk_template": {
      "template_id": "pick_up_your_order",
      "parameters": {
          "name":"John Smith",
          "packageid":"12314258",
          "address":"Bigtown City Center",
          "date":"Jan 31"
      }
    }
  }
}
```

Fields and subfields:

json key	json value	
whatsapp_template	JSON object	Single allowed top level field, contains all message parameters
template_id	string	The registered name of the template
parameters	JSON object: map of string to string	Values corresponding to the placeholder names in template

As detailed in the channel features section on Kakaotalk (above), there is a `channel_properties` value `KAKAOTALK_AUTHENTICATION` which can be set to `"true"` to allow a message with authentication data (temporary passwords etc.) to be fast-tracked for sending to the recipient.

1.3.10. Available choices for the `choices` list parameter within certain message types

The message types `choice_message`, `card_message` and `carousel_message` allow a list of up to three choices at one or several places within the message. There are four types of choices available, called `text_message`, `url_message`, `call_message`, and `location_message`. When the user clicks on a choice, the sender is notified via callback as an incoming message.

Choice `text_message`

When the `text_message` choice is selected, a text message is sent back to the sender. This is the closest analogy to responding to SMS messages. Note that while in rich channels clicking the choice will automatically send the correct response, if fallback to SMS occurs the phone user becomes responsible for sending the correct response text back to the sender. If `postback_data` is specified for a `text_message` choice, that is what is actually sent back, rather than the text displayed on the choice button (except in the fallback case).

```
"choices":[
  {"text_message":{"text":"Yes, I am available"},
   "postback_data":"YES"},
  {"text_message":{"text":"Sorry, not available"},
   "postback_data":"NO"}
]
```

json key	json value	
text_message	JSON object	a text message definition
text	string	the text of the text message, as displayed on the phone
postback_data	Optional string	alternative message actually sent back to sender, default is the text parameter value

Choice `url_message`

When a `url_message` choice is selected, the phone's default browser opens the url, and at the same time the `postback_data` is sent back to the sender to inform of the selection.

```
"choices":[
  {"url_message":
    {"title":"Book Grand Hotel",
     "url":"https://sthlmhotel.com/booking/grand"},
   "postback_data":"BOOK GRAND"},
  {"url_message":
    {"title":"Visit our home page",
     "url":"https://sthlmhotel.com/home"},
   "postback_data":"HOME"}
]
```

json key	**json value**	
url_message	JSON object	a url message definition
title	string	the text on the choice button displayed on the phone
url	string	full url to open in the browser, must begin with `http://` or `https://`
postback_data	Optional string	message actually sent back to sender, default is the title parameter value

Choice `call_message`

When a `call_message` choice is selected, the phone opens its functionality for placing telephone calls with the given phone number active (or device-dependently already dials the phone call), and at the same time the `postback_data` is sent back to the sender to inform of the selection. The default postback_data is phone_number_title e.g. "4631400100_Call our booking" if not specified.

```
"choices":[
  {"call_message":
    {"title":"Call our service desk",
     "phone_number":"46701234567"},
   "postback_data":"CALL DESK"},
  {"call_message":
    {"title":"Call a friend",
     "phone_number":"46702345678"},
   "postback_data":"CALL FRIEND"}
]
```

json key	**json value**	
call_message	JSON object	a call message definition
title	string	the text on the choice button displayed on the phone
phone_number	string	phone number to call when this choice is selected

json key	json value	
postback_data	Optional string	alternative message actually sent back to sender, default is phone_number_title e.g. "4631400100_Call our booking"

Choice `location_message`

When a `location_message` choice is selected, the phone's default map program is started and the location displayed, and at the same time the `postback_data` is sent back to the sender to inform of the selection. The default `postback_data` is latitude_longitude_title, e.g. "38.897957_-77.036560_White House" if not specified.

