SMS Platform HTTP API v1.8

Tadoo
10/09/2018

Revision Sheet

Release No.	Date	Revision Description
V1.8	10/09/2018	Added API Key Authorization
V1.7	24/11/2016	Added missing DLR parameters in push mode
V1.6	10/11/2016	Added new sub statuses in delivery reports
V1.5	04/11/2016	Added supported HTTP API methods
V1.4	24/08/2016	Updated Java sample code
V1.3	31/05/2016	Added .NET sample code
V1.2	03/02/2016	Updated styles & information
v1.1	13/07/2012	Added the description of the utf parameter
v1.0	12/12/2011	Initial version

Revision Sheet **2**

Introduction **5**

Step 1: Getting Started **5**

Step 2: Authentication **6**

Step 3: Submit SMS **7**
Send Simple SMS 7
Send Concatenated SMS 8
Send Flash SMS 9
Send Wap Push SMS 9
Send Bulk SMS 10
Send Personalized SMS 11

Step 3: Parse The Response **13**
Single SMS Response 13
Bulk SMS Response 14

Step 4: Manage The Delivery Reports (DLR) **15**
Delivery Reports (Pull Mode) 15
Simple Delivery Reports 15
Advanced Delivery Reports 16
Delivery Reports and SMS Cost 16
Delivery Reports (Push Mode) 17

Step 5: Get Your Balance Information **20**

Appendix A: List of Parameters And Commands **21**
Parameter originator (Mandatory) 22
Parameter mobile_number (Mandatory) 22
Parameter text (Mandatory) 22
Parameter request_delivery (Optional) 23
Parameter delivery_push (Optional) 23
Parameter concatenated (Optional) 23
Parameter flash (Optional) 23
Parameter utf (Optional) 23
Parameter get_cost (Optional) 23
Parameter d_name (Optional) 23
Parameter d_vname (Optional) 23
Parameter d_surname (Optional) 24
Parameter d_vsurname (Optional) 24
Parameter d_mobile (Optional) 24
Parameter d_nameday (Optional) 24
Parameter d_birthday (Optional) 24

Parameters d_custom* (Optional) 24
Command get_balance (Optional) 24
Command get_status (Optional) 25
Command get_status2 (Optional) 25

Appendix B: Allowed characters in "text" field 26

Appendix C: Example - Java/JSP HTTP Client 27

Appendix D: Example - .NET/ASP HTTP Client 30

Introduction

The current document describes in detail the HTTP API provided by the Tadoo SMS Platform, for Short Message submission. The interface enables client applications to send Short Messages to subscribers of mobile networks by conveying the message data in GET or POST requests according to the HTTP protocol. The requests are received and parsed by the SMS Platform and the appropriate Short Messages are generated and delivered to the mobile subscribers.

In addition, the SMS Platform supports delivery reports, i.e. messages that convey the status of Short Message transmission (e.g. confirmation that the message has been received by the recipient's handset). If a customer is enabled to use this feature, he can request delivery reports on per Short Message basis. If a submitted Short Message requests for delivery reports, the reports related to the message are collected from the mobile network / aggregator and are either forwarded via HTTP to the customer's server (push mode) or are collected by the customer's server (pull mode). Towards this, a customer that desires the "push" delivery reports feature must provide to the SMS Platform Support Team their server details (i.e. the URL of the page that handles the notifications on the customer's server) when their account is to be opened.

Step 1: Getting Started

In order to use the platform's HTTP API, you need to have or create an account (Sign Up). This is the only way to have a connection between your application and our platform.

Step 1: Sign Up for a SMS Platform account

If you do not already have a SMS Platform account, you need to Sign Up for one at the https://tadoo.aws.mcore.cloud/.

Step 2: Authentication

The platform's HTTP API requires authentication for each available action. In particular, each request requires an **Authorization** HTTP header, containing the value "Bearer <YOUR_API_KEY>".
For example, if your API key is: secret_api_key, then the required Authorization header would be:

Authorization: Bearer secret_api_key

If you do not provide an Authorization header with your API key, or if your API key does not match one registered for your account, an invalid response will be returned with an HTTP status code of 401.

Step 3: Submit SMS

The SMS Platform HTTP API supports the submission of four different types of Short Messages which are described in the following sections. In general, for GET requests, all submissions require the syntax of a URL in the following format:

https://tadoo.aws.mcore.cloud//smsp-in

For GET requests, the parameters should be placed in the URL, in a format similar to:
https://tadoo.aws.mcore.cloud//smsp-in?param1=value1&param2=value2

For POST requests, the parameters should be placed in the body of the HTTP POST request, instead of the URL.

The complete list of the supported parameters can be found in Appendix A.

