API specification for SMS/MMS/IVR messaging gateway

version 1.6

Piotr Isajew (pki@ex.com.pl)

September 18, 2015

www.przypominamy.com

Contents

1. Introduction

The API utilises simple XML commands encapsulated in HTTP POST requests. There is also an option to pass XML API commands in SMTP Internet mail messages.

Basic API features include:

- sending SMS messages from random 9-digit MSISDNs (Poland only)
- sending SMS messages from predefined source addresses (both numeric and alphanumeric, but subject to destination network operators' allowance)
- sending MMS messages from random 9-digit MSISDN's (Poland only)
- Text-To-Speech fallback when sending SMS message to PSTN network (Poland only)
- outgoing message status change updates (when configuration allows)
- receiving responses for messages sent through the system (when configuration allows)
- sending voice (Text-to-Speech) messages and implementing IVR sessions (when configuration allows)

Please contact us to verify which of above functions can be available for you.

By default the API is set up for unidirectional communication – transactions are being initiated by the client and synchronously confirmed by the gateway. This setup does not allow receiving incoming messages.

Upon request it's possible to configure a bidirectional communication, which allows sending asynchronous notifications to the client. Both status update and incoming message notifications can be sent (see 2.4).

1.1 Configuration parameters

The following parameters need to be agreed to in order to obtain HTTP access to the API:

- user login and password to be used to authenticate client request[1]
- IP addresses (max. 5) from which the requests can originate .

If the customer wishes to utilise asynchronous notifications, he should also define the URL address that notifications should be sent to (see 2.4).

[1] Note those are different from login and password you use to access web control panel.

1.2 Character encoding

Unless explicitly stated otherwise, all API messages should use UTF-8 character encoding.

1.3 Date representation

Unless explicitly stated otherwise, dates are given in YYYY-MM-DD HH:MM:SS format, using gateway local time zone (CET/CEST).

2. Request format

Requests are passed as XML objects encapsulated in HTTP POST messages. Gateway endpoint, to which messages should be delivered is: `http://api.superbramka.pl/rbroker`

2.1 Communication from client to the gateway

All requests sent from the client to the gateway should have the form given below:

```
<?xml version="1.0" encoding="UTF-8"?>

<gateway-request type="...">
  <request-auth>
    <partnerLogin>testUser</partnerLogin>
    <partnerPassword>testPassword</partnerPassword>
  </request-auth>

  <request-data>
     ...
     ...
     ...
  </request-data>
</gateway-request>
```

Where **type** attribute defines the type of commands that are being sent in request (in other words – XML entities that are allowed to appear in the **request-data** part).

Allowed command types are shown in table 2.1.

If the XML command, that client has sent does not have valid syntax, there is authentication error, or any other failure that cannot be properly handled on a single message level, server responds with a plain text message — the word **ERROR**, optionally followed by a descriptive text.

If request is valid, then gateway should respond with an XML message:

Command type	Description
sms-send-request	SMS message send request
mms-send-request	MMS message send request
ivr-send-request	Send TTS message or initiate IVR session
sms-status-request	query outgoing SMS message status
mms-status-request	query outgoing MMS message status
ivr-status-request	query outgoing TTS/IVR message status

Table 2.1: Allowed command types for commands sent by the client

Message	Schema location
sms-send-request	http://www.superbramka.pl/ns/1_3/sms-send-request.xsd
sms-status-request	http://www.superbramka.pl/ns/1_3/sms-status-request.xsd
mms-send-request	http://www.superbramka.pl/ns/1_3/mms-send-request.xsd
mms-status-request	http://www.superbramka.pl/ns/1_3/mms-status-request.xsd

Table 2.2: Schema locations for API commands

```
<?xml version="1.0" encoding="UTF-8"?>

<gateway-response type="...">
   ...
   ...
   ...
</gateway-response>
```

The **gateway-response** element contains children referring to commands sent in the **request-data** part. The **type** attribute determines type of the elements that make up the response and it can have values give below:

sms-send-response response for SMS message send request

sms-status-response response to SMS message status query

mms-send-response response to MMS message send request

mms-status-response response to MMS message status query

ivr-send-response – response to IVR message send request

ivr-status-response – response to IVR message status query

2.1.1 Request validation

The customer can validate commands to be sent to gateway using any XML validator capable with validating using W3C recommended schema definitions (xsd). The schema definitions are published under the addresses given in table 2.2.