```
"choices":[
  { "location_message":{
      "title":"Show on a map",
      "label":"La Tour Eiffel",
      "coordinates": {
        "latitude":48.858093,
        "longitude":2.294694
      }
    },
    "postback_data":"MAP EIFFEL"
  }
]
```

json key	json value	
location_message	JSON object	a location message definition
title	string	message to explain the location
label	Optional string	label displayed beside the map pin marker
coordinates	JSON object	contains latitude and longitude
latitude	float	latitude of location between -90.0 and 90.0, negative for southern hemisphere
longitude	float	longitude of location between -180.0 and 180.0, negative for western hemisphere
postback_data	string	Optional alternative message actually sent back to sender, default is latitude_longitude_title, e.g. "38.897957_-77.036560_White House"

Chapter 2. The `/callbackurl` endpoints

Used for setting and unsetting the two default urls (starting with `http://` or `https://`) pointing to a service listening for `HTTP POST` requests where delivery status and incoming messages should be sent, if no urls are given in the `send` request.

The listening service at a url is expected to respond with a `200 OK` result code (or the equivalent `201 Created`, `202 Accepted` or `204 No Content`). If no service is found, or the service does not respond with a result code, or a non-accepted result code is received, the callback will be retried a number of times, and then abandoned. Any HTTP body content returned in the service response will be ignored.

There may be up to a 5-minute delay until the new default url value is propagated to the different servers, so if different urls for different messages are desired, the corresponding parameters to the send request should be utilized.

Available sub-endpoints are `/query`, `/setstatus`, `/unsetstatus`, `/setincoming`, and `/unsetincoming`.

Incoming messages which are not choice responses to an earlier sent message, i.e. incoming messages where the phone user starts the conversation, can only be received by callback to a url defined by the `/setincoming` sub-endpoint, since there is no earlier send request from which to derive a callback url in that case.

All requests are `HTTP POST` with parameters in a JSON document in the `HTTP` body. For `query` and `unset`, the only parameters needed are those that identify the user account, while for `set` the relevant url to set must be supplied. Parameters that are not supplied can be omitted in the request JSON document or set to the empty string. The only non-empty parameters allowed are those that are needed for the requested action. If, for instance, `/setstatus` is called with a non-empty value for the `incomingurl` parameter, the request fails and no changes are made to the default urls. The only way to unset a url is to call the corresponding `unset` endpoint. Using `set` with a null or empty string parameter is not an allowed way of unsetting the url and will fail giving an `ERROR` result.

Note that the urls for status reports and responses to a message sent by this API will be stored at the time of sending. So responses and status messages to older messages will arrive at the urls in use at their send time - either special urls for that specific message specified in the send request, or the default urls valid at the time of sending. Later changes to the default urls will not affect messages that have already been sent.

2.1. Explanation of parameters for the /callbackurl/xxx endpoints

POST json key	json value	
username	string	username of the LEKAB message sending account, see authentication methods above
password	string	password of the LEKAB message sending account, see authentication methods above

POST json key	json value	
apikey	string	API key of the LEKAB message sending account, see authentication methods above
appname	string	Name of an app connected to the LEKAB message sending account. If LEKAB has not specified an appname to use, omit this to use the default appname "" (empty string)
statusurl	Optional/Required string	empty or url (starting with http:// or https://) default url for HTTP POST callback of status of sent messages
incomingurl	Optional/Required string	empty or url (starting with http:// or https://) default url for HTTP POST callback of incoming messages

The response for all endpoints gives the new setting for both default urls. Where no url is set, this fact is represented as an empty string value of the url.

2.2. The /callbackurl/query endpoint

This endpoint is used for querying the currently set callback urls. While this url will immediately return the latest updated value, callback to the correct urls will only be guaranteed on all servers when 5 minutes have passed after using any of the setting or unsetting endpoints.

```
https://secure.lekab.com/rich/api/callbackurl/query
```

With the contents of the HTTP body not allowing any non-empty values of the `statusurl` or `incomingurl` parameters:

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

Or equivalently with allowed empty url parameters:

```
{
  "username":"testuser",
  "password":"testpass",
  "statusurl":"",
  "incomingurl":""
}
```

This will return `200 OK` and a message similar to this one:

```
{
  "statusurl" : "https://my.company.com/rcscallbacks/status",
```

```
  "incomingurl" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

2.3. The /callbackurl/setstatus endpoint

This endpoint is used for setting the callback url for callback of message status updates. Note that callback to the correct url will be guaranteed on all servers when 5 minutes have passed after using any of the setting or unsetting endpoints, and the url will be used for status of new messages sent, not those already sent. If a message is sent with the `statusurl` parameter to the `/send` endpoint defined, that setting will override this one.

```
https://secure.lekab.com/rich/api/callbackurl/setstatus
```

