Medallia Adapter

LEKAB Communication Systems AB

Version 5.1.161, 2024-06-10

Medallia Adapter

Introduction

The Medallia software is used for sending out a series of questions to a recipient and interpreting the incoming replies. Example applications include a customer satisfaction questionnaire and an interactive assembly of a customer profile. The system has a number of built-in channels, but can also make use of third-party channel adapters.

This document describes the Medallia Adapter, which functions as such a third party channel adapter. The available channels are at present SMS, transported via our message gateway, and in the future, for customers who have arranged the necessary contractual agreements with us and the channel providers, Rich Communication channels transported via our Rich Communication API. First planned there are `text_message` and `choice_message` messages according to the Universal Profile 2.0 RCS standard, in the RCS, KakaoTalk and Line channels.

The channel adapter has two functional parts, a **sending API** taking `HTTP POST` json requests with authentication using an `access_token` parameter on the URL (which corresponds to an API Key for the user account in question), and a **callback service** transferring delivery status receipts and incoming reply messages to a url pointing to the Medallia system instance, signing the callback messages with a signature secret also supplied by the Medallia system instance.

The /send endpoint takes a json document in the `HTTP POST` request body, and returns a response in the `HTTP` response body as a json document. The callback service makes `HTTP POST` requests with a json document in a format specified by Medallia to a given url, and expects a `200 OK` empty response. The formats of these input and output json documents and the input url query parameter are described below. Formats of the callback messages into the Medallia system are also described below.

Authentication method available for requests

Every request to the sending service must include authentication, i.e. the equivalent of username and password. To conform to the standard of the Medallia software, the only allowed authentication method for the Medallia adapter sending endpoint is the following:

1. An API key generated in the Web Portal is used as a query parameter `access_token`, 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.

Callback to Medallia is authenticated, and content integrity verified, by adding a header containing the `HMAC SHA-1` signature of the `HTTP POST` body using a secret supplied by Medallia, as described below.

Chapter 1. The /send/{channel} endpoint

This endpoint is one of the two parts of the Medallia adapter. It is used by the Medallia software for sending a message to one recipient.

1.1. POST /send/sms request example

Probably from the Medallia application. The ordinary application is two-way SMS, i.e. a number from the customer's number pool will be the sender id shown on the recipient phone, and replies will be to that number.

```
https://secure.lekab.com/medallia/send/sms?access_token=TUxVOmRHVnpkSFZ6WlhJOjlKUUczTX
U2TVZVU1Exd3Y
```

With the contents of the HTTP body following the conventions of the Medallia software:

```
{
   "recipient" : {
      "id": "+491721234567"
   },
   "message": {
      "text": "Haben Sie Ihren heutigen Einkauf genossen?"
   },
   "notification_type": "REGULAR"
}
```

1.2. POST /send/sms request example for one-way messaging

Probably from the Medallia application. The ordinary application is two-way SMS, i.e. a number from the customer's number pool will be the sender id shown on the recipient phone, and replies will be to that number.

If one-way SMS is required, the Medallia system must supply a sender id to be used. In many countries, that id can be alphanumeric, but in some jurisdictions, that alphanumeric sender id must be registered, while other demand a (usually pre-registered) phone number as the sender id. Different countries have different rules about what is a campaign (which must be pre-approved) and what is one-to-one communication (which is often less restricted).

In the following example, the sender id "Shop Ltd" is requested. The SMS standard allows a maximum of 11 characters (spaces included) in a sender id, and results vary when characters outside western European alphabets are used. Note url encoding of the space character ("%20").

```
https://secure.lekab.com/medallia/send/sms?from=Shop%20Ltd&access_token=TUxVOmRHVnpkSF
```

```
Z6WlhJOjlKUUczTXU2TVZVU1Exd3Y
```

With the contents of the HTTP body following the conventions of the Medallia software:

```
{
   "recipient" : {
      "id": "+491721234567"
   },
   "message": {
      "text": "Herzlich willkommen zurück!"
   },
   "notification_type": "REGULAR"
}
```

