sendQuick Appliance API Guide

Version 3.1

TalariaX Pte Ltd

76 Playfair Road

#08-01 LHK2 Building

Singapore 367996

Tel : +65 6280 2881 Fax : +65 6280 6882

Email : info@talariax.com

www.TalariaX.com

REVISION SHEET

Release No.	Date	Description
3.1	27/07/2020	Re-released version with updated URL and emphasis on Appliance API only.

Table of Contents

sendQuick Appliance API Guide

1.0 Introduction

1.1 About TalariaX Pte Ltd

TalariaX™ develops and offers **enterprise mobile messaging solutions** to facilitate and improve business workflow and communication, and is widely used in areas such as IT alerts & notifications, secure remote access via 2-Factor Authentication, emergency & broadcast messaging, business process automation and system availability monitoring.

In addition to functionality, TalariaX's messaging solutions have also been developed with other key features in mind. These include **security** and **confidentiality** of company information, and **ease in mitigating disruption** during unplanned system downtime such as that arising from cyberattacks.

1.2 About sendQuick

sendQuick is a comprehensive Short Messaging Service (SMS) gateway that is available in the form of an **appliance** or as a **cloud-based** solution. ***sendQuick*** is used by more than 1,500 businesses, including many Fortune Global 500 companies, in 40 countries and across industries such as banking, finance, insurance, manufacturing, retail, government, education, and healthcare.

1.3 Purpose of Document

This document provides an overview of how the sendQuick messaging system interfaces and works with clients' **application servers (AppServer).**

Six methods for sending and receiving SMSes using sendQuick are described, and examples are shown where relevant and applicable. The six methods are

- Hypertext Transfer Protocol — HTTP Post Method
- eXtensible Markup Language — XML Method
- Simple Object Access Protocol — SOAP Method
- JavaScript Object Notation — JSON Method
- Simple Mail Tranfer Protocol — SMTP(Email) Method
- File Transfer Protocol — FTP Method

Other topics of interest, such as use of free line tools, are also included.

2.0 sendQuick Solutions

2.1 Business Uses of SMSes

TalariaX offers a slew of sendQuick solutions to enable businesses to reach out to their customers by tapping into the widely used and cost-effective SMS mode of communication. Broadly categorised, sendQuick solutions help businesses:

i. Broadcast messages such as that used by retailers to announce markdowns, offers and coupons;
ii. Send alerts to target audience, for example alerts on potential malwares or system downtime;
iii. Monitor system availability, for example alerting the IT department when any server is down;
iv. Offer added layer of security and confidentiality of information such as that through 2-Factor Authentication (2FA), and
v. Automate business processes to improve responsiveness to customers, example confirmation of client appointments.

2.2 AppServer and sendQuick Messaging Flow

The general flow of how sendQuick supports messaging is shown in Figure 1. Essentially, to send SMSes:

i. The sender of messages sends the intended outgoing data and instructions, using any of the six ascribed methods, from its AppServer to sendQuick. The recipient for this step can either be sendQuick box or sendQuick cloud, depending on the method subscribed by the sender;
ii. sendQuick directs the outgoing messages to the relevant network carriers;
iii. The network carriers route the messages to users' mobiles.

The flow for incoming messages works the same, only in reverse order.

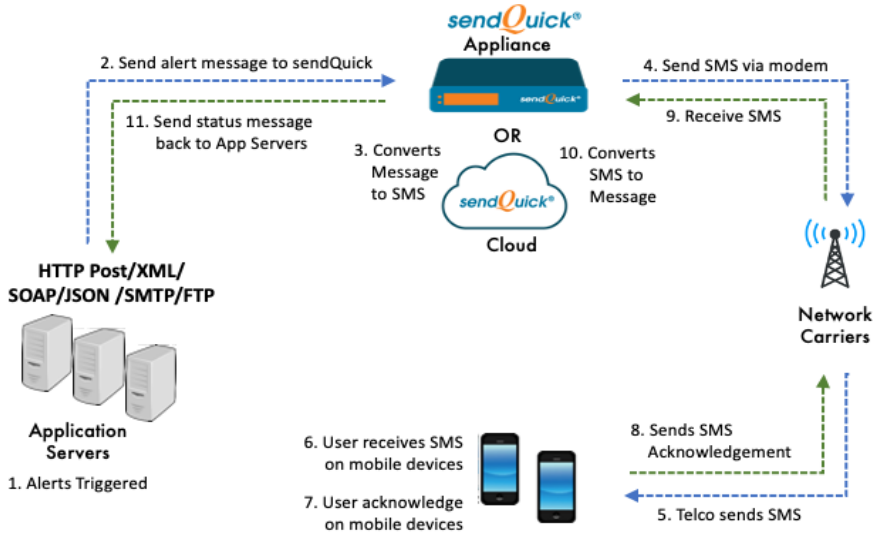

Figure 1: Overview of Transaction Flow

3.0 HTTP Post Method

3.1 Overall Transaction Flow

SMSes are essentially messages containing texts. This being the case, standard HTTP (POST/ GET) is sufficient for transferring data between sendQuick and AppServer. Figure 2 shows how sendQuick connects to any application server, using HTTP/ CGI.

Figure 2: Overall Transaction Flow

Several points to note when using HTTP:

i. Web service for the AppServer needs to be enabled, to accept HTTP (POST/GET) submitted from sendQuick;
ii. The URL path submission of sendQuick requests to AppServer has to be configured, and this can be either a global or per keyword configuration;
iii. For sending SMSes, AppServer needs to initiate a HTTP (POST/GET) to the HTTP API of the sendQuick system.

3.2 Sending SMSes

The sendQuick HTTP API URL path is

- http://<sendquickIPaddress>**/cmd/system/api/sendsms.cgi**

  Replace <sendquickIPaddress> with the IP address assigned to sendQuick device.
  E.g. http://192.168.1.101/cmd/system/api/sendsms.cgi

Table 1 shows the input parameters for sending SMSes.

