SendQuick API Guide

Version 2.9

TalariaX Pte Ltd
No 76, Playfair Road,
#08-01, LHK2
Singapore 369583

E-mail: info@talariax.com
Web: www.talariax.com

Table of Contents

1. Introduction

This document gives an overview on the different methods of connecting between sendQuick SMS gateway (sQ) and an application server (AppServ), for sending and receiving SMS.

There are several ways of connecting to sQ:
• SMTP (email)
• HTTP
• FTP
• XML
• SOAP
• JSON

2. SMTP (Email) Method

Sending SMS

The setting for the message sending e-mail is <tar_num>@<serverIP/domainname>

Example:
If you set the IP address 192.168.1.8, then the e-mail for sending messages is
<label>-<tar_num>@192.168.1.8

You will need to e-mail to sQ server in the following format:

To	:	label-91234567@192.168.1.8
Subject	:	Test message
Message	:	Test message for you.

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

Receiving SMS

The received SMS message will be sent to the e-mail specified. An example of the received e-mail is as below.

From	:	91234567@192.168.1.8
Date	:	Monday, June 10, 2002 2:31 PM
To	:	sh_ang@yahoo.com <sh_ang@yahoo.com>
Subject	:	SMS From 91234567

Sender: 91234567
Timestamp: 02/06/10,14:32:54+32
Message: Test

3. Send and Receive SMS API

3.1 Overall Transaction Flow

Refer to the diagram below:

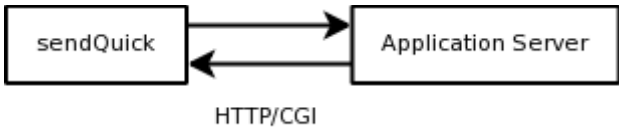


Figure 1.1: Overall Transaction Flow

Description:

1. Since SMS are basically text content messages, thus simple HTTP or web submit (POST or GET) should be sufficient for transferring the data between the sQ and AppServ. Both sQ and AppServ will be communicating via standard HTTP (POST/GET) access.
2. AppServ need to enable its web service to accept HTTP (POST/GET) submit from sQ.
3. sQ will need to be configured the URL path submission to AppServ. This can either be a global or per keyword configuration.
4. For sending SMS, AppServ will need to initiate a HTTP (POST/GET) to the HTTP API of the sQ system.

3.2 Send SMS API

sendQuick HTTP API URL path: /cmd/system/api/sendsms.cgi

The input parameters are as follows:

Parameter	*Description*	*Data Type*
tar_num	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. Example: +6591234567 For multiple numbers, +651234567, +658888888
tar_msg	This parameter is the message content for the recipient. This string must be encoded into URL encoded string, otherwise, the recipient may receive partial or corrupted message.	The message can contain alphabets, numbers and special characters. The message should be html encoded. Message value shouldn't be empty. Example 1: This is a test message for ASCII. Example 2: 안녕하세요 수 있습니다 for UTF8.

Parameter	***Description***	***Data Type***
tar_mode	The processing mode. • text – ASCII text SMS • flashtext – flash SMS for ASCII characters • utf8 – 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 “utf8”. Flash sms can support ASCII characters. Set the mode as “flashtext” for flash sms. The parameter value is case-sensitive. Example: text
clientid	The identifier tag to be triggered if there are more than one applications running on the same server. This field is optional.	The value can be alphanumeric. Example: A1245G
trackid*	The tracking ID for each request. This field is optional.	Trackid value should be an integer. Example: 1001
status_url*	The message status URL that the system will respond to when the message has completed processing. This field is optional.	Should provide the full URL in order to post back. http://<ip>/<path> Example: http://192.168.1.10/webroot/mymessage-status.php
priority	This is an advisory parameter for setting the priority of the outgoing SMS.	The value should be an integer between 1-9. Highest Prioirty = 1, Lowest Priority = 9. If the priority is not set, the default is value '5'. Example: 5
dr	To request delivery report for this message. 1 to enable, 0 to disable. If it is not specified, the system will use the configuration specified in the SMS System Setup.	The value should be integer, either 1 or 0. Example: 1
mov	Message overwrite would make the handset replace previously received SMS with the new SMS. 1 to enable, 0 to disable. Note: This feature may not be applicable for some mobile handset such as iPhone. Note: This feature may not be applicable	The value should be integer, either 1 or 0. Example: 1