Send Simple SMS

In order to send a simple SMS (up to 160 characters) only the mandatory parameters *originator*, *mobile_number* and *text* are required. In addition, in order to request for delivery report information, the optional parameter *request_delivery* must be set as true. The request format for a simple SMS submission (with request for delivery report) is0:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=<SMS originator>

mobile_number=<mobile_number>

text=<SMS text>

request_delivery=true

Example:

We want to send a text SMS to 306944444444 containing the phrase HELLO FRIEND with originator MYCOMPANY.
The GET HTTP request needed is:

https://tadoo.aws.mcore.cloud//smsp-in?
mobile_number=306944444444&originator=MYCOMPANY&text=HELLO+FRIEND&request_delivery=true

Please note that the values of the parameters must be URL encoded. So, the value "HELLO+FRIEND" of parameter "text" is the encoding of "HELLO FRIEND". The following table presents some special character mappings according to URL encoding:

Character	URL encoding
&	%26
+	%2B
%	%25
#	%23
Blank space	%20
=	%3D
?	%3F
;	%3B
\n	%0D

Table 1: Special character mappings according to URL encoding. For a more complete list follow this link <http://www.w3schools.com/tags/ref_urlencode.asp>

Send Concatenated SMS

SMS Platform HTTP interface supports the submission of long messages (up to 459 characters) as concatenated messages. In this case, the message appears as one long Short Message at the recipient's handset. In order to send a concatenated message, the optional parameter *concatenated* must be set as true. The request format for a concatenated SMS submission (with request for delivery report) is:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=value1

mobile_number=value2

text=value3

concatenated=true

request_delivery=true

Please note that the *concatenated* parameter cannot be used for flash Short Messages.

Send Flash SMS

SMS Platform HTTP interface also supports the submission of flash messages (up to 160 characters) that are presented automatically on the recipient's mobile screen. Thus the recipient does not have to enter in inbox. In order to send a flash message, the optional parameter *flash* must be set as true. The request format for a flash SMS submission (with request for delivery report) is:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=value1

mobile_number=value2

text=value3

flash=true

request_delivery=true

Send Wap Push SMS

The platform's HTTP API also provides a way to send WAP Push Service Indication SMS. All you need is to syntax the value of the text parameter in the following format:

<wap><url>url_text</url>message_text</wap>

The URL format for a WAP Push SMS submission (with request for delivery report) is:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=value1

mobile_number=value2

text=<wap><url>url_text</url>message_text</wap>

request_delivery=true

Example:

We want to send a **wap push SMS** to 306944444444 containing the phrase HELLO FRIEND and the url http://example.com/image.png, with originator MYCOMPANY.
The GET HTTP request needed is:

https://tadoo.aws.mcore.cloud//smsp-in?
originator=MYCOMPANY&mobile_number=306944444444&text=<wap><url>http://example.com/image.png</url>HELLO+FRIEND</wap>

Please note that:

- The text should be maximum 160 GSM characters long. In case of wap push SMS the xml tags are counted in the 160 characters. So, the sum of url_txt+message_text should be maximum 138 characters
- The request must be UTF-8 encoded

Send Bulk SMS

In the previous sections, the four different types of Short Messages that SMS Platform HTTP interface supports were presented. All these types can be sent in both *Single SMS per Request* (see above examples) and *Bulk SMS per Request* mode**.**

The only difference between a single and a bulk SMS submission is that in the first case the value of the *mobile_number* parameter is a single mobile number while in the second case the *mobile_number* parameter is a comma separated list of mobile numbers.

The request format for a bulk simple SMS submission (with request for delivery report) is:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=value1

text=value3

request_delivery=true

mobile_number=number1,number2,...,number100

Example:

We want to send two text SMS to 306944444444 and 306944444445 containing the phrase HELLO FRIEND with originator MYCOMPANY.
The GET HTTP request needed is:

https://tadoo.aws.mcore.cloud//smsp-in?
originator=MYCOMPANY&text=HELLO+FRIEND&request_delivery=true&
mobile_number=306944444444,306944444445

Please note that in bulk SMS submissions, your balance must be twice (or four times in case of concatenated SMS) the number of destinations. In other case, a "You have not enough balance" error code will be returned. It is also a good practice to put the mobile_number parameter at the end of the URL.

Send Personalized SMS

Finally, SMS Platform HTTP interface supports the single or bulk submission of simple, concatenated, flash and wap push Short Messages with *personalized* content for each recipient. In personalised SMS, the message body contains special keywords, in the form **${keyword}**, which are replaced during the SMS submission by their corresponding recipient specific values. The different values for each keyword are provided by using parameters in the form **d_keyword=value1,value2,...,valuen** (value1 replaces the ${keyword} for the first destination number during the submission e.t.c).