Parameter Name	Data Type	Max Length	Description	Example of Values
tar_num (Mandatory)	String	220	This is the parameter for target or recipient mobile number. For international format, the '+'# character must be used. To send multiple numbers, use ',' to separate the numbers.	The number has to be numeric and allow '+' special character. **Use %2B in URL in order for it to be URL-encoded to "+"** Example: %2B6591234567 For multiple numbers: %2B6591234561, %2B6591234562
tar_msg (Mandatory)	String	sendQuick can support long messages. Refer to sendQuick configuration	This parameter is the content or body of the message to be conveyed to the recipient. To ensure that recipients receive the entire message or text, this parameter must be encoded into the URL encoded string. Failure to do this may result in partial or corrupted messages delivered to recipients.	Alphabets, numbers and special characters are allowed, and the message should be html-encoded. Message value should not be empty. Example 1: This is a test message for ASCII. Example 2: 안녕하세요 수 있습니다for UTF8.
tar_mode	String	Fixed	This is the processing mode, and can be any of these: **Value** / **Type of messages** text / ASCII text SMS flashtext / flash SMS for ASCII characters utf / UTF8 messages (for non- ASCII characters)	This parameter defines how the message should be processed. For ASCII (English Text), the mode should be set as "text". For UTF (Chinese, Japanese, Korean, etc), the mode should be set as "utf". Flash SMS can support ASCII characters. Set the mode as "flashtext" for flash SMS. The parameter value is case-sensitive.

Parameter Name	Data Type	Max Length	Description	Example of Values
trackid* (optional)	String	50	This is an optional field to track the identity of each individual request.	Trackid value should be an integer. Example: 1001
status_url* (optional)	String	100	The message status URL that the system will respond to when the message has completed processing. This field is optional.	The full URL (http://<ip>/<path>) needs to be included, in order to receive back the status of the post. Example: http://192.168.1.10/webroot/mymessage-status.php
label** (optional)	String	12	This parameter is used to send messages via specific modems. To use this parameter, the label must be associated with a specific modem.	The value can be alphanumeric. Example: marketingmodem
priority (optional)	String	1	This is an advisory parameter for setting the priority of the outgoing SMS.	The value should be an integer between 1 to 9, with the highest priority being 1 and the lowest being 9. Default value is '5'. Example: 1
clientid (optional)	String	20	This is an optional field, and is triggered if there are more than one application running on the same server.	The value can be alphanumeric. Example: client123
username (optional)	String	20	Required for authentication with username/password	The value can be alphanumeric. Example: username123
password (optional)	String	20	Required for authentication with username/password	The value can be alphanumeric. Example: password123
token (optional)	String	20	Required for authentication with token	The value can be alphanumeric. Example: token123

Table 1: Input Parameters for sendQuick API

* The usage of trackid and status_url is discussed more in Section 9.2
** Refer to Section 9.3 for discussion on label.
# Standard HTTP (POST/GET) will treat '+' as a blank space.

1. Example for Single Mobile

http://192.168.1.101/cmd/system/api/sendsms.cgi?tar_num=%2B6591234567&tar_msg=test&tar_mode=text

2. Example for Multiple Mobile

http://192.168.1.101/cmd/system/api/sendsms.cgi?tar_num=%2B6591234561,%2B6591234562,%2B6591234563&tar_msg=test&tar_mode=text

Note : for tar_num, remember to use %2B to represent the “+” character

3.2.1 sendQuick Response to sendsms.cgi

The parameters in Table 2 show the response from sendQuick on the processing status and indicate whether messages have been queued for delivery.

Type	Response
For single tar_num	OK Queue: <sendQuick-message-ID>
For multiple tar_num$_n$	OK Queue: <sendQuick-message-ID$_1$>-<tar_num$_1$> <sendQuick-message-ID$_2$>-<tar_num$_2$> ... <sendQuick-message-ID>-<tar_num$_n$>

Table 2: HTTP sendsms.cgi Response

3.2.2 Check Status of Outgoing SMSes

To check the status of outgoing SMSes, use the API URL path
http://*<sendquickIPaddress>***/cmd/system/api/msgstatus.cgi**

The input parameters are shown in the table below.

Parameter Name	Data Type	Max Length	Description	Example of Values
mode	string	fixed	"queue" only	The value should be alphanumeric which is as predefined. Example: queue
msgid	string	50	The <sendQuick-message-ID> returned from sendsms.cgi	The message id returned from the API. Example: M12345

Table 3: Parameters for Status of Outgoing SMSes

Example Checking Status of Outgoing SMSes

1. Example HTTP Request

http://192.168.1.101/cmd/system/api/msgstatus.cgi?mode=queue&msgid=M12345

2. The format of the response will be as follows

<start-process-dtm>*<tab>***<completed-process-dtm>***<tab>***<status>***<tab>***<modem-imei>***<tab>***<smsc>***<tab>***<tar_num>***<tab>***<tar_msg>**

3. Example of HTTP Response

31-10-020 10:57:56 31-10-020 18:33:45 Y 359126030021471 +6596845999 +6591234567 test message

The URL response when no associating *<sendQuick-message-ID>* is found

➔ No result found: <sendQuick-message-ID>

Note: This is the case where the *<sendQuick-message-ID>* is invalid or the message has been deleted from the system.

3.2.3 Getting the Status of SMSes Sent

This is for sendQuick server to call back the client's status_url to update the message status after SMSes have been sent out. The *status_url* should accept the input parameters shown in the table below.