Parameter	*Description*	*Data Type*
	for some mobile handset such as iPhone.	
label**	To send the message via specific modem. To use this parameter, a label must be associated with a modem.	The value can be alphanumeric. Example: marketing1

* *The usage of trackid and status_url will be discussed in the section 3.5.*
** *Refer to section 3.7.6 for discussion.*
\# *Standard HTTP (POST/GET) treat '+' as a blank space.*

3.2.1 HTTP Method

Example for Single Mobile
http://192.168.1.101/cmd/system/api/sendsms.cgi?tar_num=+6512345678&tar_msg=test&tar_mode=text

Example for Multiple Mobile
http://192.168.1.101/cmd/system/api/sendsms.cgi?
tar_num=+6512345678,+65912345679,+65912345676&tar_msg=test&tar_mode=text

sendsms.cgi Response

Type	Response
For single tar_num	OK Queue: <sendQuick-message-ID>
For multiple tar_num_n	OK Queue: <sendQuick-message-ID1>-<tar_num1> <sendQuick-message-ID2>-<tar_num2> <sendQuick-message-ID3>-<tar_num3> ... <sendQuick-message-ID_n>-<tar_num1_n>

3.2.2 XML Method

API : http://ip address/webapp/sendsms_xml.php

Example for Single Mobile

```
<?xml version="1.0"?>
        <post_data>
                <info>
```

```
                <tar_num>+65912345678</tar_num>
                <tar_msg>test xml frm </tar_msg>
                <tar_mode>text</tar_mode>
                <priority>3</priority>
        </info>
</post_data>
```

Exaple for Multiple Mobiles

```
<?xml version="1.0"?>
        <post_data>
                <info>
                        <tar_num>+65912345678,+65912345679,+65912345676</tar_num>
                        <tar_msg>test xml </tar_msg>
                        <tar_mode>text</tar_mode>
                        <priority>3</priority>
                </info>
        </post_data>
```

sendsms_xml.php Response

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

3.2.3 SOAP Method

API : http://ip address/webapp/sendsms_soap.php

Example for Single Mobile

```
POST /webapp/sendsms_soap.php 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: 651

<?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"><tar_num xsi:type="xsd:string">+65912345678</tar_num><tar_msg
xsi:type="xsd:string">test soap</tar_msg><tar_mode
xsi:type="xsd:string">text</tar_mode></ns5130:processAPI></SOAP-ENV:Body></SOAP-ENV:Envelope>

Example for Multiple Mobiles

POST /webapp/sendsms_soap.php 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: 651

<?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"><tar_num
xsi:type="xsd:string">+65912345678,+65912345679,+65912345676</tar_num><tar_msg xsi:type="xsd:string">test
soap </tar_msg><tar_mode xsi:type="xsd:string">text</tar_mode></ns5130:processAPI></SOAP-
ENV:Body></SOAP-ENV:Envelope>

sendsms_soap.php Response

Type	Response
For single tar_num	HTTP/1.1 200 OK Date: Wed, 31 Oct 2012 02:57:55 GMT Server: Apache X-Powered-By: PHP/5.4.4 Set-Cookie: PHPSESSID=rsoojeoc8k3emff48lhc643uq0; path=/ Expires: Thu, 19 Nov 1981 08:52:00 GMT Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 Pragma: no-cache X-SOAP-Server: NuSOAP/0.7.3 (1.114) Content-Length: 493 Connection: close Content-Type: text/xml; charset=utf-8 <?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">+65912345678:10</return></ns1:processAPIResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
For multiple tar_num$_n$	HTTP/1.1 200 OK Date: Wed, 31 Oct 2012 02:57:55 GMT

	Server: Apache X-Powered-By: PHP/5.4.4 Set-Cookie: PHPSESSID=rsoojeoc8k3emff48lhc643uq0; path=/ Expires: Thu, 19 Nov 1981 08:52:00 GMT Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 Pragma: no-cache X-SOAP-Server: NuSOAP/0.7.3 (1.114) Content-Length: 493 Connection: close Content-Type: text/xml; charset=utf-8 <?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">+65912345678:10 +65912345679:11 +65912345676:12</return></ns1:processAPIResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>

3.2.4 JSON Method

API : http://ip address/webapp/sendsms_json.php

Example for Single Mobile