The following table presents the list of available personalized keywords and their corresponding value parameters:

Keyword	Keyword Values	Description
${name}	d_name=v1,...,vn	The name of the recipient
${vname}	d_vname=v1,...,vn	The name of the recipient (vocative)
${surname}	d_surname=v1,...,vn	The surname of the recipient
${vsurname}	d_vsurname=v1,...,vn	The surname of the recipient (vocative)
${mobile}	d_mobile=v1,...,vn	The mobile of the recipient
${birthday}	d_birthday=v1,...,vn	The birthday of the recipient
${nameday}	d_nameday=v1,...,vn	The name of the recipient
${custom0} ... ${custom4}	d_custom0=v1,...,vn ... d_custom4=v1,...,vn	Five general purposed custom fields

Table 2: Special personalized keywords and their value parameters

The request format for a bulk personalized SMS submission (with request for delivery report) is:

https://tadoo.aws.mcore.cloud//smsp-in

with parameters:

originator=value1

text=Text with ${keyword}

request_delivery=true

d_keyword=value1,value2,...,valuen

mobile_number=number1,number2,...,numbern

Example :

We want to send two text SMS to George (306944444444) and John (306944444445) containing the phrase HELLO "and their name" with originator MYCOMPANY.
The GET HTTP request needed is:

https://tadoo.aws.mcore.cloud//smsp-in?originator=MYCOMPANY&text=HELLO+${name}&request_delivery=true
&d_name=George,John&mobile_number=306944444444,306944444445

George will receive the SMS "HELLO George", while John will receive the SMS "HELLO John"

Please note that in personalized SMS submissions the maximum permitted characters (up to 160 or up to 459) and the total cost are calculated after the keywords replacement.

Step 3: Parse The Response

Upon the receipt of a Short Message submission request, the SMS Platform parses the submitted parameters, performs the required checks and sends back a response. More detailed information is provided in the following sections.

Single SMS Response

The format of SMS Platform response in case of a single Short Message submission is the following:

acceptance status|message id|balance

The *acceptance status* reports if the submission has been accepted or rejected from the SMS Platform. The acceptance status can have the following possible values:

Value	Description
1	The message is accepted
0	The message is rejected
-1	The username is empty
-2	The password is empty
-3	The user does not exist.
-4	The user has been deactivated
-5	The destination mobile number is wrong
-6	The country code is wrong
-7	The message originator has wrong format
-8	More than 100 destination numbers (for bulk submissions with GET Http requests)
-9	The text parameter is missing
-10	The value of the keyword is missing
-11	The text parameter contains invalid keywords
-12	The d_nameday or the d_birthday parameter has wrong format
-13	The d_mobile parameter has wrong format
-15	The text contains invalid characters
-16	Some of the submitted data are wrong
-17	Internal server error
-18	The text is too long

Table 3: SMS Platform HTTP interface error codes

The *message id* is a unique identifier for the submitted SMS and the *balance* is the number of credits the account holds.

Example :

Supposing a customer has submitted a single message, the SMS Platform has accepted that message, has assigned to it the id 97533f46-84c6-49f3-802e-470221cb3db7 and the balance of the customer is 25.
The response of the SMS Platform upon the submission will be:

1|97533f46-84c6-49f3-802e-470221cb3db7|25

Supposing the SMS Platform has rejected the message because of wrong mobile number format (for that reason has not assigned any id to the message).
The response of the SMS Platform upon the submission will be:

-2|0|25

Bulk SMS Response

The format of SMS Platform response in case of a bulk Short Message submission is the following:

acceptance status 1|message id 1|acceptance status 2|message id 2

The *acceptance status* reports if the submission has been accepted or rejected from the SMS Platform (see Table 3) while the *message id* is a unique identifier for the submitted SMS.

Example :

Supposing a customer has submitted two messages (as a bulk submission), the SMS Platform has accepted the first message and has assigned to it the id 97533f46-84c6-49f3-802e-470221cb3db7. Furthermore, the SMS Platform has rejected the second message because of wrong mobile number format (for that reason has not assigned any id to the message).

The response of the SMS Platform upon the bulk submission will be:

1|97533f46-84c6-49f3-802e-470221cb3db7|-2|0

Step 4: Manage The Delivery Reports (DLR)

SMS Platform HTTP interface supports DLR status information in two different ways (pull and push mode). Both modes require to set the parameter **request_delivery= true** during the HTTP request.

Delivery Reports (Pull Mode)

If you set the delivery_push parameter equals to "false", then you will have to use the get_status or get_status2 parameters in order to receive the altered delivery reports (pull mode). More detailed information is provided in the following sections.

Simple Delivery Reports

Using the parameter **get_status** one can retrieve the list of all SMS sent and their status, if their status has been changed since the last time the of this request type.

The status is returned in pairs **message_id | message_status**

message_id: the message id returned on submission of the SMS
message_status: the current message status (s sent, f failed, d delivered)

Example:

Suppose that the submission of an SMS has returned the message id *97533f46-84c6-49f3-802e-470221cb3db7*. The following request will display the status of this SMS if it has been changed. It will also return all the messages that have altered status. The request is:

https://tadoo.aws.mcore.cloud//smsp-in?get_status=true

The response will be:

97533f46-84c6-49f3-802e-470221cb3db7|d|351ce2f0-953c-4a4f-905b-7d3ae4891a80|f

which means that the SMS with id 97533f46-84c6-49f3-802e-470221cb3db7 is delivered and the one with id 351ce2f0-953c-4a4f-905b-7d3ae4891a80 is failed.

Advanced Delivery Reports

Using the parameter **get_status2** one can retrieve in addition the sub status (more specific than status) of a specific sent.