Parameter Name	Data Type	Max Length	Description
trackid	string	50	This is the trackid submitted by the AppServer for sendQuick to track the status of outgoing SMSes.
status	string	1	This is the status of outgoing SMSes, represented as follows: Y – successfully sent F – failed to send D – Message deleted from server R – Received delivery report from recipient mobile phone.
totalsms	integer	2	This will be the integer value of the total SMSes used for sending the complete message. NOTE: Not applicable for status D or R
mno	string	20	The mobile number of the recipient.

Table 4: Parameters for Getting Status of SMSes Sent Out

Example : Getting Status of SMSes Sent Out Using HTTP

Step 1:
http://192.168.1.101/cmd/system/api/sendsms.cgi?tar_num=%2B6591234567&tar_msg=testmessage&tar_mode=text&trackid=1001&status_url=http://yourclientserverip/apipath/api.cgi

Step 2:
SMS is processed in sendQuick server accordingly and sendQuick responses to the status_url with “*Successful*” or “*Failed*” depending on the status of delivery

Step 3:
http://<yourclientserverip>/apipath/api.cgi?trackid=1001&status=Y&totalsms=1&mno=%2B6591234567

3.2.4 Delete Outgoing SMSes via HTTP

To delete outgoing messages, input API URL path

- http://<sendquickIPaddress>**/cmd/system/api/reqdelete.cgi**

  Replace <sendquickIPaddress> with the IP address assigned to sendQuick device.
  E.g. http://192.168.1.101/cmd/system/api/reqdelete.cgi

The input parameters are shown in table below.

Parameter	Data Type	Max Length	Description
trackid	string	50	This is the trackid submitted by the AppServer for sendQuick to track the status of outgoing SMSes.
tar_num	string	20	The mobile number associated with the trackid.

Table 5: Parameters for Deleting Outgoing SMSes

Note: If tar_num is excluded in the parameters, all the message that is associated with the trackid will be deleted from the system.

Example : Deleting Outgoing SMSes Using HTTP
http://192.168.1.101/cmd/system/api/reqdelete.cgi?tar_num=%2B6591234567&trackid=1001

3.3 Receiving SMSes

This section is for incoming SMSes. Incoming SMSes will be posted to the Client API registered with sendQuick.

Table 6 shows the input parameters for receiving SMSes.

Parameter Name	Data Type	Max Length	Description
mno	string	20	The mobile number of the sender.
txt	string	user defined	The content of the sms message.
dtm	string	19	The date and time when the message was received. + is space between date and time
charset	string	fixed	The character set of the content. "*text*" for ASCII message "*utf*" for unicode UTF-8 message

Table 6: Parameters for Receiving SMSes

Example : Receiving SMSes using HTTP

http://192.168.1.101/clientapi_path/clientapi.cgi?mno=%2B6591234567&txt=testmessage&dtm=05/07/2020+17:57:01&charset=text

4.0 XML Method

4.1 Sending Messages

The parameters for sending SMSes are the same as that for the HTTP method, shown in Table 1. The URL for the API is

- http://<sendQuickIPAddress>**/api/sendsms_xml.php**

Example : Sending SMSes to a Single Mobile Using XML

```
<?xml version="1.0"?>
<info>
<tar_num>+6512345678<tar_num>
<tar_msg>Test Message</tar_msg>
<tar_mode>text</tar_mode>
<label>marketingmodem</label>
<priority>3</priority>
</info>
```

Example : Sending SMSes to Multiple Mobiles Using XML

```
<?xml version="1.0"?>
<info>
<tar_num>+6512345671,+6512345672,+6512345673<tar_num>
<tar_msg>Test Message</tar_msg>
<tar_mode>text</tar_mode>
<label>marketingmodem</label>
<priority>3</priority>
</info>
```

4.2 Responses to sendsms_xml.php

The parameters in the table below show the response from sendQuick on the processing status and indicate whether messages have been queued for delivery.

Type	Response
For single tar_num	<mobilenum>:<message-id>
For multiple tar_num$_n$	<mobilenum$_1$>-<message-id$_1$> <mobilenum$_2$>-<message-id$_2$> ... <mobilenum$_n$>:-<message-id$_n$>

Table 8: XML Response Parameters

4.3 Receiving SMSes

The parameters for receiving SMSes are the same as that for the HTTP method, which is shown in Table 6.

Example : Receiving SMSes Using XML Method

```
http://192.168.1.101/clientapi_path/clientapi.cgi
<?xml version="1.0"?>
<mno>+6512345678</mno>
<txt>test message received</txt>
<dtm>05/07/2020 17:57:01</dtm>
<charset>text</charset>
```

5.0 SOAP Method

5.1 Sending SMSes

The API URL for sending SMSes is

- http://<sendQuickIPAddress>**/api/sendsms_soap.php**

Examples of sending messages using SOAP are shown next, based on the same parameters shown in Table 1.

Example : Sending SMSes to a Single Mobile Using SOAP

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:apiwsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<mns:processAPI xmlns:mns="urn:apiwsdl" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<tar_num xsi:type="xsd:string">+6591234567</tar_num>
<tar_msg xsi:type="xsd:string">test soap</tar_msg>
<tar_mode xsi:type="xsd:string">text</tar_mode>
</mns:processAPI>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example : Sending SMSes to Multiple Mobiles Using SOAP

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="urn:apiwsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<mns:processAPI xmlns:mns="urn:apiwsdl" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<tar_num xsi:type="xsd:string">+6591234561,+6591234562,+6591234563</tar_num>
<tar_msg xsi:type="xsd:string">test soap</tar_msg>
<tar_mode xsi:type="xsd:string">text</tar_mode>
</mns:processAPI>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.2 sendQuick Responses to sendsms_soap.php

The table below shows the response from sendQuick on the processing status and indicates whether messages have been queued for delivery.