2.1.2 Sending SMS message

When sending SMS messages, the **gateway-request** message should use **sms-send-request** type consist of one or more **sms-send-request** commands in the **request-data** section.

Example:

```
<sms-send-request>
  <!-- required part -->
  <message-id>12346562</message-id>
  <msisdn>601601601</msisdn>
  <sms-text>Example of very simple send-request</sms-text>
  <!-- optional part -->
  <dont-send-before>2010-03-01 17:00</dont-send-before>
  <tts-text>Text</tts-text>
  <src-addr>Test</src-addr>
</sms-send-request>
```

Meaning of fields in the request if given below:

message-id it's a **unique** 63-bit unsigned integer that uniquely identifies the message for the client system

msisdn the destination number (preferred format: `+48123456789`)

sms-text message text[1]

dont-send-before specifies the earliest time the message can be sent[2].

tts-text provides an alternative text to be read through the text-to-speech synthesizer if the system determines, that destination number is in PSTN network (currently TTS messages are limited to 4000 characters)

src-addr allows client to set source address (numeric or alphanumeric) from which the message is to be sent (note that this feature need to properly configured in the gateway in order to work); if this field is missing system will default to send from random number

2.1.3 Sending MMS messages

The MMS message send request can have different forms, depending of how the message is defined. General structure is as follows:

```
<mms-send-request>
   <message-id>4444</message-id>
   <msisdn>48501501501</msisdn>
   <message-template>
     <!-- template specification -->
   </message-template>

   <message-subject>The subject</message-subject>

   <message-data>
     <!-- message contents -->
   </message-data>
</mms-send-request>
```

Message template

The system requires outgoing messages to have a SMIL template defined. One option is to pass a reference to one of templates defined in web control panel. Another is to pass SMIL definition directly in the command.

If using a template from web panel, a **message-template** part should contain the id of SMIL template, i.e.:

```
<message-template>
   <template-id>11</template-id>
</message-template>
```

It's also possible to define SMIL inside the command, as follows:

[1] Can contain either characters from GSM 7-bit character set, or UTF-8 encoded characters.
[2] Depending on system load the message can be sent later.

```
<message-template>
   <smil-data>
      <smil>
        <head>
          <layout>
            <region id="a" top="0" left="0" height="100%"
                  fit="meet"/>
          </layout>
        </head>
        <body>
          <par>
             <img src="pict1.jpg" region="a"/>
          </par>
        </body>
      </smil>
   </smil-data>
</message-template>
```

The only limitation is that we require the source url's for multimedia content (*src* attributes) to be given as file names (without paths, domains, etc.) with valid extensions. Those names don't have to refer to valid files on client side — they are just virtual names to identify content on server side.

Message data

It's required to define a data definition for each content object defined in SMIL template. Message definitions are to be given in **message-data** part of the command, as **message-part** entities:

```
<message-data>
   <message-part type="specification type" name="object name">
      <!-- object contents -->
   </message-part>
</message-data>
```

The **name** attribute should be the same as file name used to declare that object in SMIL template (i.e.: *pict1.jpg*).

The **type** attribute makes it possible to define the method to be used to obtain the contents of the object. The following values are allowed:

base64 binary contents of the object is given in the **message-part** element, base64 encoded.

href **message-part** contains an URL address from which the binary contents can be downloaded to the systembiektu

text the object is plain text object and it's text is given directly in the **message-part** element

The system does not, in any way limit the size of objects used to compose the multimedia message, it's however recommended not to exceed the reasonable sizes resulting from destination networks' specifications. Total size of a single outgoing message is currently administratively limited to 280 kilobytes.