The status and sub status is returned in pairs **message_id | message_status | message_sub_status:**

message_id: the message id returned on submission of the SMS
message_status: the current message status (s sent, f failed, d delivered)
message_sub_status: the current message sub status (d delivered, e expired, x deleted, u undeliverable, a accepted, r rejected, n unknown) (see Table 4)

Example:

Suppose that the submission of an SMS has returned the message id *97533f46-84c6-49f3-802e-470221cb3db7*. The following request will display the status of this SMS if it has been changed. It will also return all the messages that have altered status. The request is:

https://tadoo.aws.mcore.cloud//smsp-in?get_status2=true

The response will be:

97533f46-84c6-49f3-802e-470221cb3db7**|d|d|**351ce2f0-953c-4a4f-905b-7d3ae4891a80**|f|e**

which means that the SMS with id 97533f46-84c6-49f3-802e-470221cb3db7is delivered and the one with id 351ce2f0-953c-4a4f-905b-7d3ae4891a80is failed because of the validity period expiration.

Delivery Reports and SMS Cost

Both simple and advanced delivery reports can be combined with the **get_cost** parameter. If the user has set the get_cost parameter equals to "true" he will receive a response in the following format:

message_id|message_status|message_cost
(simple delivery reports response)

message_id|message_status|message_sub_status|message_cost
(advanced delivery reports response)

message_id: the message id returned on submission of the SMS
message_status: the current message status (s sent, f failed, d delivered)
message_sub_status: the current message sub status (d delivered, e expired, x deleted, u undeliverable, a accepted, r rejected, n unknown) (see Table 4)
message_cost: the credits that the SMS cost

Example:

Suppose that the submission of an SMS has returned the message id 12100. The following request will display the status of this SMS if it has been changed. It will also return all the messages that have altered status. The request is:

https://tadoo.aws.mcore.cloud//smsp-in?get_status2=true &get_cost=true

The response will be:

12100|d|d|2|12102|f|e|1

which means that the SMS with id 12100 is delivered and cost 2 credits and the one with id 12102 is failed because of the validity period expiration and cost 1 credit.

Delivery Reports (Push Mode)

Upon the receipt of a delivery report, the SMS Platform will forward it to the specified customer's server. Towards this, an HTTP request of the following format will be generated and forwarded to the customer's server:

customer_url?msgid=msg_id&status=msg_status&sstatus=msg_sstatus &smscost=msg_smscost

where customer_url is defined in the user's settings and targets the script that implements the customer's logic for delivery reports handling

An overview of the parameters contained in a delivery report HTTP Request is shown in the following table. Please note that all parameters are mandatory and will always be included in the HTTP Request.

Parameter	Type	Possible Values/Comment
msgid	string	The message id returned on submission of the SMS
status	string	The current message status. The possible values are: ● **s – sent:** the message has been submitted to the provider ● **f – failed:** the message failed to be delivered the message has been delivered to the final destination ● **d – delivered:** the message has been delivered to the final destination
sstatus	string	The current message sub status. The possible values are: ● **s – sent::** the message has been submitted to the ● provider ● **d - delivered:** the message has been delivered to the final destination ● **f – failed:** the message was marked as failed by the ● provider ● **e - expired:** the message was stored by the provider until the expiration date of the message was passed ● **x - deleted:** the message has been deleted manually by the provider. This is not supposed to happen, unless the operator detects a large amount of spam messages ● **u – undeliverable:** the message could not be delivered because it does not exist anymore, or because the operator could not find an appropriate route to this user ● **a – accepted:** the message has been accepted to be delivered by the provider ● **r – rejected:** the message has been rejected because of syntactic or semantic problems with the message parameters. ● **o – billing error:** the message failed to get billed by ● the platform ● **m – no price set::** the message failed because there ● is not any price set in the user's pricelist ● **y – billing failure:** the sms failed. Please

		contact • your administrator with error code #BSPFTB • **j – not enough balance:** The user does not have • enough balance. • **l – curfew period applied:** The message marked to be sent after the defined curfew period • **z – spam:** The message marked as spam • **i – invalid amount::** the amount to be charged is not valid • **k – insufficient balance:** The subscriber has insufficient balance • **n – unknown:** an unknown error has occurred
smscost	number	The credits that the SMS cost
currency_code	string	The currency of the smscost
retry	number	The number of the current push retry
mcc (Optional)	number	The mobile country code of the destination number. This information will be pushed only if it is available.
mnc (Optional)	number	The mobile network code of the destination number. This information will be pushed only if it is available.
external_id (Optional)	number	The id the sms provider assigned to the message. This information will be pushed only if it is available.
contact_id (Optional)	number	The id of the contact the message was sent to. This information will be pushed only if it is available.
campaign_id (Optional)	number	The id of the campaign the message belongs to. This information will be pushed only if it is available.
reference (Optional)	number	The id of the group message dispatch the message was part of. This information will be pushed only if it is available.