Type	Response
For single tar_num	<?xml version="1.0" encoding="ISO-8859-1"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV:Body><ns1:processAPIResponse xmlns:ns1="http://tempuri.org"><return xsi:type="xsd:string">+6591234567:10</return></ns1:processAPIResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
For multiple tar_num$_n$	<?xml version="1.0" encoding="ISO-8859-1"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV:Body><ns1:processAPIResponse xmlns:ns1="http://tempuri.org"><return xsi:type="xsd:string">+6591234561:10 +6591234562:11 +6591234563:12</return></ns1:processAPIResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>

Table 9: SOAP Response Parameters

5.3 Receiving SMSes

The parameters for receiving SMSes are the same as that for the HTTP method, which is shown in Table 6.

Example : Receiving SMSes Using SOAP

```
POST /clientapi_path/clientapi.cgi HTTP/1.0
Host: 127.0.0.1
User-Agent: NuSOAP/0.7.3 (1.114)
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-Length: 653

<?xml version="1.0" encoding="ISO-8859-1"?><SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-
ENV:Body><ns5130:processAPI xmlns:ns5130="http://tempuri.org"><mno
xsi:type="xsd:string">+6591234567</mno><txt xsi:type="xsd:string">test
soap</txt><charset xsi:type="xsd:string">text</charset><dtm
xsi:type="xsd:string">05/07/2020 17:57:01</dtm></ns5130:processAPI></SOAP-
ENV:Body></SOAP-ENV:Envelope>
```

6.0 JSON Method

6.1 Sending SMSes

The URL for sending SMSes using JSON is

- http://<sendQuickIPAddress>**/api/sendsms_json.php**

Examples of sending messages using JSON are shown next, based on the same parameters shown in Table 1.

Example : Sending SMSes to a Single Mobile Using JSON

```
{
 "tar_num": "+6591234567"
 "tar_msg": "test json",
 "tar_mode": "text",
}
```

Example : Sending SMSes to Multiple Mobiles Using JSON

```
{
 "tar_num": "+6591234561,+6591234562,+6591234563"
 "tar_msg": "test json",
 "tar_mode": "text",
}
```

6.2 sendQuick Response to sendsms_json.php

The table below shows the response from sendQuick on the processing status and indicates whether messages have been queued for delivery.

Type	Response
For single tar_num	<mobilenum>:<message-id>
For multiple tar_num$_n$	<mobilenum$_1$>-<message-id$_1$> <mobilenum$_2$>-<message-id$_2$> ... <mobilenum$_n$>:-<message-id$_n$>

Table 10: JSON Response Parameters

6.3 Receiving SMSes

The parameters for receiving SMSes are the same as that for the HTTP method, which is shown in Table 6.

Example : Receiving SMSes Using JSON Method

```
http://192.168.1.101/clientapi_path/clientapi.cgi
      {
       "Entry": {
             "mno": "+6591234567"
             "txt": "test",
             "dtm": "05/07/2020 17:57:01",
             "charset": "text",
             }
      }
```

7.0 SMTP (Email) Method

7.1 Sending SMS

The syntax to use for the message when sending e-mail is

- <tar_num>@<serverIP/domainname>

where <tar_num> is the number to send the message to and <serverIP/domainname> being the IP address or domain name of the server.

For example:
If you set the IP address 192.168.1.101 and the number to send the SMS is 91234567, then the e-mail for sending messages is ***91234567@192.168.1.101***

Example

To	:	91234567@192.168.1.101
Subject	:	Test message
Message	:	Test message for you

There is no difference for sending SMS messages via email in different languages. The system will be able to recognise from the email header.

7.2 Receiving SMS

Incoming SMS messages are routed from sendQuick to clients' email based on emails specified by clients.

Example

Sender: +6597654321
Timestamp: 01/07/20 17:13:22
IMEI: 359123456701234
SMSC: +6591234567
Message: Test message for you

8.0 FTP Method

8.1 Activation

For this method, the FTP service in sendQuick needs to be activated first.

To do this, login to the sendQuick web interface as a Server Administrator.

Navigate to ***Messaging Setup > SMS Messaging Setup***
Ensure that the SFTP/FTP to SMS Service is “Enabled”

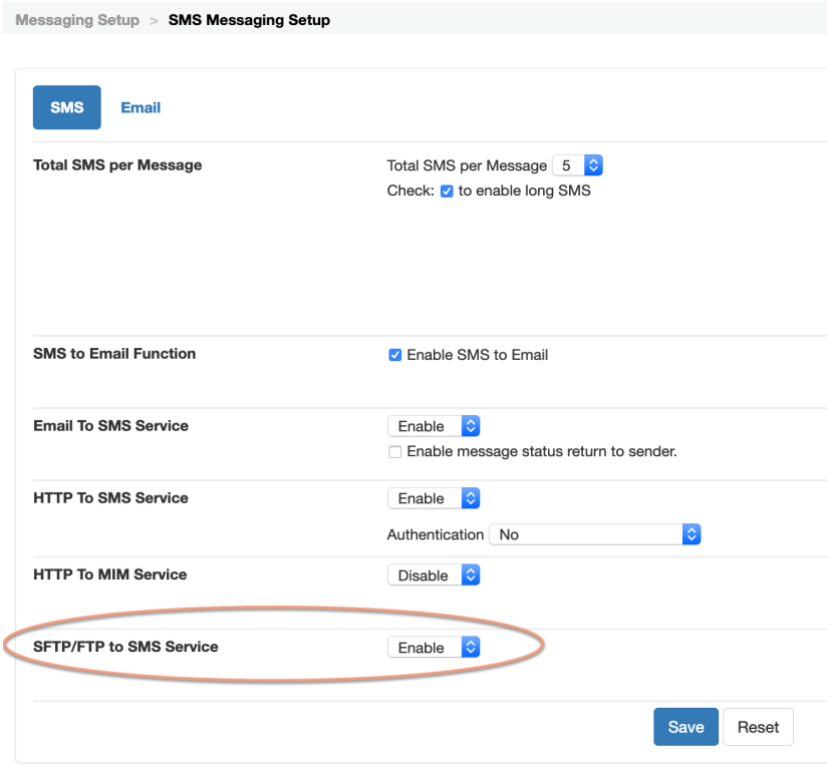

Figure 3.1: Changing FTP Account and Password

Next, configure the FTP account and Password as follows:

- On the sendQuick web interface, navigate to ***Password Management > FTP Login Account***
- Enter your new password and click “***Save***” (see below)

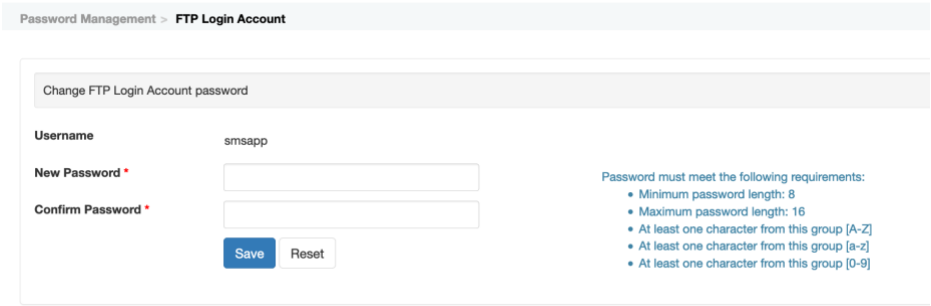

Figure 3.2: Changing FTP Account and Password

8.2 Folder, File & Format

8.2.1 Folder

The FTP folder name in the server is “/smsftp/”
This is the directory where the file with login 'smsapp' username will be stored.

8.2.2 File Structure

The structure of files sent using FTP are as indicated in the table below

Field Size	Remarks
Hand Phonex(15)	Numeric only
Messagex(160)	

Table 7: FTP File Structure

8.2.3 File Format

The format for data within the message file must follow that shown in the example below using comma-delimited (i.e. use comma) to separate values, and double-quoted to encapsulate one piece of data. When the data file is created using Excel spreadsheet, the double-quotes are automatically created.

Example : FTP File Format

Format:

handphone,message
handphone1,message1

Example:

“96367680”,”How are you?”
“96180556”,”I am fine”

To check the formatting of any data prior to sending the file, open the target file for viewing in any text editor, for example Notepad or TextEdit.

8.2.4 File Type

Files allowed are text files with extension “*.msg”*. There are two text files that need to be uploaded when using the FTP method – one being the file containing the message (the ***.msg*** file) and the other being the file to signify completion of upload (the ***.end*** file). The message file must be in csv format and .end is a blank file.

8.2.5 Upload the Files

Upload the ***.msg*** file containing the message into the sendQuick folder :
/home/smsapp/smsftp

Thereafter, upload the ***.end*** file into the same folder, to inform the system that the message file upload has been complete

9.0 Useful Topics

In this section, other useful topics such as preparation of AppServer to communicate with sendQuick and free tools of interest, are covered. Examples showing codes and parameter commands are given where practicable.

9.1 Examples of Applications

A few sample scripts are shown to illustrate how the AppServer has to be prepared to accept input from the sendQuick system. The programming languages used in these examples are PHP or Perl, but the underlying concept is similar for all HTTP scripting or programming platforms.

9.1.1 Sending of SMSes

The following sample code shows a simple HTTP submission in Perl.

Example : Sample Code for Simple HTTP CGI submitted with Perl

```
#!/usr/bin/perl -w
use LWP::UserAgent;
use URI::Escape;

# Create the simple HTTP agent.
$ua = LWP::UserAgent->new;
$ua->agent("MyTestApp/0.1 ");

# Composing the parameters and content.
my $tar_msg = "testing 1 2 3 -- From TalariaX (Singapore)";
my $mno = '91234567';
my $req = HTTP::Request->new(POST =>
'http://192.168.1.101/cmd/system/api/sendsms.cgi');
$req->content_type('application/x-www-form-urlencoded');
$req->content('tar_num=' . uri_escape($mno) .
            '&tar_mode=text' .
'&tar_msg=' . uri_escape($tar_msg);

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
print "Submit successful: ", $res->content, "\n";
} else {
print "Submit failure: ", $res->status_line, "\n";
}
```

NOTE:

- The assumed IP address of sendQuick is 192.168.1.101
- This script uses the LWP module of Perl. Please download the LWP module from http://search.cpan.org in order to test the script.

9.1.2 Global Script for Receiving SMS

To configure the global receiving script, specify the URL path to the sendQuick server, by configuring the **Messaging Setup**.

On the sendQuick web interface, navigate to ***Messaging Setup > SMS Response Action***.
As shown in the example below, assuming the AppServer IP is 192.168.1.22 and path is **/sms/receivesms.php,** fill in the field as follows:

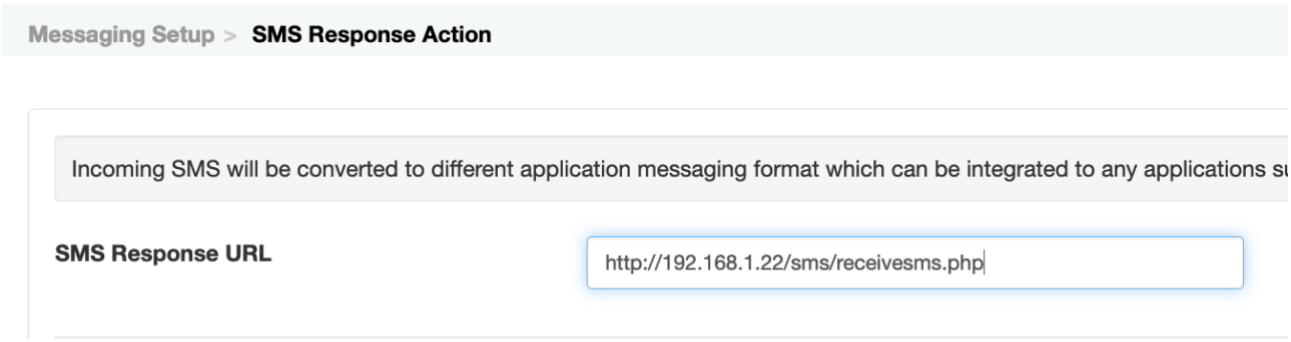

Figure 4: Configuration of Global Receiving Script

Example : Sample Code for Receiving SMSes Using PHP

```
<?php
/* Even though there are total 4 input parameters,
charset and dtm is not applicable for this example,
so we ignore them completely. */
$mno = $_REQUEST['mno'];
$msg = $_REQUEST['txt'];

if( !isset($mno) || !isset($msg) ){
echo "Invalid input - Missing data";

exit;
}

if( strlen($mno) == 0 || strlen($msg) == 0 ){
echo "Invalid input - Blank data.";
exit;
}

try {
/* now we had captured the data, we store it into our database table.
We use sendQuickLite (standard module in PHP 5.1.x and above) to store the
message. To create this table, just download sendQuicklite3 from
http://www.sendQuicklite.org/ use the command line utility to create this
database (messagedb.sendQuicklite) and table: incoming_sms:

CREATE TABLE `incoming_sms` (`idx` INTEGER PRIMARY KEY , `mno` VARCHAR, `txt`
VARCHAR)

NOTE: Highly recommend to use Firefox plugin to manage sendQuickLite
database:
http://sendQuicklitemanager.mozdev.org/
*/
```

```
// NOTE: Path to the messagedb.sendQuicklite required to full path.
$dbh = new PDO('sendQuicklite://<path>/<to>/messagedb.sendQuicklite');

$stmt = $dbh->prepare("INSERT INTO incoming_sms (mno,txt) VALUES
(:mno, :txt)");
$stmt->bindParam(':mno',$mno);
$stmt->bindParam(':txt',$msg);
$stmt->execute();

} catch ( Exception $e ) {
$msg = $e->getMessage();
echo "DB ERROR: ", $msg;
exit;
}
?>

<html>
<body>
Message Received. Thank You.
</body>
```

9.1.3 Route by Specific SMS Keyword

SMS Keyword here refers to the first word of the received SMS message. To enable this option, login to the ***sendQuick Messaging Portal*** for Users.

Navigate to ***Keyword Management > Add SMS Keyword***

Figure 5 shows an example of how the keyword "***sms***" is created. Once this keyword is created, users can trigger this keyword by entering an SMS message starting with the word '*sms*', such as '*sms hello world*'.

Note that for HTTP processing, the other input fields of the keyword creation are not required. The process using keywords is the same as that for global receiving script, with the parameters shown in Table 6 applying here.

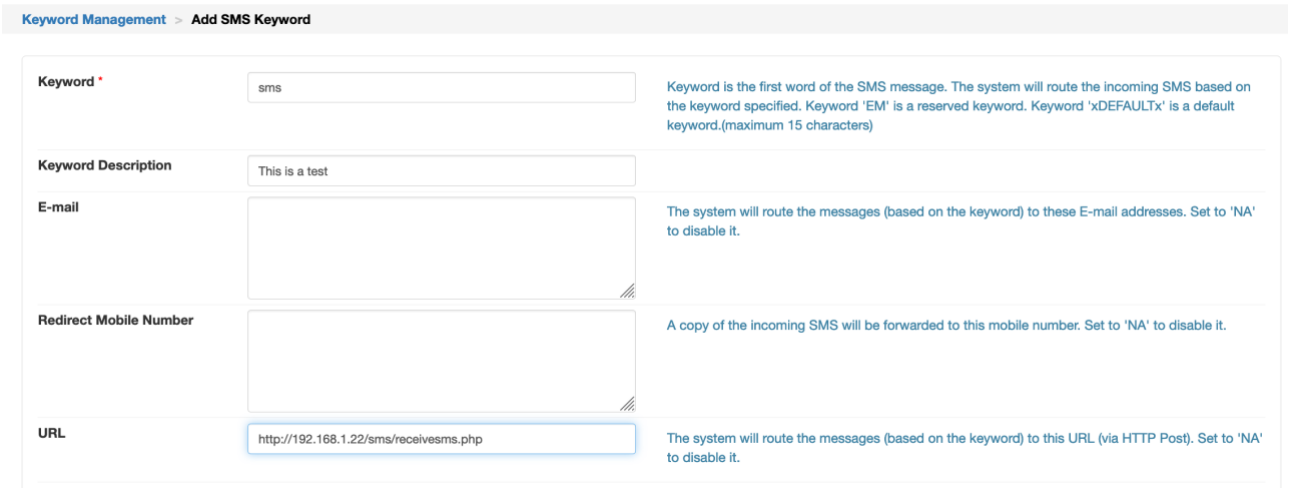

Figure 5: Create SMS Keyword

9.1.4 Script to Generate SMS Response to Sender

The objective of this example is to show how a script can generate an SMS reply to the sender upon receiving the SMS message, using send SMS API