2.1.4 Outgoing message status query

Query request should contain in it's **request-data** part, one or more ***-status-request** messages, i.e.:

```
<sms-status-request>
 <message-id>1243124314</message-id>
</sms-status-request>
```

where **message-id** is a 63-bit unsigned integer uniquely identifying outgoing message in client system.

2.2 Messages sent from gateway to the client

Messages sent from the gateway to the client have the following form:

```
<?xml version="1.0" encoding="UTF-8"?>

<gateway-response type="...">
  ...
  ...
</gateway-response>
```

where **type** attribute determines type of elements sent in the response. Currently it can have one of the following values:

sms-send-response means that **gateway-response** contains **sms-send-response** elements

sms-status-response means that **gateway-response** contains **sms-status-response** elements

mms-send-response — **gateway-response** contains (**mms-status-response**) elements

mms-status-response — **gateway-response** contains **mms-status-response**

status-notification used for asynchronous notifications; in that case **gateway-response** can contain any ***-status-response** elements

2.2.1 Message send request response

As a response to a message send request, the system sens a response element, that contains the following fields:

message-id is a 63-bit unsigned integer, identifying the message in client system

result operation status code

in example:

```
<sms-send-response>
  <message-id>121212</message-id>
  <result>OK|ERROR</result>
</sms-send-response>
```

Meaning of the **result** field is as follows:

OK message has been correctly accepted for sending

ERROR message submission has been rejected

2.2.2 Status request response

An example of status response message is given below:

```
<gateway-response type="sms-status-response">
  <sms-status-response>
    <message-id>12346562</message-id>
    <part>0</part>
    <status>pending|processing|sent|confirmed|paused|failed|not found</status>
    <status-time>YYYY-MM-DD HH:MM</status-time>
    <sent-time>YYYY-MM-DD HH:MM</sent-time>
    <delivered-time>YYYY-MM-DD HH:MM</delivered-time>
  </sms-status-response>
</gateway-response>
```

The **part** element identifies part of message that was to long to be sent as one entity, so it has been sent in separate (unrelated) parts. Note that this has nothing to do with concatenated sms messages.

The **status** field determines the current status of the message and can have one of the following values:

pending message has been accepted for sending, but not yet processed

processing the system is currently processing that message

sent message has been sent out from the system

confirmed the delivery confirmation has been received from the destination network

paused message has been administratively paused

failed message sending has permanently failed

The **status-time**, **sent-time**, and **delivered-time** elements contain time stamps related to message processing. Their meaning is as follows:

status-time time of last message status change

sent-time message sent time

delivered-time time of delivery receipt

2.3 Additional notes

2.3.1 Message sending time

If send request does not contain **dont-send-before** field, send attempts will be retried at regular intervals, until any final state is reached.

If **dont-send-before** is defined in past, it will be adjusted to current date.

2.3.2 Delivery notification receipts

All SMS messages sent through the system are being sent with delivery notification request bit set. When delivery report is received for sent message, **delivered-time** field is set to the value given by that report. Please note however that technical implementation of delivery reports is varying by network, handset, and message type, and specifically:

- some handsets may not generate reports for some types of messages
- some routes only give report with 1-minute accuracy (so it may happen that message being sent at 13:30:31 will be reported as delivered at 13:30:00)
- most reports are sent upon message being delivered to the handset (not necessarily being read by user)
- there is no time zone information in delivery reports

Upon customer request the system can be configured to sent asynchronous message status change updates to the given URL. Those notifications are sent as XML documents encapsulated in HTTP POST requests. Root element is always one of ***-status-response**.

During normal operation each notification is being sent only once. There is no obligation that server will retry notification transmission in case of errors, client system should however be prepared to the situation, when it receives the same notification more than once.

2.3.3 Message size limitation

To achieve maximum performance, command XML's are limited to no more than 1000 commands in one XML document.

2.4 Bidirectional communication

In order to use bidirectional communication, the customer should define an URL address, to which notifications and/or incoming messages are to be sent.