With the contents of the HTTP body requiring a valid statusurl parameter and not allowing any non-empty incomingurl parameter:

```
{"username":"testuser","password":"testpass",
 "statusurl":"https://my.company.com/rcscallbacks/betterstatus"}
```

Will return `200 OK` and a message similar to this one:

```
{
  "statusurl" : "https://my.company.com/rcscallbacks/betterstatus",
  "incomingurl" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

2.4. The /callbackurl/unsetstatus endpoint

This endpoint is used for unsetting the callback url for callback of message status updates. Note that stopping the callback will be guaranteed on all servers when 5 minutes have passed after using the unsetting endpoint, and the stopping only applies to new messages sent, not those already sent. If a message is sent with the `statusurl` parameter to the `/send` endpoint defined, that setting will override this one.

```
https://secure.lekab.com/rich/api/callbackurl/unsetstatus
```

With the contents of the HTTP body:

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

Will return `200 OK` and a message similar to this one, where the statusurl will always be the empty string:

```
{
  "statusurl" : "",
  "incomingurl" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

2.5. The /callbackurl/setincoming endpoint

This endpoint is used for setting the default callback url for callback of incoming messages, including responses to messages sent, and unsolicited incoming messages. Note that callback to the correct url will be guaranteed on all servers when 5 minutes have passed after using the setting endpoint, and the new url will be used for responses to new messages sent, not those already sent. If a message is sent with the `incomingurl` parameter to the `/send` endpoint defined, that setting will override this one.

```
https://secure.lekab.com/rich/api/callbackurl/setincoming
```

With the contents of the HTTP body requiring a valid incomingurl parameter and not allowing any non-empty statusurl parameter:

```
{"username":"testuser",
"password":"testpass",
"statusurl":"",
"incomingurl":"https://my.company.com/rcscallbacks/newincoming"}
```

Will return `200 OK` and a message similar to this one:

```
{
  "statusurl" : "https://my.company.com/rcscallbacks/status",
  "incomingurl" : "https://my.company.com/rcscallbacks/newincoming"
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

2.6. The /callbackurl/unsetincoming endpoint

This endpoint is used for unsetting the callback url for callback of responses and other incoming messages. Note that stopping the callback will be guaranteed on all servers when 5 minutes have passed after using the unsetting endpoint, and the stopping of responses only applies to new messages sent, not those already sent. If a message is sent with the `incomingurl` parameter to the `/send` endpoint defined, that setting will override this one.

```
https://secure.lekab.com/rich/api/callbackurl/unsetincoming
```

With the contents of the HTTP body:

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

Will return `200 OK` and a message similar to this one, where the incomingurl will always be the empty string:

```
{
  "statusurl" : "https://my.company.com/rcscallbacks/status",
  "incomingurl" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

Chapter 3. The `/opt` endpoints: `/opt/optin` and `/opt/optout`

Used for registering (`/optin`) and removing (`/optout`) the registration of a recipient's permission for you to send messages. Currently, only the WHATSAPP channel requires the use of this endpoint, and attempts to use the endpoint with another channel will result in an error response.

All requests are `HTTP POST` with parameters in a JSON document in the `HTTP` body. The parameters needed are those that identify the user account, and the channel ("WHATSAPP") and the address (phone number) of the recipient.

3.1. Explanation of parameters for the /opt/xxx endpoints

POST json key	json value	
username	string	username of the LEKAB message sending account, see authentication methods above
password	string	password of the LEKAB message sending account, see authentication methods above
apikey	string	API key of the LEKAB message sending account, see authentication methods above
appname	string	Name of an app connected to the LEKAB message sending account. If LEKAB has not specified an appname to use, omit this to use the default appname "" (empty string)
channel	String	The channel to opt in or out for (currently only "WHATSAPP" is allowed)
address	String	The phone number (for Whatsapp) of the recipient
remark	string	Place to save info about how the enduser requested the opt-in/opt-out
statusurl	string	url (starting with http:// or https://) replacing pre-set default url for HTTP POST callback of status of this opt-in/opt-out
metadata	string	identifying data not sent to operator but echoed back with status reports and replies

The response repeats your request, unless there is an obvious error in your input.