Sample Code : Generating SMS Reply to Sender in PHP

```
<?php
      /* Even though there are total 4 input parameters,
      charset and dtm is not applicable for this example,
      so we ignore them completely. */
      $mno = $_REQUEST['mno'];
      $msg = $_REQUEST['txt'];

      if( !isset($mno) || !isset($msg) ){
            echo "Invalid input - Missing data";
            exit;
      }

      if( strlen($mno) == 0 || strlen($msg) == 0 ){
            echo "Invalid input - Blank data.";
            exit;
      }

      try {
      /* now we had captured the data, we store it into our database table.
      We use sendQuickLite (standard module in PHP 5.1.x and above) to store
the message.
      To create this table, just download sendQuicklite3 from
http://www.sendQuicklite.org/
      use the command line utility to create this database
(messagedb.sendQuicklite)
      and table: incoming_sms:

      CREATE TABLE `incoming_sms` (`idx` INTEGER PRIMARY KEY , `mno` VARCHAR,
`txt` VARCHAR)

      NOTE: Highly recommend to use Firefox plugin to manage sendQuickLite
database:
      http://sendQuicklitemanager.mozdev.org/
      */

      // NOTE: Path to the messagedb.sendQuicklite required to full path.
      $dbh = new PDO('sendQuicklite://<path>/<to>/messagedb.sendQuicklite');

      $stmt = $dbh->prepare("INSERT INTO incoming_sms (mno,txt) VALUES
(:mno, :txt)");
      $stmt->bindParam(':mno',$mno);
      $stmt->bindParam(':txt',$msg);
      $stmt->execute();

      } catch ( Exception $e ) {
            $msg = $e->getMessage();
     echo "DB ERROR: ", $msg;
            exit;
      }

      // Now we generate a response to the sender.
```

```
      $url = "http://192.168.1.22/cmd/system/api/sendsms.cgi";
      $tar_mno = $mno;
      $tar_msg = "You said: $msg\nI said: Thank you.";

      // URL encoding should always be utilised for proper data passing.
      $param = "tar_num=" . urlencode($tar_mno) .
            "&tar_msg=" . urlencode($tar_msg) .
            "&tar_mode=text";

      /* We use curl library for HTTP submit, this may require additional
setup in PHP
      in order to be usable.

      Refer http://www.php.net for the documentation on how this can be
enable. */

      $ch = curl_init();
      curl_setopt($ch, CURLOPT_URL, $url);
      curl_setopt($ch, CURLOPT_POST, 1);
      curl_setopt($ch, CURLOPT_POSTFIELDS, $param);
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
      $urlresp = curl_exec($ch);
?>

<html>
<body>
Message Received. Thank You.
Your response: <?php echo urlresp;?>
</body>
```

9.2 Advance Topic for trackid and status_url

Where AppServer requires sendQuick to return the status of the messages it submitted, AppServer will need to submit the ***trackid*** and ***status _url*** parameters to sendQuick.

With inclusion of these parameters, sendQuick returns the results of messages processed to the URL specified by the ***status_url*** parameter. This process flow is shown in the figure below

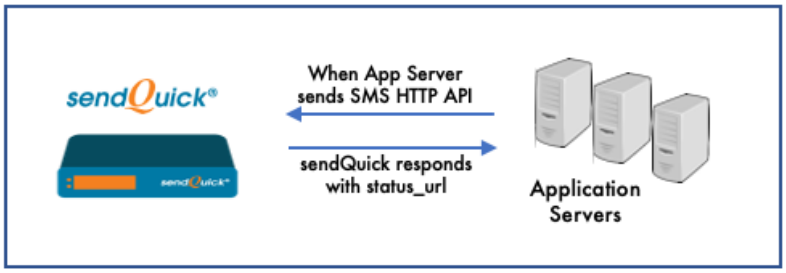

Figure 6: Sending SMS flow with status_url

The next two examples illustrate the sending of SMSes with status tracking: Example 1 is a perl script for making the HTTP submit, and Example 2 is a php script which reads the input of the message and generates an email to inform the user of message status.

Example 1 : Sending SMS with Status Tracking using HTTP (perl script)

```
#!/usr/bin/perl -w

use LWP::UserAgent;
use URI::Escape;

# Create the simple HTTP agent.
$ua = LWP::UserAgent->new;
$ua->agent("MyTestApp/0.1 ");

# Composing the parameters and content.
my $tar_msg = "testing 1 2 3 -- From TalariaX (Singapore)";
my $mno = '91234567';
my $req = HTTP::Request->new(POST =>
'http://192.168.1.101/cmd/system/api/sendsms.cgi');
$req->content_type('application/x-www-form-urlencoded');

# Now we add 2 additional parameters: trackid and status_url.
# Here we assume the AppServer is with IP: 192.168.1.22
$req->content('tar_num=' . uri_escape($mno) .
      '&trackid=0001' .
      '&status_url=' . uri_escape("http://192.168.1.22/sms/status.php") .
      '&tar_mode=text' .
      '&tar_msg=' . uri_escape($tar_msg));

# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
```

```
      print "Submit successful: " , $res->content, "\n";
} else {
      print "Submit failure: " , $res->status_line, "\n";
}
```

Example 2 : Example of Status URL in PHP

```
<?php
      /* The status_url will be provided with the following input parameters:
      - mno : the mobile number that was processed.
      - trackid : The trackid that given initially
      - totalsms : The total number of SMS sent for submission of sending the
message.
      - status : The status of this message. Y – successfully sent; F –
failure to send.
      - smsc : The SMSC of SIM card used for sending SMS.
      - imei : The IMEI number of the modem used for sending SMS.
      */
      $mno = $_REQUEST['mno'];
      $trackid = $_REQUEST['trackid'];
      $totalsms = $_REQUEST['totalsms'];
      $stat = $_REQUEST['status'];
      $smsc = $_REQUEST['smsc'];
      $imei = $_REQUEST['imei'];

// After we read the status, we generate an email to
// inform someone about the status of this message

$mailmsg = "The message for $mno for trackid: $trackid " .
"with total SMS: $totalsms was returned with status: $stat";

// We use the standard mail() of PHP.
$to = 'seiheng@talariax.com';
$subject = 'SMS Notice for ' . $trackid;
$message = $mailmsg;
$headers = 'From: sms-php-example@talariax.com' . "\r\n" .
'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);
?>


<html>
<body>
<?php echo $mailmsg; ?>
</body>
</html>
```