Table 4: Delivery reports HTTP request parameters

Example :

Suppose the user has defined in his settings the following url:

http://myserver/reports.php

and the SMS Platform has received a delivery report for the message with id:12, which reports that the message failed to be delivered because the user has his handset inactivated and it's cost was 1 credit. In the above scenario the SMS Platform will send the following HTTP request:

http://myserver/reports.php?msgid=12&status=f&sstatus=e&smscost=1

The customer's script must reply with a successful HTTP Status code (e.g. 200 – OK). Otherwise, the server will continue to send the message, assuming that the initial attempt was unsuccessful.

Step 5: Get Your Balance Information

SMS Platform enables you to poll our server to receive information about your current balance using the get_balance *command*. Upon the receipt of a get_balance command SMS Platform will return the number of credits available on this particular account. In order to do this a URL in the following format must be generated:

Example:

https://tadoo.aws.mcore.cloud//smsp-in?get_balance=true

The response will be:

12

which means that you have 12 credits remaining in your account.

Appendix A: List of Parameters And Commands

Table below contains the list of available parameters:

Parameter	Description	Type	Valid Values/ Limitations
originator	Message sender	String	It must be either numeric with a length range between 10 and 16 digits (only for non-greek destinations) or alphanumeric up to 11 latin chars (including the special chars !:;+.-)
mobile_number	Message destination	String	Must be a valid mobile number
text	Message text	String	Max 160 chars (or 459 chars for concatenated messages)
request_delivery	Request delivery reports indicator	Boolean	The boolean values true/false
delivery_push	Push/Pull mode delivery reports indicator	Boolean	The boolean values true for push mode and false for pull mode
concatenated	Long message text indicator	Boolean	The boolean values true for text up to 459 chars and false for text up to 160 chars
flash	Flash message indicator	Boolean	The boolean values true for flash text message up to 160 chars and false for simple text message
get_cost	Request sms cost with delivery report	Boolean	The boolean values true and false
d_name	Values for keyword ${name}	String	Comma separated list of strings
d_vname	Values for keyword ${vname}	String	Comma separated list of strings
d_surname	Values for keyword ${surname}	String	Comma separated list of strings
d_vsurname	Values for keyword ${vsurname}	String	Comma separated list of strings
d_mobile	Values for keyword ${mobile}	String	Comma separated list of numeric strings
d_nameday	Values for keyword ${nameday}	String	Comma separated list of strings. Only digits and the special characters -/ are allowed.
d_birthday	Values for keyword ${birthday}	String	Comma separated list of strings. Only digits and the special characters -/ are allowed.
d_custom0 ... d_custom4	Values for keywords ${custom0} … ${custom4}	String	Comma separated list of strings.

Command	Description	Type	Valid Values/ Limitations
get_balance	Get balance information	-	Just include the parameter in your request without any value.
get_status	Retrieve delivery reports	-	Only for use with pull delivery reports mode. Just include the parameter in your request without any value.
get_status2	Retrieve detailed delivery reports	-	Only for use with pull delivery reports mode. Just include the parameter in your request without any value.

Table 5: SMS Platform HTTP API request parameters and commands

Parameter originator (Mandatory)

The originator of the message. Should be either numeric with a length range between 10 and 16 digits (only for non-Greek destinations) or alphanumeric up to 11 latin chars, including the special characters `!:;+.-`

Examples:

originator=MyCompany, originator=My-Company

Parameter mobile_number (Mandatory)

The "mobile_number" parameter is used to specify the destination(s) GSM number of the Short Message. The destination number must not be longer than 20 characters and it must be written in international format (**without the leading + or 00**). In case of bulk Short Message submissions the value of the "mobile_number" must be a comma separated list of mobile numbers.

Examples:

mobile_number=306909876543

(single message submission)

mobile_number=306909876543,306909876542,306909876541

(bulk message submission)

Parameter text (Mandatory)

The Short Message text. It must not exceed the length of 160 characters, in case of a non concatenated message, or 459 characters, in case of a concatenated message. In the second case the submitted message appears as one long Short Message at the recipient's handset. The allowed characters for these types can be found in Appendix B. SMS Platform expects the value of text parameter in UTF-8 encoding. Of course, since it's on URL it must be URL-escaped, i.e. special characters like the ones shown in the table above must be transformed according to URL-encoding.

Parameter request_delivery (Optional)

If this parameter is used with "true" value, then delivery reports are requested for the corresponding Short Message. The default value of the parameter "request_delivery" is "false".

Parameter delivery_push (Optional)

The "delivery_push" parameter is used to specify the way in which the requested delivery reports will be received by the customer's server. If this parameter is "true" (push mode), the SMS Platform will forward every received report to the customer's server. If it's value is "false" (pull mode), the customer receives the new delivery reports by using the get_status or get_status2 parameter.

Parameter concatenated (Optional)

If this parameter is used with "true" value, the message text length can be up to 459 chars otherwise can be up to 160 chars. If the text length exceeds the 160 (or 459) chars, only the first 160 (or 459) will be send. By default a Short Message is considered as non concatenated, if "concatenated" parameter is not specified.