```
{
    "Entry": {
            "tar_num": "+65912345678 "
            "tar_msg": "Test json",
            "tar_mode": "text",
            }
  }
```

Example for Multiple Mobiles

```
{
    "Entry": {
            "tar_num": "+65912345678,+65912345679,+65912345676"
            "tar_msg": "Test json",
            "tar_mode": "text",
            }
  }
```

sendsms_json.php Response

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

3.3 Receiving SMS

This is for incoming SMS. sendQuick will post to the Client API registered in sendQuick. The receiving script should accept the following input parameters:

Parameters	***Description***
mno	The mobile number of the sender.
txt	The content of the SMS message.
dtm	The date and time when the message was received.
charset	The character set of the content. Either 'text' for ASCII message or 'utf-8' for unicode UTF-8 message.

3.3.1 HTTP Method

Example

http://192.168.1.102/clientapi_path/clientapi.cgi?mno=+6512345678&txt=test&dtm=05/11/2012 17:57:01&charset=text

3.3.2 XML Method

Example

http://192.168.1.102/clientapi_path/clientapi.cgi

```
<?xml version="1.0"?>
        <mno>+6512345678</mno>
        <txt>test</txt>
        <dtm>05/11/2012 17:57:01</dtm>
        <charset>text</charset>
```

3.3.3 SOAP Method

Example

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">+6512345678</mno><txt xsi:type="xsd:string">test soap</txt><charset xsi:type="xsd:string">text</charset><dtm xsi:type="xsd:string">05/11/2012 17:57:01</dtm></ns5130:processAPI></SOAP-ENV:Body></SOAP-ENV:Envelope>

3.3.4 JSON Method

Example

http://192.168.1.102/clientapi_path/clientapi.cgi