1.2.1. Explanation of parameters to /send/sms

Note that json key names are case sensitive, so that "Recipient" is not a valid substitute for "recipient".

key	value	
access_token	URL query parameter	API Key for the sending user account in the messaging gateway
from	URL query parameter	SMS sender id (if present and non-empty, one-way SMS is used)
recipient	object	json object containing the recipient id
id	string	recipient phone number, E.164 globalized (RFC2806), "plus and country code"
message	object	specification of the message to send
text	string	the text of the sms
notification_type	string (optional)	any value will be ignored

1.3. POST /send/rcs request example (future Rich extension)

probably from the Medallia application

```
https://secure.lekab.com/medallia/send/rcs?access_token=TUxVOmRHVnpkSFZ6WlhJOjlKUUczTX
U2TVZVU1Exd3Y
```

With the contents of the HTTP body following the conventions of the Medallia software:

```
{
    "recipient" : {
       "id": "+491721234567"
    },
    "message": {
       "text": "Haben Sie Ihren heutigen Einkauf genossen?",
       "quick_replies": [
          {
            "content_type": "text",
            "title": "Jawohl!",
            "payload": "YES"
          },
          {
            "content_type": "text",
            "title": "Gar nicht!",
            "payload": "NO"
          }
       ]
    },
    "notification_type": "REGULAR"
 }
```

1.3.1. Explanation of parameters to /send/rcs, /send/kakaotalk etc

Note that json key names are case sensitive, so that "Recipient" is not a valid substitute for "recipient".

key	value	
access_token	URL query parameter	API Key for the user account sending the message
recipient	object	json object containing the recipient id
id	string	recipient phone number, E.164 globalized (RFC2806), "plus and country code", for LINE channel instead the LINE ID
message	object	specification of the message to send
text	string	the text of the `text_message` or `choice_message`
quick_replies	array of objects (optional)	rich message choice buttons in `choice_message` if present (ignored for sms)
content_type	string (optional)	must be "text", empty string or not present
title	string	text on the choice button (cannot be empty)
payload	string (optional)	text to send back to Medallia when button is pressed, defaults to title
notification_type	string (optional)	any value will be ignored

1.4. Responses to /send/sms

1.4.1. HTTP successful response to /send/sms

A successful request will return 200 OK and a json document of the following format.

Note that the request for sending may be successful even if the SMS sending later fails due to external factors (phone turned off, phone subscription expired, no such subscriber). The subsequent overall fate of the SMS is not returned as an error, but later via the delivery status callback functionality (see below).

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

```
{"recipient_id":"+491721234567","message_id":"1695262"}
```

1.4.2. HTTP failed response to /send/sms

A failed request will return a failing error code (400, 401, 403, 404 or 500) and a json document of the following format.

If the request succeeds, but SMS sending later fails due to external factors (phone turned off, phone subscription expired, no such subscriber), the subsequent overall fate of the SMS is not returned as an error, but later via the delivery status callback functionality (see below).

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

```
{"error":"Unauthorized"}
```

1.4.3. Explanation of response to /send/sms

json key	json value	
recipient_id	string	recipient phone number, E.164 globalized (RFC2806), "plus and country code", for LINE channel instead the LINE ID
message_id	string	A unique message id (may be up to 50 characters), referenced in all callback messages
error	string	Error message from the adapter

The response will either contain the fields `recipient_id` and `message_id` (successful) or `error` (unsuccessful). The `Content-Type` header of the response is `application/json` for all responses.

1.5. Setup of /send/{channel} and callback on Medallia

In the "channel definition", the parameter `Customer send message URL` should be set to the adapter **url**, e.g.

```
https://secure.lekab.com/medallia/send/sms
```

Likewise, in the "channel definition", the parameter `Token` should be set to the **api key** of the sending user account in the message gateway e.g.

```
TUxVOmRHVnpkSFZ6WlhJOjlKUUczTXU2TVZVU1Exd3Y
```

For the callback functionality (see below), three strings must delivered (once, when setting up the channel) from the Medallia system administrator to our customer service, for inclusion in the database table as described in the following paragraphs.