If that URL is given, the gateway will send notifications (as described in 2.2) encapsulated in HTTP POST requests.

If client side responds to such POST request with a HTTP 200 status code – it's assumed that notification has been properly delivered and wont be retransmitted anymore.

2.4.1 Receiving responses to outgoing messages

There is an additional **gateway-response** type — **incoming-message** which is used for XML documents containing incoming messages.

The **incoming-message** gateway response element can contain any of the following entities:

incoming-sms-message incoming SMS message

incoming-ivr-message action report for IVR session

Incoming messages can contain the following fields:

message-id 63-bit unsigned integer identifying message to the client system

msisdn source number, from which the message has been received

received-time message receive time

sms-text message text

3. Voice messages

Since version 1.4 API supports voice messages. Those messages can be sent both to mobile and fixed networks.

This feature can be used for sending both simple Text-To-Speech messages, and initiate more complicated IVR sessions. The latter requires an IVR scenario to be defined in gateway.

In the rest of this chapter it's assumed that the reader understands API operations in respect to SMS and MMS messages as described in chapter 2.

3.1 Sending TTS/IVR messages

TTS/IVR message can be sent by using **gateway-request** message of **ivr-send-request** type.

The **ivr-send-request** element has the following fields:

message-id, msisdn, dont-send-before — like for other types of messages

scenario-id optional id of IVR scenario (as defined in web interface)

tts-argument text argument to be used by the speech synthesizer – number of these elements should match number of dynamic parameters in scenario.

3.1.1 Sending a simple TTS message

To send a simple TTS message it's required to:

1. Skip the **scenario-id** element.
2. Use exactly one **tts-argument** element.

The system will dial the destination number and read a synthesized message which text is given in the **tts-argument** field.

3.1.2 Initiating IVR session

IVR sessions are available, if configured by the service provider. Upon configuration of this feature, service provider assigns a scenario id to the customer.

Depending on the scenario configured, client side may be required to provide one or more **tts-argument** fields in each **ivr-send-request** message.

3.2 IVR message extended status

Extended status information is available for IVR session calls – it's not possible to get it for simple TTS messages. Client should prepare a receiving script according to information from chapter 2.4.1. It's not possible to obtain extended information using **ivr-status-request** calls.

Client script will receive **incoming-ivr-message** elements consisting of the following fields:

message-id unique 63-bit unsigned integer identifying the message on server side

client-message-id unique 63-bit unsigned integer identifying message on client side (that id is given by the client in **ivr-send-request**)

received-time time when the call had been performed

ivr-path (option) sequence of DTMF digits selected by user during in-call navigation

voice-recording (optional element) delivers a file with recording of the ivr session (see 3.2.1)

3.2.1 Recording IVR session

There is an optional feature that, once configured, allows recording of IVR sessions. Sessions are recorded into a WAV files. Currently the only way to deliver such a recording to the client is by a **voice-recording** field in extended status message.

The **voice-recording** element has **type** attribute, which indicates how to access the recording. Currently the only supported value of that attribute is *base64* which means that the file is base64 encoded and delivered directly in the **voice-recording** element.

4. Diagnostics

There are two levels of error reporting to the client: generic and detailed.

4.1 Generic reporting

Generic reporting is used to report errors in situations like:

- communication attempt from IP address which is not allowed, or using invalid authentication data
- XML parser failures
- other unhandled errors in server code

Generic errors are reported by replying to HTTP POST request with a plain text message which first line contains the single word ERROR. In second line there may be more descriptive information about the nature of the error that had occurred.

If it's not obvious, from the error message, what client should do to fix the situation, and the situation is repeatable, it's necessary to inform the service provider about that error.

4.2 Detailed reporting

Detailed reporting is used to report problems that relate to specific ***-send-request** commands.

Detailed error information is delivered inside the relevant ***-send-response** element.

In case of error **result** field of ***-send-response** element contains text `ERROR`. Additionally there may be an **error** element, containing more detailed information about the error.