```
{
  "Entry": {
        "mno": "+6512345678 "
        "txt": "test",
        "dtm": "05/11/2012 17:57:01",
        "charset": "text",
        }
}
```

3.4 Getting Outgoing SMS Status via HTTP

This is for sendQuick server to call back client's status_url to update the message status, after sending out SMS. The status_url should accept the following input parameters:

Parameters	***Description***
trackid	This will be the trackid submitted by the AppServ for sQ to track the status of the outgoing SMS
status	This will be the status of the outgoing SMS. • Y – sent successful • F – failed to send. • D – Message Delete from server • R – Received delivery report from recipient mobile phone.
totalsms	This will be the integer value of the total SMS used for sending the

Parameters	*Description*
	complete message. NOTE: Not applicable for status D and R
mno	The mobile number of the recipient.

Example

Step 1: *http://192.168.1.101/cmd/system/api/sendsms.cgi?tar_num=+6512345678&tar_msg=test&tar_mode=text&trackid=1001&status_url=http://yourclientserverip/apipath/api.cgi*

Step 2*: SMS will be processed in sendQuick server accordingly and sendQuick will response the message status upon Successful or Failure to the status_url*

Step 3: *http://yourclientserverip/apipath/api.cgi?trackid=1001&status=1&totalsms=1&mno=+6512345678*

3.5 Delete Outgoing SMS via HTTP

API URL path: /cmd/system/api/reqdelete.cgi

The input parameters are as follows:

Parameters	*Description*
trackid	The trackid sent by the AppServ
tar_num	The mobile number associated with the trackid.

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

http://192.168.1.101/cmd/system/api/reqdelete.cgi?tar_num=+6512345678&trackid=1001

3.6 Check Outgoing SMS Status via HTTP

API status URL path: /cmd/system/api/msgstatus.cgi (if sendsms.cgi is used for sending sms)

The input parameters are as follows:

Parameters	*Description*	*Data Type*
mode	“queue” only.	The value should be alphanumeric which is as predefined. Example: queue

msgid	The <sendQuick-message-ID> returned from sendsms.cgi	The message id returned from the API. Example: M12345

Example

Sample HTTP Request

http://192.168.1.20/cmd/system/api/msgstatus.cgi?mode=queue&msgid=M12345
OR
http://192.168.1.20/webapp/msgstatus.php?mode=queue&msgid=m12345

Sample HTTP Response

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

31-10-012 10:57:56 31-10-012 18:33:45 Y 359126030021471 +6596845999 +6582821664 test message

The URL response when no associating <sendQuick-message-ID> is found
No result found: <sendQuick-message-ID>

Note: This may happen, if the <sendQuick-message-ID> is invalid or the message has been deleted from the system.

The URL response when the associating <sendQuick-message-ID> is found
<start-process-dtm><tab><completed-process-dtm><tab><status><tab><modem-imei><tab><smsc><tab><tar_num><tab><tar_msg>

Example:
17/10/2012 17:39:40 17/10/2012 17:39:57 Y 359126030021471 +6596845999 202.63.133.40 96189556 this is a test

3.7 Example Applications

In this section, a few sample scripts will be shown on how the application server (AppServer) needs to be prepared for accepting input from sQ system. The programming language for the example will be in PHP or Perl, but the underlying concept is similar for all HTTP scripting or programming platform. The discussion will be presented using a few example applications.

3.7.1 Sending of SMS

The following sample code show a simple HTTP submission in Perl

Sample Code 1: Simple HTTP CGI submit 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 = '96189556';
my $req = HTTP::Request->new(POST => 'http://192.168.1.8/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 script assume the IP address of sQ is 192.168.1.8.
- This script use LWP module of Perl. Please download the LWP module from http://search.cpan.org in order to test the script.

3.7.2 Global Script for Receiving SMS

To configure global receiving script, we need to specify the URL path to the sendQuick server. This is done by configuration in the SMS System Setup. Refer to the sample screenshot below:

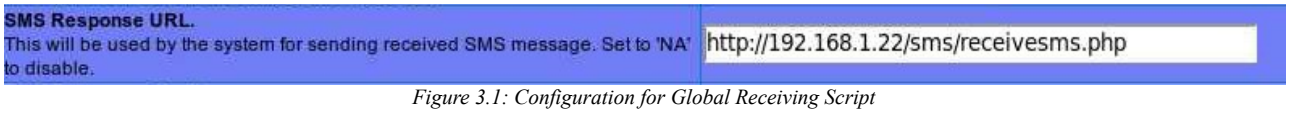


Figure 3.1: Configuration for Global Receiving Script

Description:

- Enter the URL path into the “SMS Response URL” in the “SMS System Setup”.
- The screenshot above assume the AppServ IP: 192.168.1.22 and path is /sms/receivesms.php

Refer to the following example on how receiving script read the received SMS.

Sample Code 2: How receiving SMS can be done 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 SQLite (standard module in PHP 5.1.x and above) to store the message.
                To create this table, just download sqlite3 from http://www.sqlite.org/
                use the command line utility to create this database (messagedb.sqlite)
                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 SQLite database:
                http://sqlitemanager.mozdev.org/
                */

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

                $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>
```

3.7.3 Route by Specific SMS Keyword

SMS Keyword here refers to the first word of the received SMS message. To enable this option, create the desired SMS keyword under "SMS Keyword Management". Refer to the following screenshot:

Figure 3.2: Create a SMS Keyword

Description:

- In this example, the SMS keyword is 'sms'. This means that in order to trigger this processing, user will need to enter a SMS message starting with the word 'sms'. eg: 'sms hello world'
- For HTTP processing, the other input fields of the keyword creation are not needed.
- Since the input parameters will be the same as the global receiving script, refer to Example 3.2 for how receiving process can be done.