Parameter flash (Optional)

If this parameter is used with "true" value, the message will be send as a flash message. In case of a flash message the message text has to be up to 160 chars (the "concatenated" parameter is ignored). If the text length exceeds the 160 chars, only the first 160 will be send. By default a Short Message is considered as non flash, if "flash" parameter is not specified.

Parameter utf (Optional)

If this parameter is used with "true" value, the message will be send as a Unicode message. In case of a Unicode message each message has to be up to 70 chars, but instead of the flash messages, you can combine the "utf" parameter with the "concatenated" parameter in order to send long Unicode messages. By default a Short Message is considered as non Unicode, if "utf" parameter is not specified.

Parameter get_cost (Optional)

This parameter is used in combination with the commands get_status or get_status2. When this parameter is included in the HTTP request with a "true" value , the SMS Platform returns a list of the delivery reports of the messages the user has sent including the credits that every sms cost. *Detailed information about the delivery reports is provided in the section [Delivery Reports (DLR)].*

Parameter d_name (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${name}. The value of this parameter is a comma separated list of strings that will replace the ${name} keyword during the submission (the first string will replace the ${name} keyword for the first destination e.t.c).

Parameter d_vname (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${vname}. The value of this parameter is a comma separated list of strings that will replace the ${vname} keyword during the submission (the first string will replace the ${vname} keyword for the first destination e.t.c).

Parameter d_surname (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${surname}. The value of this parameter is a comma separated list of strings that will replace the ${surname} keyword during the submission (the first string will replace the ${surname} keyword for the first destination e.t.c).

Parameter d_vsurname (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${vsurname}. The value of this parameter is a comma separated list of strings that will replace the ${vsurname} keyword during the submission (the first string will replace the ${vsurname} keyword for the first destination e.t.c).

Parameter d_mobile (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${mobile}. The value of this parameter is a comma separated list of "numeric" strings that will replace the ${mobile} keyword during the submission (the first string will replace the ${mobile} keyword for the first destination e.t.c).

Parameter d_nameday (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${nameday}. The value of this parameter is a comma separated list of strings (that contain only digits and the characters -/), that will replace the ${nameday} keyword during the submission (the first string will replace the ${nameday} keyword for the first destination e.t.c).

Parameter d_birthday (Optional)

This parameter is used for persinalized Short Message submissions and only when the message text contains the keyword ${birthday}. The value of this parameter is a comma separated list of strings (that contain only digits and the characters -/), that will replace the ${birthday} keyword during the submission (the first string will replace the ${birthday} keyword for the first destination e.t.c).

Parameters d_custom (Optional)*

This parameters are used for persinalized Short Message submissions and only when the message text contains the keywords ${custom0} - ${custom4}. The values of these parameters is comma separated lists of strings that will replace the ${custom0} - ${custom4} keywords during the submission (the first string in d_custom0 will replace the ${cutom0} keyword for the first destination e.t.c).

Command get_balance (Optional)

If this parameter is included in the request, the SMS Platform returns the current balance of the user and then stops the processing of that request. *Detailed information about the user balance is provided in the section [User Account Balance].*

Example:

https://tadoo.aws.mcore.cloud//smsp-in?get_balance=true

Command get_status (Optional)

This parameter is useful only when the **request_delivery=true** and **delivery_push=false**. When this parameter is included in the HTTP request, the SMS Platform returns a list of the delivery reports of the messages the user has sent. *Detailed information about the delivery reports is provided in the section [Delivery Reports (DLR)].*

Example:

https://tadoo.aws.mcore.cloud//smsp-in?get_status=true

Command get_status2 (Optional)

This parameter is useful only when the **request_delivery=true** and **delivery_push=false**. When this parameter is included in the request url, the SMS Platform returns a list with detailed delivery reports of the messages the user has sent. *Detailed information about the delivery reports is provided in the section [Delivery Reports (DLR)].*

Example:

https://tadoo.aws.mcore.cloud//smsp-in?get_status2=true

Appendix B: Allowed characters in "text" field

The following table shows the characters allowed in the "message" parameter of a Short Message submission request. Please note that Greek small letters are automatically transformed into the corresponding capital ones since a normal text Short Message doesn't support Greek small letters.

ISO Character Set 8859-7 (Greek)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80																
90																
A0	NBSP 00A0	‘ 02BD	’ 02BC	£ 00A3			¦ 00A6	§ 00A7	¨ 00A8	© 00A9		« 00AB	¬ 00AC	- 00AD		― 2015
B0	° 00B0	± 00B1	² 00B2	³ 00B3	΄ 0384	΅ 0385	Ά 0386	· 00B7	Έ 0388	Ή 0389	Ί 038A	» 00BB	Ό 038C	½ 00BD	Ύ 038E	Ώ 038F
C0	ΐ 0390	Α 0391	Β 0392	Γ 0393	Δ 0394	Ε 0395	Ζ 0396	Η 0397	Θ 0398	Ι 0399	Κ 039A	Λ 039B	Μ 039C	Ν 039D	Ξ 039E	Ο 039F
D0	Π 03A0	Ρ 03A1		Σ 03A3	Τ 03A4	Υ 03A5	Φ 03A6	Χ 03A7	Ψ 03A8	Ω 03A9	Ϊ 03AA	Ϋ 03AB	ά 03AC	έ 03AD	ή 03AE	ί 03AF
E0	ΰ 03B0	α 03B1	β 03B2	γ 03B3	δ 03B4	ε 03B5	ζ 03B6	η 03B7	θ 03B8	ι 03B9	κ 03BA	λ 03BB	μ 03BC	ν 03BD	ξ 03BE	ο 03BF
F0	π 03C0	ρ 03C1	ς 03C2	σ 03C3	τ 03C4	υ 03C5	φ 03C6	χ 03C7	ψ 03C8	ω 03C9	ϊ 03CA	ϋ 03CB	ό 03CC	ύ 03CD	ώ 03CE	