Example:

```
<gateway-response type="ivr-send-response">

  <ivr-send-response>
    <message-id>21</message-id>
    <result>ERROR</result>
    <error>
      <code>201</code>
    </error>
  </ivr-send-response>

</gateway-response>
```

Code	Description
100	unknown error (please contact technical support if you get this frequently)
	permanent errors
200	permanent error, unknown reason, do not try to resend the message
201	incorrect destination number
202	incorrect source address (i.e. you are trying to send from alphanumeric sender that has not been configured for you)
203	invalid characters in message text
204	message text length limit exceeded
205	incorrect value of **campaign-id** element
206	MMS message size limit exceeded
207	message with that message-id has been submitted earlier
210	message type is not allowed
220	required parameter is missing
	temporary errors
300	temporary error, unknown reason, you may try to resend the message later
301	no more message credits to use, depending on account type, you need to buy more credits or contact your service provider to increase your message limit

Table 4.1: Error codes used by XML API

The **error** element does always contain **code** field – which gives numeric code of an error. Additionally it may contain an **info** field – containing textual description of an error.

Error code is a 3-digit integer value. Both temporary and permanent error situations are reported. Error codes that designate temporary errors have a digit 3 as their most significant digit. Any other codes should be treated as a permanent errors.

Detailed list of error codes used by current API version is shown in table 4.1.

5. Email encapsulation

Upon customer request, API can be configured to use email based encapsulation instead of HTTP one.

Request formats are the same as described in chapter 2.

If using this type of communication, API messages are passed in the body of e-mail messages sent between gateway e-mail address and the e-mail address provided by customer.

Message subject is ignored. Gateway sends e-mails with an empty subject field.

Message body cannot contain any data besides of XML request and digital signature.

When using e-mail encapsulation it's required to secure the communication using PGP, as described on page 17.

In order to configure e-mail access the customer should contact service provider with the following data:

- e-mail address, which will be used for both sending messages to the gateway, and receiving API responses
- public key to be used for communication

It's also up to the customer to configure his e-mail system, including anti-spam filters, to properly receive messages from the gateway.

5.1 Errors and retransmission

Internet mail is asynchronous from it's nature and it may happen that message will wait several hours in server queue before being delivered. Gateway will process such message when it will be delivered to it from e-mail system, and it will send a reply e-mail then. Reply e-mail may be, by itself a subject of several mail handling delays (i.e. greylisting, etc.).

Because of that it's not recommended for client application to retry commands at API level before a reasonable timeout (at least 1 hour) passes without reply from server side.

The Gateway does not ever expect client application to confirm any XML message sent to the client using e-mail encapsulation nor it retransmits such messages in case of failure. It's expected that any communication failures will be properly handled by SMTP queuing and retransmission mechanisms.

6. Securing communication

6.1 HTTP communication

Gateway API does not, in any way, encrypt or verify authenticity of requests received from client system. It is however possible, upon client request, to configure an IPsec tunnel between the gateway and client's application. Please contact technical support if you would like an IPsec setup.

6.2 e-mail communication

We **require** e-mail communication to be secured by digitally signing messages with OpenPGP (or compatible solution). Customers are required to provide their public keys during connection setup. Messages should be signed with a key generated for the e-mail address from which they are sent.

Current API version does not allow PGP/Mime to be used. Digital signatures should be put inline inside the message body.

Digital signatures of messages sent from the gateway can be verified by a public key, that is available for download at: `http://api.superbramka.pl/gpg/api_pubkey.asc`.

Upon client request e-mail communication can be encrypted with PGP.

7. Examples

This chapter contains some examples for using API features from several programming languages. For simplification examples in this chapter do not contain any error handling code (note however, that it's not a good idea to skip this part in a production grade application).

7.1 Sending SMS from Unix command line using wget

It's possible to send simple requests to the gateway directly from command-line, using tools like *wget* or *curl*.

Consider the following example:

```
wget -O- -q --post-file=req.xml http://api.superbramka.pl/rbroker
```

which causes sending the contents of *req.xml* to the gateway, and prints it's response on standard output.