The three necessary strings are the **Instance URL** of the Medallia system of the customer, e.g.

```
https://conversations.fra1.medallia.eu/cg/mc/custom/abcdef01-2345-6789-0abc-
defabcdef012
```

and the **Page Id** inside Medallia of the customer application using Medallia, e.g.

```
Pg123456AcmeCustom
```

and, finally, the **Signing secret** used for SHA-1 HMAC signature of the callback `HTTP` body, for authentication of callback messages to be accepted at this Instance URL, e.g.

```
AbcdEFGH+IJJ4~%GmJ$abcdefgh*qv12345
```

1.6. Setup of /send/sms and callback in the messaging gateway

Setup for SMS sending inside our system begins with standard two-way API user setup with receipt and incoming **callback** set to `MEDALLIA` with the user id as "callback url". The sending user account must have the roles `Api` and `Two-way` and the user’s default **number pool** should contain a single dedicated number, so that conversations are kept together in the recipient phone. Number pool switching, e.g. for different country prefixes, is recommended when the customer needs different senders for different countries, and this is set up in the database table `t_number_pool` with each of the alternate number pools also containing a single number.

Further send channel parameters and setup of Medallia callback (see below) is done in the database table `t_medallia_settings` (see next paragraph) where a database row, with the composite key consisting of the (integer) user id and the (string) channel "SMS", contains all further settings of the Medallia adapter for the customer. The three strings needed from Medallia for each customer channel (above) go in this database row.

Note that the SMS sending (as opposed to the callback) functionality of the adapter will work also when no row with the proper user id and "SMS" channel is present. Not so for the future extension to Rich channels, which will have to be defined each in a row in this table, and also in the Rich authorization tables. Such setup will also need further contractual agreements with us and with the respective channel administrative bodies.

1.6.1. All columns in the `t_medallia_settings` database table

column	type	
c_userid	long integer	id of the user account, part of table primary key
c_channel	string (capital letters only)	SMS (or later KAKAOTALK etc), part of table primary key
c_version	long integer	for internal use, set to 0 when creating a new row
c_url	string from Medallia	the full instance url for the Medallia instance where the customer is defined
c_pageid	string from Medallia	an internal id for the customer and channel used inside Medallia
c_secret	string from Medallia	SHA-1 HMAC secret used for signing callbacks
c_readnotif	Y or N (default N)	a second "delivered" receipt is sent on READ status from a Rich channel (ignored for SMS)
c_checknumber	Y or N (default N)	SMS only, do not try to send if non-mobile number according to Google library
c_smsreplchars	Y or N (default N)	SMS only, edit message to replace with cheaper characters (needs role, contact sales)
c_smsvalidmin	integer (default 1440)	SMS only, stop retrying from operator to phone after time (if operator honors)
c_richappname	string (default "")	Rich channels only, used for customers with different apps in Rich Content Channels API

Chapter 2. Callback to Medallia of delivery status and incoming replies

This is the second part of the adapter, responsible for the callback of message delivery status and incoming reply messages to the Medallia system.

When a message has been sent via the `/send/sms` (etc.) endpoint, delivery status events related to the message generate `HTTP POST` callbacks of json documents to the instance url that was set for the user account and channel in the `t_medallia_settings` database table. If no pre-set url is defined, no message delivery status reports will be sent back to the Medallia system, but the message will anyway be delivered to the recipient (or not, if there is a problem).

When the recipient responds, the same url will receive a json document in a `HTTP POST` callback containing the response and referring to the id of the original question.

The listening service at the instance url 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.