Allowed characters Characters converted to corresponding capitals

Appendix C: Example - Java/JSP HTTP Client

Below you can find an example of a Java / JSP implementation of a form that submits the message parameters to a Java Web Servlet which in turn performs the HTTP request to SMS Platform API. The servlet code is using Apache HTTPClient 4.5.2 to perform the HTTP request.

Form HTML/JSP code: Submits the message parameters to a servlet

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head><title>SMS Form</title></head>
<body>
<form action="${pageContext.request.contextPath}/SmsForm" method="POST">
    <c:if test="${requestScope.result ne null && !empty requestScope.result}">
        <div id="result" style="color:${requestScope.resultColor}">${requestScope.result}</div>
    </c:if>
    <div>
        <label style="display:inline-block; width: 50px;" for="from">From :</label>
        <input type="text" id="from" name="from" value=""/>
    </div>
    <div>
        <label style="display:inline-block; width: 50px;" for="to">To :</label>
        <input type="text" id="to" name="to" value=""/>
    </div>
    <div>
        <label style="display:inline-block; width: 50px;" for="text">Text :</label>
        <textarea cols="20" rows="3" name="body" id="text"></textarea>
    </div>
    <input type="submit" value='Send'/>
</form>
</body>
</html>
```

Servlet Code: Receives the form parameters and performs a separate HTTP request to the API:

```
package tadoo.aws.mcore.cloud.smsform;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.client.methods.RequestBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

@WebServlet(name = "SmsForm", value = "/SmsForm")
public class SmsFormServlet extends HttpServlet {
    private static final Logger logger = LoggerFactory.getLogger(SmsFormServlet.class);
    //your credentials
    private static final String API_KEY = "<YOUR API KEY>";
    private static final String AUTHORIZATION_VALUE = "Bearer " + API_KEY;

    //the provided values for the platform
    //e.g. for PLATFORM_URL: http://example.com:1111/smsp-in
    //the host is example.com, the port is 1111 and the platform url path is smsp-in
    private static final String PLATFORM_HOST = "<PLATFORM HOST";
    private static final int PLATFORM_PORT = <PLATFORM PORT>;
    private static final String PLATFORM_URL =
"https://"+PLATFORM_HOST+":"+PLATFORM_PORT+"/<PLATFORM URL PATH>";

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        forwardToView(request, response);
    }
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        doRequest(request, response);
    }

    public void doRequest(HttpServletRequest servletRequest, HttpServletResponse servletResponse) throws
ServletException, IOException {
        final String originator = servletRequest.getParameter("from");
        final String destination = servletRequest.getParameter("to");
        final String text = servletRequest.getParameter("body");
        logger.info("sending sms with params: from:{} to:{} text:{}", originator, destination, text);
        //credentials provider automatically url-encodes the credentials
        try (CloseableHttpClient client = HttpClients.createDefault()) {
            HttpUriRequest post = RequestBuilder.post(PLATFORM_URL)
```

```
                //this is required in order to convert any non-latin characters correctly
                .addHeader(HttpHeaders.AUTHORIZATION, AUTHORIZATION_VALUE)
                .setCharset(StandardCharsets.UTF_8)
                //all parameters are automatically url-encoded
                .addParameter("originator", originator)
                .addParameter("mobile_number", destination)
                .addParameter("text", text)
                //for receiving dlrs
                .addParameter("request_delivery", Boolean.TRUE.toString())
                //for sending SMS with unicode support (extended character set)
                .addParameter("utf", Boolean.TRUE.toString())
                .build();
        try(CloseableHttpResponse response = client.execute(post)) {
            if(response.getStatusLine().getStatusCode() != 200) {
                smsError(servletRequest);
            } else {
                String receivedResponse = response.getEntity() != null ?
                        EntityUtils.toString(response.getEntity()) : "";
                String[] parts = receivedResponse.split("\\|");