9.3 Using Parameter 'label'

To use the ‘label’ parameter, a string must be associated with a modem. Refer to figure below for the association screen. This string cannot contain the space character. On the sendQuick admin web interface, navigate to ***Modem Setup > Modem Routing***

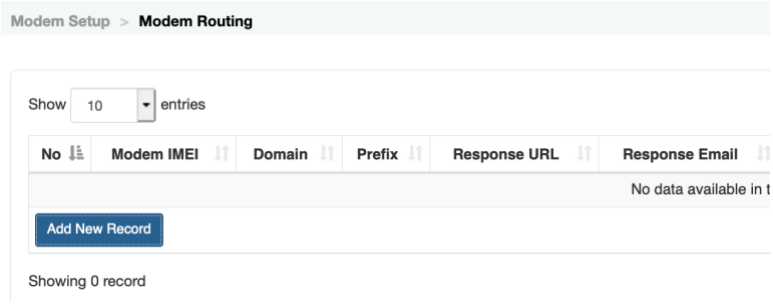

Figure 7: Modem Routing

Click on “Add New Record” and fill in the Modem IMEI and the label to be used to identify this modem.

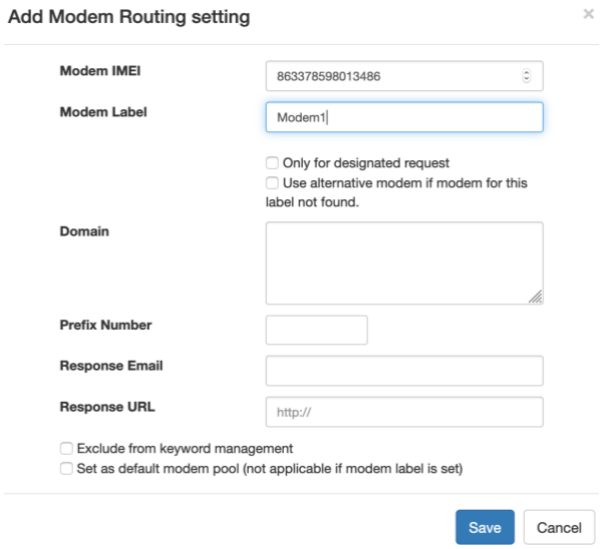

Figure 8: Add Label To Modem

Some points to note:

- Mode IMEI is the unique string set for each modem. This can be found under the modem status page.

- Modem Label can be any string (without space character), and can be a same string of another modem. If there are 2 modems set with a same string, the system would choose the modem that is more idle.

- A modem can be associated with more than 1 label. Each label need to be separated by a "," character.

9.4 Useful Free Utilities

Several free utilities for sending SMSes are discussed in this section.

9.4.1 curl

curl is a command line tool for transferring files with URL syntax, and supports FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS and FILE. This tool is available for Linux and Windows platforms, and is very useful for applications capable of executing commands to link to other applications.

Example : Sending SMSes using curl

$> curl -- data-urlencode "tar_num=91234567" --data-urlencode "tar_msg=this is a test SMS & it is using curl for sending" http://<sendQuickIP>/cmd/system/api/sendsms.cgi

NOTE: --data-urlencode is only available for version 7.18.0 and above. For earlier versions, the content needs to be URL encoded before submission to curl for execution. Visit https://curl.haxx.se/ to get more information and also to download curl.

9.4.2 Java Command Tool

This Java example command tool is written by TalariaX. It is a simple utility for sending SMS via HTTP POST to sendQuick device.

Example:

$> java httppost “91234567” “this is a test”
http://<sendQuickIP>/cmd/system/api/sendsms.cgi

Sample Code : Java command tool for sending SMS.

Filename: httppost.java

```
import java.lang.*;
import java.net.*;
import java.io.*;

public class httppost {

      private static String sendsms_url = "http://<sendQuick
IP>/cmd/system/api/sendsms.cgi";

      public static void main(String[] args){

            /*
             * java httppost <mobile number> <SMS message> <sendQuick URL>
             * eg:
             * java httppost "+6591234567" "This is a test SMS" <sendQuick
URL>
```

```
		 */
		String mno = args[0];
		String msg = args[1];
		String url = args[2];

		sendsms_url = url;

		HttpSubmit(mno, msg);
	}

	public static void HttpSubmit(String mno, String msg)
	{
		URL url;
		URLConnection urlConn;
		DataOutputStream printout;
		BufferedReader input;

		try {
			// URL of CGI-Bin script.
			url = new URL (sendsms_url);
			// URL connection channel.
			urlConn = url.openConnection();
			// Let the run-time system (RTS) know that we want input.
			urlConn.setDoInput (true);
			// Let the RTS know that we want to do output.
			urlConn.setDoOutput (true);

			// No caching, we want the real thing.
			urlConn.setUseCaches (false);

			// Specify the content type.
			urlConn.setRequestProperty("Content-Type",
				"application/x-www-form-urlencoded");

			// Send POST output.
			printout = new DataOutputStream (urlConn.getOutputStream
());
			String content ="tar_num=" + URLEncoder.encode (mno, "UTF-
8") +
				"&tar_msg=" + URLEncoder.encode(msg, "UTF-8");
			printout.writeBytes (content);
			printout.flush ();
			printout.close ();

			// Get response data.
			input = new BufferedReader(new
InputStreamReader(urlConn.getInputStream()));
			String str;
			while (null != ((str = input.readLine()))) {
			System.out.println (str);
			}
			input.close ();
		} catch ( Exception e ){
			e.printStackTrace();
		}
	}
}
```