3.2. The /opt/optin endpoint

This endpoint is used for opting in to (allowing) messaging from your app to the recipient. The call will return immediately, with a summary of the supplied data, while the actual result of the opt-in operation will be returned asynchronously via callback on the status url (see below).

```
https://secure.lekab.com/rich/api/opt/optin
```

With the contents of the HTTP body:

```
{
  "username":"testuser",
  "password":"testpass",
  "channel" : "WHATSAPP",
  "address" : "46701234567",
  "remark" : "Web sign-up"
}
```

This will quickly return `200 OK` and a message similar to this one:

```
{
  "requestid" : "692020948953509888",
  "action" : "optin",
  "channel" : "WHATSAPP",
  "address" : "46701234567",
  "remark" : "Web sign-up",
  "statusurl" : "",
  "metadata" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

3.3. The /opt/optout endpoint

This endpoint is used for securing that you do not send messages to a recipient who does not want messages. The call will return immediately, with a summary of the supplied data, while the actual result of the opt-out operation will be returned asynchronously via callback on the status url (see below).

```
https://secure.lekab.com/rich/api/opt/optout
```

With the contents of the HTTP body:

```
{
  "username":"testuser",
  "password":"testpass",
  "channel" : "WHATSAPP",
  "address" : "46701234567",
  "remark" : "E-mail from john.smith@email.com"
}
```

This will quickly return `200 OK` and a message similar to this one:

```
{
  "requestid" : "692022519078588416",
  "action" : "optout",
  "channel" : "WHATSAPP",
  "address" : "46701234567",
  "remark" : "E-mail from john.smith@email.com",
  "statusurl" : "",
  "metadata" : ""
}
```

If there was an error in the request, there will instead be a non-`200` non-`OK` status code and a response body similar to this:

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

Chapter 4. Callback to url of message status

When a message has been sent via the `/send` endpoint, the delivery status of the message can be followed by `HTTP POST` callbacks to the `statusurl` that was set in the message, or, if no `statusurl` was supplied, to the pre-set `statusurl` established by calls to the `/callbackurl/setstatus` endpoint. If also no pre-set url is defined, no message delivery status reports will be sent back to the sender, but the message will anyway be delivered to the recipient (or not, if there is a problem).

The listening service at the `statusurl` is expected to respond with a `200 OK` result code (or the equivalent `201 Created`, `202 Accepted` or `204 No Content`). If no server is found, the server does not answer with a result code, or a non-accepted result code is received, the callback will be retried a number of times, and then abandoned.

4.1. Examples of callback of message delivery status

This is a message informing of a `DELIVERED` status for the previously sent message:

```
{
  "id": "652141860785008640",
  "event": "STATUS",
  "send_time": "2021-03-05T11:38:56.000Z",
  "to_channel": "RCS",
  "to_address": "46701234567",
  "status": "DELIVERED",
  "status_reason": "",
  "status_time": "2021-03-05T11:39:05.149Z",
  "operator_id": "01F012XD8HX96D1AXMS3J41WZZ",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": "This is the metadata"
}
```

This is a message informing of a `READ` status for the previously sent message:

```
{
  "id": "652141860785008640",
  "event": "STATUS",
  "send_time": "2021-03-05T11:38:56.000Z",
  "to_channel": "RCS",
  "to_address": "46701234567",
  "status": "READ",
  "status_reason": "",
  "status_time": "2021-03-05T11:39:05.329Z",
  "operator_id": "01F012XD8HX96D1AXMS3J41WZZ",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": "This is the metadata"
```

```
}
```

This is a message informing of a `FAILED` status for a previously sent message. For `FAILED` and `SWITCHED` status, there is usually a non-empty `status_reason`, where the operator gives a probable reason for the status.