2.1. Examples of callback of message delivery status

2.1.1. Callback of successful message delivery status

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

```
{
  "object" : "page",
  "entry" : [ {
    "id" : "Pg123456AcmeCustom",
    "time" : 1672912663747,
    "messaging" : [ {
      "sender" : {
        "id" : "+491721234567"
      },
      "recipient" : {
        "id" : "Pg123456AcmeCustom"
      },
      "timestamp" : 1672912663747,
      "delivery" : {
        "mids" : [ "1695530" ],
        "status" : "delivered"
      }
    } ]
  } ]
}
```

The message will have HTTP headers which include

```
Content-type: application/json
```

and the HMAC SHA-1 signature of the entire UTF-8 encoded HTTP body using the supplied secret (example is not accurate)

```
X-Hub-Signature: sha1=c2bfc5766f1625f48562bc91beeb2ca5a6386ff3
```

Note that the LINE channel does not send successful delivery status receipts, but only unsuccessful in such cases.

2.1.2. Callback of unsuccessful message delivery status

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

```
{
  "object" : "page",
  "entry" : [ {
    "id" : "Pg123456AcmeCustom",
    "time" : 1672912663747,
    "messaging" : [ {
      "sender" : {
        "id" : "+491721234567"
      },
      "recipient" : {
        "id" : "Pg123456AcmeCustom"
      },
      "timestamp" : 1672912663747,
      "delivery" : {
        "mids" : [ "1695530" ],
        "status" : "undelivered",
        "error" : {
          "code" : 4,
          "name" : "Expired",
          "message" : "Recipient unavailable for 24 hours"
        }
      }
    } ]
  } ]
}
```

The message will have HTTP headers which include

```
Content-type: application/json
```

and the HMAC SHA-1 signature of the entire UTF-8 encoded HTTP body using the supplied secret (example is not accurate)

```
X-Hub-Signature: sha1=c2bfc5766f1625f48562bc91beeb2ca5a6386ff3
```

Note that the LINE channel does not send successful delivery status receipts, but only unsuccessful in such cases.

2.1.3. Explanation of fields in the delivery callback json object

json key	json value	
object	string	always "page"
entry	array of objects	always one single object
id (outer)	string	Medallia page id
time	long integer	Unix timestamp millis after epoch for delivery event
messaging	array of objects	always one single object
sender	object	actually refers to the message recipient
id (in sender)	string	recipient phone number, E.164 globalized (RFC2806), "plus and country code", for LINE channel instead the LINE ID
recipient	object	actually refers to the page id of the Medallia sender
id (in recipient)	string	Medallia page id
timestamp	long integer	Unix timestamp millis after epoch for delivery event
delivery	object	presence of this field **identifies this as delivery receipt**
mids	array of strings	always one single string, internal message id (may be up to 50 characters)
status	string	failed, sent, delivered, or undelivered
error	object	only present for failed and undelivered status
code	integer	our error code of delivery failure
name	string	our error name of delivery failure
message	string	error message from operator, if any

2.2. Example of callback of incoming response message

This is a callback message informing of an incoming answer to a previously sent message:

```
{
  "object" : "page",
  "entry" : [ {
    "id" : "Pg123456AcmeCustom",
    "time" : 1672912936212,
    "messaging" : [ {
      "sender" : {
        "id" : "+491721234567"
      },
      "recipient" : {
        "id" : "Pg123456AcmeCustom"
      },
      "timestamp" : 1672912936212,
      "message" : {
        "mid" : "1695530",
        "text" : "Ja"
      }
    } ]
  } ]
}
```

The message will have `HTTP` headers which include

```
Content-type: application/json
```

and the `HMAC SHA-1` signature of the entire UTF-8 encoded `HTTP` body using the supplied secret (example is not accurate)

```
X-Hub-Signature: sha1=c2bfc5766f1625f48562bc91beeb2ca5a6386ff3
```

2.2.1. Explanation of fields in the incoming response message callback json object

json key	json value	
object	string	always "page"
entry	array of objects	always one single object
id (outer)	string	Medallia page id
time	long integer	Unix timestamp millis after epoch for incoming event
messaging	array of objects	always one single object
sender	object	refers to the sender of the incoming message
id (in sender)	string	sender phone number, E.164 globalized (RFC2806), "plus and country code", for LINE channel instead the LINE ID
recipient	object	refers to the page id of the Medallia application
id (in recipient)	string	Medallia page id
timestamp	long integer	Unix timestamp millis after epoch for incoming event

json key	json value	
message	object	presence of this field **identifies this as incoming response**
mid	string	Internal message id of the outgoing message to which this is a response (may be up to 50 characters)
text	string	text of the incoming message, for SMS what was actually keyed into the mobile phone, for Rich buttons the payload, if present, or else title