If *req.xml* would contain:

```
<?xml version="1.0" encoding="UTF-8"?>
<gateway-request type="sms-send-request">
 <request-auth>
  <partnerLogin>test</partnerLogin>
  <partnerPassword>test</partnerPassword>
 </request-auth>
 <request-data>
  <sms-send-request>
   <message-id>5</message-id>
   <msisdn>+48601601601</msisdn>
   <sms-text>Example text</sms-text>
  </sms-send-request>
 </request-data>
</gateway-request>
```

the gateway response could look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gateway-response type="sms-send-response">
 <sms-send-response>
  <message-id>5</message-id>
  <result>OK</result>
 </sms-send-response>
</gateway-response>
```

which means that the message has been accepted for sending and gateway will try to send it.

7.2 Sending SMS message with PHP

In the simplest form PHP scripts could use a function like that:

```
function sendSMS($msgid, $destNumber, $text) {
  $h = curl_init();
  $request = '<?xml version="1.0" encoding="UTF-8"?> <gateway-request ' .
    'type="sms-send-request"> <request-auth><partnerLogin>test'.
    '</partnerLogin><partnerPassword>test</partnerPassword></request-auth>'.
    '<request-data><sms-send-request><message-id>' . $msgid . '</message-id>'.
    '<msisdn>' . $destNumber . '</msisdn><sms-text>' . $text .
    '</sms-text></sms-send-request></request-data>'.
    '</gateway-request>';

  curl_setopt($h, CURLOPT_URL, 'http://api.superbramka.pl/rbroker');
  curl_setopt($h, CURLOPT_RETURNTRANSFER, true);
  curl_setopt($h, CURLOPT_POST, true);
  curl_setopt($h, CURLOPT_POSTFIELDS, $request);

  $rv = curl_exec($h);

  curl_close($h);
  return $rv;
}
```

So we could use a code like that, to send an sms:

```
$res = sendSMS(5, '+48601601601', 'Test message');
```

7.3 Receiving incoming messages with PHP

You can receive incoming messages using script given below to handle POST requests from the gateway:

```
<?php
function parseInput($input) {
  $doc = new DomDocument();
  $doc->loadXML($input);

  $root = $doc->firstChild;
  if($root->nodeName == 'gateway-response' &&
     $root->attributes->getNamedItem('type')->value ==
     'incoming-sms-message') {
    for($item = $root->firstChild; $item != NULL;
        $item = $item->nextSibling) {
      if($item->nodeName == 'incoming-sms-message') {
        $msgdata = array();
        for($fld = $item->firstChild; $fld != NULL;
            $fld = $fld->nextSibling) {
          $msgdata[$fld->nodeName] =  $fld->nodeValue;
        }

        // Here we can do something reasonable with the message
```

```
        echo "Received message #" . $msgdata['message-id'] .
          " sent from " . $msgdata['msisdn'] . " text: " .
          $msgdata['sms-text'] . "<br/>\n";
      }
    }

  }
}

$in = file_get_contents('php://input');
parseInput($in);
?>
```

7.4 Sending SMS messages with e-mail encapsulation

Here we use a simple Perl program to send a request to the gateway:

```
#!/usr/bin/perl

use GPG;
use Mail::Sendmail;

$/ = undef;
$request = <STDIN>;
$gpg = new GPG;
$pass = 'MY_GPG_KEY_PASS';
$key = 'MY_GPG_KEY_ID';
$dest = 'rbroker@api.test';

$out = $gpg->clearsign($key, $pass, $request);
# Alternatively encrypt and sign
#$out = $gpg->sign_encrypt($key, $pass, $request,
#                          $dest);
die $gpg->error() if $gpg->error();

%mail = (To => $dest,
         From => 'api@customer.com',
         'Content-Type' => 'text/plain; charset="utf-8"',
         Message => $out );
sendmail(%mail) or die $Mail::Sendmail::error;
```

8. Revision history

8.1 Changes in version 1.6

Updated chapter about voice messages.

8.2 Changes in version 1.5

initial English version of the documentation.