3.7.4 Script that Generate a SMS Response to Sender

The objective of this example is to show how a script can generate a SMS reply to the sender upon receiving the SMS message. It will use the send SMS API (describe in section 3.2) for sending of SMS messages.

Sample Code 3: Sending SMS back to sender once received a SMS 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 SQLite (standard module in PHP 5.1.x and above) to store the message.
        To create this table, just download sqlite3 from http://www.sqlite.org/
        use the command line utility to create this database (messagedb.sqlite)
        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 SQLite database:
        http://sqlitemanager.mozdev.org/
        */

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

        $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://192.168.1.27/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>
```

3.7.5 Advance Topic for trackid and status_url

In some situation, AppServ require sQ to return the status of the message it submitted. Under such requirement, AppServ will need to submit 2 additional parameters, trackid and status_url, to sQ. Once the message has been completed in processing, sQ will return the result, based on trackid, to the URL specified by status_url parameter.

Processing flow (refer figure 4):

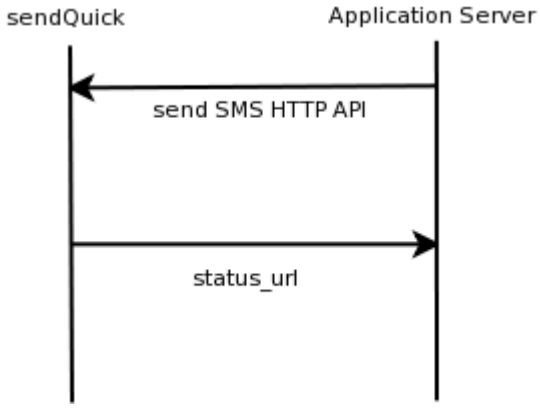


Figure 4: Sending SMS flow with status_url

Description:

- Application Server (AppServ) call send SMS HTTP API for sending SMS.
- AppServ must provide trackid and status_url for sendQuick (sQ) to return the process result to AppServ.
- Upon completed processing, sQ will call the status_url and return the result of the outgoing SMS message.

To demonstrate this example, we will need 2 scripts. The first script (Example 4) will be a perl script for making HTTP submit, the second script (Example 5) will be a php script that read the input of the message and generate an email to inform the user of message status.

Sample Code 4: Sending SMS with status URL

```
#!/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 = '96189556';
my $req = HTTP::Request->new(POST => 'http://192.168.1.8/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 AppServ 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";
}
```

Sample Code 5: 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>
```

3.7.6 Using Parameter 'label'

To use the parameter, a string must be used to associate with a modem. This string cannot contain space character. Refer to below screenshot:

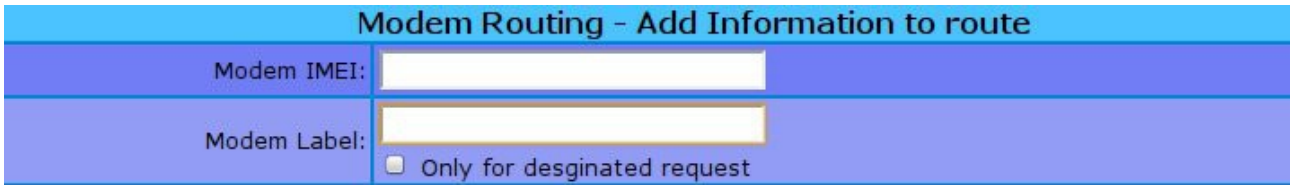


Figure 3.3: Modem Routing

Description:

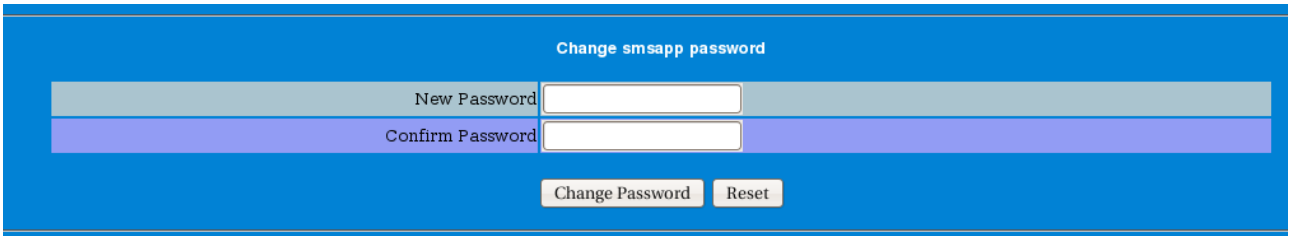
- Mode IMEI is the unique string set for each modem. This can be found under the modems status page.
- Modem Label can be any string (without space character), and can be a same string of another modem. If the there is 2 modem set with a same string, the system would choose these modem that at the modem is more idle than the other.
- A modem can be associated with more than 1 label. Each label need to separated by a "," character.

4. FTP Method

You will need to first activate the FTP service in sendQuick.
Login to sQ web interface as Server Administrator > SMS System Setup > Enable FTP to SMS Service. Then, Save Setting > Activate Setting.

To configure the FTP account and Password

- URL: http://sendQuick_server_IP/appliance/
- Enter admin username and password
- Then click 'Change Password' from Menu
- Enter your new password at 'change smsapp password'



FTP Folder Name on the server: /smsftp/

This is the directory that you will be placing the file, once you login as 'smsapp' username

4.1 FTP File Structure & Format

(a) File Structure

FieldSize	Remarks
Hand Phonex(15)	Keep to numeric
Messagex(160)	

(b) File Format

The data in the file must be according to the following sequence and using comma delimiter (comma separated value). There need to be a double-quote for the data (as below). When the data file is created by Excel spreadsheet, the double-quote is being automatically created. You can check the formatting by opening the file in a text editor, like Notepad.

handphone,message

handphone1,message1

Eg:

“96367680”,”how are you?”

“96180556”,”I am doing fine”

(c) File Type

File type allowed: *text file with extension .msg.* There are 2 text files to upload by FTP (*.msg and *.end). The data must be saved in the .csv and .end is just a blank file. At first, you have to put .msg file in the folder (/home/smsapp/smsftp) and after successfully uploaded the .msg file, you have to put the .end file in that folder to infom the system that file uploading is completed.

5. Sending SMS to sendQuick ASP (Hosted/Cloud SMS)

This section covers the method for sending SMS from App Server to sendQuick ASP (SMS cloud service). The following steps are for sending SMS via SMS Proxy link:

Step 1
You will need to provide your IP (to send to sendQuick ASP), so that this can be configured in sendQuick ASP system.

Step 2
Our sendQuick ASP Administrator will provide a username and password.

Step 3
URL for submitting SMS: http://www.sendquickasp.com/cmd/system/api/smsproxy.cgi

The input parameters are as follows:

Parameter	***Description***	***Data Type***
tar_num	This is the parameter for target or recipient mobile number.	The number has to be numeric, with the country code specified (for both local and international). '+' sign is not needed. Example: 6591234567
tar_msg	This parameter is the message content for the recipient. This string must be encoded into URL encoded string, otherwise, the recipient may receive partial or corrupted message.	The message can contain alphabets, numbers and special characters. The message should be html encoded. Message value shouldn't be empty. Example 1: This is a test message for ASCII. Example 2: 안녕하세요 수 있습니다 for UTF8.
username	This will be provided by our sendQuick ASP administrator	Alphanumeric format
passwd	This will be provided by our sendQuick ASP administrator	Alphanumeric format
callerid	This is the name or code that will appear on the FROM field of the SMS.	Alphanumeric format. Up to a maximum of 11 characters is allowed.