```
{
  "id": "653217127892783104",
  "event": "STATUS",
  "send_time": "2021-03-08T10:51:39.000Z",
  "to_channel": "RCS",
  "to_address": "46702345678",
  "status": "FAILED",
  "status_reason": "RECIPIENT_NOT_REACHABLE : The underlying channel reported: Unable
to find rcs support for the given recipient",
  "status_time": "2021-03-08T10:51:44.244Z",
  "operator_id": "01F08QD2DD5HB21R9AMMYR1CY5",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30U2B",
  "metadata": "This is the metadata"
}
```

4.2. Explanation of JSON fields in the callback of message delivery status

POST json key	json value	
id	string containing long integer	LEKAB message id same as in the response to a `/send` request
event	string	Type of the callback: "STATUS"
send_time	string containing ISO 8601 UTC timepoint	When message was sent. The Z signifies Zulu (UTC = GMT+0) time
to_channel	string	The channel for which send status is reported, e.g. "RCS" or "SMS"
to_address	string	The address in the channel where the message was sent for which status is reported
status	string	One of the available statuses (see below)
status_reason	string	Explanation only non-empty for FAILED or SWITCHED status (see below)

POST json key	json value	
status_time	string containing ISO 8601 UTC timepoint	When this status report was received. The Z signifies Zulu (UTC = GMT+0) time
operator_id	string	id assigned to this message by the operator
operator_conversation	string	id assigned to the conversation containing this message by the operator
operator_contact	string	id assigned to the contact containing the recipient of this message by the operator
metadata	string	text supplied when sending this message returned to identify it

4.2.1. Non-final statuses

There has been a switch of channels due to a problem with the latest channel tried. The status_reason field may contain an explanatory text.

`SWITCHED`

4.2.2. Successful final statuses

The recipient's phone has supposedly acknowledged receipt of this message, and in the `READ` case that the user has opened the message for reading (only in some channels, not available for SMS).

`DELIVERED`, `READ`

4.2.3. Failing final statuses

The recipient will not get this message. The status_reason field may contain an explanatory text.

`FAILED`

4.2.4. Unclear

Probably failing final status

`UNKNOWN`

Chapter 5. Callback to url of responses and other incoming messages

When a message has been sent via the `/send` endpoint, choice responses prompted in the message can be received by `HTTP POST` callbacks to the `incomingurl` that was set in the message, or, if no `incomingurl` was supplied, to the pre-set `incomingurl` established by calls to the `/callbackurl/setincoming` endpoint. If also no pre-set url is defined, no choice responses will be sent back to the sender.

Also, if the phone user initiates a conversation and sends in text or media, these messages will be received at the pre-set `incomingurl`.

The listening service at the `incomingurl` is expected to respond with a `200 OK` result code (or the equivalent `201 Created`, `202 Accepted` or `204 No Content`). If no server is found, the server does not answer with a result code, or a non-accepted result code is received, the callback will be retried a number of times, and then abandoned.

5.1. Examples of callback of incoming messages

This is a response to an earlier message. The `postback_data` defined when sending the message was the text "MENU HAVERHILL". Had no `postback_data` been defined, the default postback data for the type of choice in question (see above) would be sent.

```
{
  "id": "652141989248110592",
  "event": "INCOMING",
  "incoming_time": "2021-03-05T11:39:26.444Z",
  "response_to": "652141860785008640",
  "from_channel": "RCS",
  "from_address": "46701234567",
  "incoming_type": "RESPONSE",
  "incoming_data": "MENU HAVERHILL",
  "incoming_message": "{\"choice_response_message\":{\"message_id\":
\"01F012XD8HX96D1AXMS3J41WZZ\",\"postback_data\":\"MENU HAVERHILL\"}}",
  "operator_id": "01F012YB8W4C500KKFHMPT0W4E",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": "This is the metadata"
}
```

This is a response to the same earlier message. The `postback_data` defined when sending the message was the text "BOOK CTHULHU". Had no `postback_data` been defined, the default postback data for the type of choice in question (see above) would be sent. Note that several responses to the same message can be received, if the recipient selects several choices after one another.