                if(parts.length < 2 || !"1".equals(parts[0])) {
                    smsError(servletRequest);
                } else {
                    String id = parts[1];
                    smsSuccess(servletRequest, id);
                }
            }
        }
    } catch (Exception e) {
        logger.error("Exception while submitting sms", e);
        smsError(servletRequest);
    }
    forwardToView(servletRequest, servletResponse);
  }
  private void smsSuccess(HttpServletRequest servletRequest, String submittedSmsId) {
    servletRequest.setAttribute("resultColor", "darkgreen");
    servletRequest.setAttribute("result", "sms submitted successfully, with ID:"+submittedSmsId);
  }
  private void smsError(HttpServletRequest servletRequest) {
    servletRequest.setAttribute("resultColor", "red");
    servletRequest.setAttribute("result", "failure submitting sms");
  }
  private void forwardToView(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.getRequestDispatcher("smsform.jsp").forward(request, response);
  }
}
```

Appendix D: Example - .NET/ASP HTTP Client

Below you can find an example of a .NET/ASP implementation of a form that submits the message parameters to a server, which in turn performs the HTTP request to SMS Platform API.

Form HTML/ASP code: Submits the message parameters to the server**:**

*sampleForm.aspx**:***

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="sampleForm.aspx.cs"
Inherits="SMSPlatformHTTPAPI.sampleForm" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Simple .Net Page</title>
   <style>
      label.form {
         display: inline-block;
         width: 150px;
      }
      .form_response {
         margin-top: 20px;
      }
   </style>
</head>
<body>
   <div id="single">
      <form id="form1" runat="server">
         <div>
            <asp:Label ID="lblFrom" AssociatedControlID="txtFrom" CssClass="form"
runat="server">From:</asp:Label>
            <asp:TextBox ID="txtFrom" ClientIDMode="Static" runat="server" />
         </div>
         <div>
            <asp:Label ID="lblTo" AssociatedControlID="txtTo" CssClass="form"
runat="server">To:</asp:Label>
            <asp:TextBox ID="txtTo" ClientIDMode="Static" runat="server" />
         </div>
         <div id="text_div">
            <asp:Label ID="lblText" AssociatedControlID="txtText" CssClass="form"
runat="server">Text:</asp:Label>
            <asp:TextBox ID="txtText" ClientIDMode="Static" TextMode="MultiLine" Columns="20" Rows="3"
runat="server" />
         </div>
         <div>
            <asp:Button ID="btnSubmit" Text="Submit" runat="server" OnClick="btnSubmit_Click" />
         </div>
         <div class="form_response">
            <asp:Literal ID="ltlAPIResponse" Text="<span>{0}</span>" runat="server" />
         </div>
      </form>
   </div>
</body>
</html>
```

Server Code: Receives the form parameters and performs a separate HTTP request to the API:

sampleForm.aspx.cs:

```
using System;
using System.IO;
using System.Net;

namespace SMSPlatformHTTPAPI{
   public partial class sampleForm : System.Web.UI.Page{
      private string To { get; set; }
      private string From { get; set; }
      private string Text { get; set; }
      private const string API_KEY = "<YOUR_API_KEY>";
      private const string AUTHORIZATON_VALUE = "Bearer " + API_KEY;
      private const string Host = "<YOUR PLATFORM'S HTTP API URL>";
      private const string ParameterUrl = "?
mobile_number={0}&originator={1}&text={2}&request_delivery=true&utf={3}";
      private const string Port = "<YOUR PLATFORM'S HTTP API PORT>";
      private const string Path = "<YOUR PLATFORM'S HTTP API PATH";
      protected void Page_Load(object sender, EventArgs e){}
      protected void btnSubmit_Click(object sender, EventArgs e) {
         To = Request.Form[txtTo.UniqueID];
         From = Request.Form[txtFrom.UniqueID];
         Text = Request.Form[txtText.UniqueID];
         //for unicode sms change this to true
         string Utf = false;
         string Parameters = String.Format(ParameterUrl, To, From, Text, Utf);
         string Url = String.Format("http://{0}:{1}{2}{3}", Host, Port, Path, Parameters);
         var httpReq = (HttpWebRequest)WebRequest.Create(Url);
         httpReq.Method = "GET";
         httpReq.UseDefaultCredentials = false;
         httpReq.Headers["Authorization"] = AUTHORIZATION_VALUE;

         HttpWebResponse httpRes = (HttpWebResponse)httpReq.GetResponse();
         string Response = "<strong>TimeStamp:</strong> " + DateTime.Now.ToString("dd/MM/yyyy
HH:mm:ss") + "<br />";
         Response += "<strong>Status Code:</strong> " + httpRes.StatusCode + "<br />";
         Response += "<strong>Server:</strong> " + httpRes.StatusCode + "<br />";

         Stream stream = httpRes.GetResponseStream();
         StreamReader reader = new StreamReader(stream);

         string Answer = reader.ReadToEnd();
         string messageStatus = Answer.Split('|')[0];
         string messageId = Answer.Split('|')[1];
         string balance = Answer.Split('|')[2];

         Response += "<strong>Message Status:</strong> " + messageStatus + "<br />";
         Response += "<strong>Message Id:</strong> " + messageId + "<br />";
         Response += "<strong>Remaining Balance:</strong> " + balance + "";

         ltlAPIResponse.Text = String.Format(ltlAPIResponse.Text, Response);
      }
   }
}
```