Example:

To send a message "test SMS" to mobile number: 6596189556. (Assume user name and password is "user" and "pass" respectively). The callerid is assumed to be "TEST"

URL: http://www.sendquickasp.com/cmd/system/api/smsproxy.cgi
tar_num=6596189556
tar_msg=test SMS
callerid=TEST
username=user
passwd=pass

The following code is the sample code in perl to submit the request of the example above:

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

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

BEGIN:
{
	my $url = "http://www.sendquickasp.com/cmd/system/api/smsproxy.cgi";
	my $msg = "test SMS";
	my $ua = LWP::UserAgent->new();
	my $req = HTTP::Request->new(POST => $url);
	my $param = "tar_num=6596189556&tar_msg=" . uri_escape($msg) .
		"&username=user&passwd=pass&callerid=ipx";

	$req->content($param);

	my $res = $ua->request($req);

	if( $res->is_success ){
	   print $res->content, "\n";
	} else {
		print $res->status_line, "\n";
	}

	exit;
}
```

NOTE:

- The link assumes an international number format. Thus, all number must provide the country code and area code if applicable (without the '+' character).
- uri_escape should always be called whenever it is needed.

Return Result:

- sent
This means the message was sent successful

- unsent <text>
This means the message not sent successful. <text> is optional description/remark of the post result.

- queued
This means, the message has been queued, and awaits provider to return the success status.

6. Useful Free Utilities

This section will discuss some freely available utilities for sending SMS.

6.1 curl

curl is a command line tool for transferring files with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS and FILE. It available for Linux and Windows platform.
Since is a command line tool, it is very useful for applications that capable of executing external applications.

Example:

```
$> curl --data-urlencode "tar_num=9123567" --data-urlencode "tar_msg=this is a test SMS & it is using curl for sending" http://<sendQuick IP>/cmd/system/api/sendsms.cgi
```

NOTE: --data-urlencode only available for version 7.18.0 and above. It is highly recommended to use 7.18.0 and above for applications. For lower version, before executing curl, the content will need to be URL encoded before submission.

Curl can be downloaded from:

http://curl.haxx.se/

6.2 wget

GNU Wget is a free software package for retrieving files using HTTP, HTTPS and FTP, the most widely-used Internet protocols. It is a non-interactive command line tool, so it may easily be called from scripts, cron jobs, terminals without X-Windows support, etc.
Example:

```
$> wget --post-data 'tar_num=9123456&tar_msg=this is a test' -O result.txt  http://<sendQuick IP>/cmd/system/api/sendsms.cgi
--2008-05-20 18:00:08--  http://202.172.42.35/cmd/system/api/sendsms.cgi
Connecting to 202.172.42.35:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: `result.txt'

   [ <=>                                   ] 19          --.-K/s   in 0s

2008-05-20 18:00:09 (1.10 MB/s) - `result.txt' saved [19]
```

This example will save the POST result into 'result.txt'. However, wget does not perform URL encoding. Hence this utility only suitable for messages without special characters. Wget comes default for all Linux platform. For Windows, the latest version can be found from here:

http://www.christopherlewis.com/WGet/default.htm

9.3 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 “96189556” “this is a test” http://<sendQuick IP>/cmd/system/api/sendsms.cgi

Sample Code 6: 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();
                    }
          }
}
```

Appendix A: Change History

Version 1.0

- Consolidated sendQuick HTTP API Guide v2.7 into sendQuick API Guide 1.0
- Added SMTP, FTP methods and new API methods XML, SOAP, JSON
- Added new parameter “clientid”
- Added new API URL path for checking outgoing SMS status via XML, SOAP, JSON
- Added “Data Type” for parameter specifications
- Added examples for HTTP API

Version 1.1

- Added XML/SOAP/JSON for incoming message
- Re-arrange the section of header. Remove header 6/7/8 and put it under 3.2.