```
{
```

```
  "id": "652142236972093440",
  "event": "INCOMING",
  "incoming_time": "2021-03-05T11:40:25.510Z",
  "response_to": "652141860785008640",
  "from_channel": "RCS",
  "from_address": "46701234567",
  "incoming_type": "RESPONSE",
  "incoming_data": "BOOK CTHULHU",
  "incoming_message": "{\"choice_response_message\":{\"message_id\":
\"01F012XD8HX96D1AXMS3J41WZZ\",\"postback_data\":\"BOOK CTHULHU\"}}",
  "operator_id": "01F01304TC3PBR03RB9BSF0DQ7",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": "This is the metadata"
}
```


This is a phone-user initiated message, sent in without being a response to anything. The message is of the type `text_message` (see above). The actual text is found as `incoming_data`.


```
{
  "id": "652140815493107712",
  "event": "INCOMING",
  "incoming_time": "2021-03-05T11:34:46.598Z",
  "response_to": "0",
  "from_channel": "RCS",
  "from_address": "46701234567",
  "incoming_type": "TEXT",
  "incoming_data": "Hej",
  "incoming_message": "{\"text_message\":{\"text\":\"Hej\"}}",
  "operator_id": "01F012NSN97TM81C1B2ARF13EP",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": ""
}
```


This is a phone-user initiated message, sent in without being a response to anything. The message is of the type `media_message` (see above). The url where the picture is found in temporary storage is the `incoming_data`. Note that this message and the former one were received within a second from one another. This is what happens if a phone-user sends in a picture together with a text in an RCS chat window.


```
{
  "id": "652140818517200896",
  "event": "INCOMING",
  "incoming_time": "2021-03-05T11:34:47.322Z",
  "response_to": "0",
  "from_channel": "RCS",
  "from_address": "46701234567",
```

```
  "incoming_type": "MEDIA",
  "incoming_data": "https://rcs-cnt.s3.amazonaws.com/47508dc2-79ff-4976-88e4-
d5b01d4064e9/MzZiYWEyNjk2M2U0ZmVhZDE5ZTQ0NDE0.jpg",
  "incoming_message": "{\"media_message\":{\"url\":\"https://rcs-
cnt.s3.amazonaws.com/47508dc2-79ff-4976-88e4-
d5b01d4064e9/MzZiYWEyNjk2M2U0ZmVhZDE5ZTQ0NDE0.jpg\"}}",
  "operator_id": "01F012NTQYMM3Q1X7ZY94Q1YY6",
  "operator_conversation": "01EWJTKNDNG7T5082S1GWS1CMA",
  "operator_contact": "01EWJTKNBJ4JKS0QQ8H8Q30XEA",
  "metadata": ""
}
```


5.2. Explanation of JSON fields in the callback of incoming message

POST json key	json value	
id	string containing long integer	LEKAB incoming message id
event	string	Type of the callback: "INCOMING"
incoming_time	string containing ISO 8601 UTC timepoint	When message was received. The Z signifies Zulu (UTC = GMT+0) time
response_to	string containing long integer	Incoming message is a response to this LEKAB outgoing message id, or "0" for non-response incoming message
from_channel	string	The channel where the message arrived, e.g. "RCS" or "SMS"
from_address	string	The address in the channel where the message arrived
incoming_type	string	One of the types "RESPONSE", "TEXT" or "MEDIA". "NONE" if indeterminable or other
incoming_data	string	`postback_data` for RESPONSE, text for TEXT, url for MEDIA
incoming_message	string containing JSON object	Full json message (see message format for `/send`, or `choice_response_message` for response, see examples)
operator_id	string	id assigned to this message by the operator
operator_conversation	string	id assigned to the conversation containing this message by the operator
operator_contact	string	id assigned to the contact containing the sender of this message by the operator

POST json key	json value	
metadata	string	text supplied when sending the message to which this is a response, returned to identify it

5.3. Types of messages in the callback of incoming messages

If the callback involves an incoming message that is a choice response to an earlier sent message, i.e. the phone user has clicked on a button displayed with the message, the incoming message type will be `RESPONSE`, and the data will be the `postback_data` of the selected choice (see choices under message format for `send` above).

In the Google Messages app for Android phones, it is relatively easy to send a text message or a picture or video clip to an RCS bot. Such user initiated incoming messages are of type `TEXT` for text messages and of type `MEDIA` for pictures or video clips. It is not easy or maybe even not possible to send the other types of RCS messages (choice message, card message, carousel message and location message) from a mobile handset, and if the phone user would anyway succeed with this, they are reported as type `NONE`, and the full message string needs to be parsed.

As mentioned in the examples, if the phone user sends in a picture with an accompanying text, this is received as two messages of the respective types.

The lifetime of the media at the url given in an incoming message will be finite (on the order of days or weeks) and operator dependent, so if safe-keeping of the media is necessary (e.g. "this is a picture of the collision damage to my car, license plate ABC 123"), it should be downloaded promptly from the given url.

Chapter 6. Callback to url of status of opt-in and opt-out requests

When opt-in or opt-out has been requested via one of the `/opt/optin` or `/opt/optout` endpoints, the success status of the request is returned via `HTTP POST` callbacks to the `statusurl` that was set in the request, or, if no `statusurl` was supplied, to the pre-set `statusurl` established by calls to the `/callbackurl/setstatus` endpoint. If also no pre-set url is defined, no optin/optout status reports will be sent back to the requester, but the request will still be handled properly in the background (or not, if there is a problem).

The listening service at the `statusurl` is expected to respond with a `200 OK` result code (or the equivalent `201 Created`, `202 Accepted` or `204 No Content`). If no server is found, the server does not answer with a result code, or a non-accepted result code is received, the callback will be retried a number of times, and then abandoned.

6.1. Examples of callback of opt-in and opt-out status

This is a message informing of a `OPT_IN_SUCCEEDED` status for the previously requested opt-in:

```
{
  "id": "692020948953509888",
  "event": "OPTIN",
  "request_time": "2021-06-23T12:44:11.000Z",
  "channel": "WHATSAPP",
  "address": "46701234567",
  "status": "OPT_IN_SUCCEEDED",
  "status_reason": "",
  "operator_id": "01F8WEB9EZ8T381YBVT1RH1HGS",
  "operator_contact": "01F7NFR73EJ2M20ZSQP7Y706G1",
  "operator_appid": "01EPQ4H3302SJC16BW5Y2H1NA6",
  "remark": "Web sign-up",
  "metadata": ""
}
```

This is a message informing of a `OPT_OUT_SUCCEEDED` status for the previously requested opt-out:

```
{
  "id": "692022519078588416",
  "event": "OPTOUT",
  "request_time": "2021-06-24T14:50:26.000Z",
  "channel": "WHATSAPP",
  "address": "46701234567",
  "status": "OPT_OUT_SUCCEEDED",
  "status_reason": "",
  "operator_id": "01F8WEPS9FKY221XYM1XS11CEP",
  "operator_contact": "01F7NFR73EJ2M20ZSQP7Y706G1",
```

```
  "operator_appid": "01EPQ4H3302SJC16BW5Y2H1NA6",
  "remark": "E-mail from john.smith@email.com",
  "metadata": ""
}
```

If the `status` is "OPT_IN_FAILED" or "OPT_OUT_FAILED", there is usually a non-empty `status_reason`, where the operator gives a probable reason for the failing action.

6.2. Explanation of JSON fields in the callback of opt-in and opt-out status

POST json key	json value	
id	string containing long integer	LEKAB opt-in/opt-out request id
event	string	Type of the request for which status is reported: "OPTIN" or "OPTOUT"
request_time	string containing ISO 8601 UTC timepoint	When request happened. The Z signifies Zulu (UTC = GMT+0) time
channel	string	The channel for which opt-in or opt-out was requested, e.g. "WHATSAPP"
address	string	The address in the channel for which opt-in or opt-out was requested
status	string	"OPT_XX_SUCCEEDED" or "OPT_XX_FAILED" where XX is "IN" or "OUT"
status_reason	string	Explanation only non-empty for OPT_XX_FAILED status (see below)
operator_id	string	id assigned to this request by the operator
operator_contact	string	id assigned by the operator to the contact containing the address opting in or out
operator_appid	string	id assigned by the operator to the app to which opt-in or opt-out is requested
remark	string	Should contain saved info about how the enduser requested the opt-in/opt-out
metadata	string	identifying data not sent to operator but echoed back with opt-in/opt-out status report