2.3. Signature of the callback messages

To validate the author and integrity of the callback information, every callback HTTP POST request has a signature in a `HTTP` header `X-Hub-Signature` containing the `HMAC SHA-1` signature of the entire UTF-8 encoded message `HTTP POST` body, using the secret received from Medallia for the instance url, as entered into the `t_medallia_settings` database table.

The following Java code is used to generate the signature.

```java
private static String signature(final String data, final String secret) {
    String signaturehex = "";
    try {
        SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(StandardCharsets
.UTF_8), HMAC_SHA1_ALGORITHM);
        Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
        mac.init(signingKey);
        byte[] signatureBytes = mac.doFinal(data.getBytes(StandardCharsets.UTF_8));
        signaturehex = Hex.encodeHexString(signatureBytes);
    } catch (Exception e) {
        return "";
    }
    return "sha1=" + signaturehex;
}
```

Chapter 3. Medallia Adapter Implementation details

3.1. Medallia Adapter Documentation used

The Adapter was implemented according to specifications in the document

```
Medallia Conversations Custom Channel Integration 2022-10-10.pdf
```

3.2. Implementation limitations and conventions

The fields `notification_type` and `sender_action` in the /send/{channel} json request body, as well as any other fields not listed above, are allowed but completely ignored. All json keys must have the capitalization described, i.e., "Recipient" is not a valid alternative to "recipient".

Medallia requests for `profileData` are not supported by the adapter implementation, and the corresponding `HTTP GET` endpoint will not be found, so layers outside the adapter implementation are likely to respond with a 404 Not found.

Medallia `End-of-conversation` signal and `Typing` signal requests are recognized but not supported by the implementation, and will result in a 400 "Message empty or null" error message, because such requests have a structure similar to a send request missing a message text. To avoid these errors, set "disableEndSignal" in the Medallia account settings, and "Typing off" in the Medallia Custom channel properties.

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

As mentioned above, Rich messaging channels RCS, KakaoTalk and Line are not yet available, but in an advanced stage of preparation.

Note that the Line channel uses a special Line app id instead of a phone number, and the id should not generally be preceded by a + sign or country code. Also, the Line channel does not send successful delivery receipts, but only unsuccessful in such cases.

3.3. Supported character sets in the resulting SMS and pricing issues

The character set used in the input to the API does not affect the resulting SMS. Instead, the characters that are requested to be sent will determine the size and pricing of the message.

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

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

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

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

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

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

3.4. Potential problems with callback of Rich channel READ receipts

In contrast to SMS, where it is unknown whether the recipient has opened and read the message, many Rich communication channels send one receipt upon message delivery, and a second receipt when the message has been read by the user. The specification for Medallia receipts says that a successful delivery has only one kind of format, without any way of marking whether it pertains to READ or DELIVERED.

Therefore, we offer two options via the flag `c_readnotif` in the database table `t_medallia_settings`, each with their own disadvantages.

The first, and default, with flag value `N`, is to not confuse Medallia with two `delivered` receipts for the same message. The disadvantage of this is that our system will only deliver callbacks of messages statuses that are of a higher level of finality than earlier callbacked statuses. A READ callback has higher finality than a DELIVERED callback. That means that if, for some reason, the operator sends the READ receipt before the DELIVERED receipt (which can indeed happen), we mark the message as READ, but do not send any callback due to the flag being set to `N`, and when we get the DELIVERED receipt we do not send any callback, because of the status already being the more final READ. So in those instances, no receipt callback arrives.

The second, with flag value `Y`, gives at least one, but most of the time two `delivered` receipts for the same message. The Medallia service provider will have to ensure that this does not cause confusion in the Medallia system.