Globe SMS/MMS API Developer's Guide

v. 2.1

January 2009

Document History

To ensure that you have the most current version of this document, please check the Globe Labs website (http://www.globelabs.com.ph)

Version	Author	Date	Notes
1.02	Globe Labs	July 25, 2008	Initial Public Release
1.03	Globe Labs / Melvin Dave P. Vivas	July 28, 2008	Addition of Sample Java Code for sendSMS method
1.04	Globe Labs	July 29, 2008	Added new response codes and fields in XML messaging
1.05	Globe Labs	Aug 5, 2008	Changed XML examples in Appendices
1.06	Globe Labs	Aug 28, 2008	Fixed typographical errors for "messageString" in sample code and XML Added messageType node for all types of callbacks. SMS for SMS messages, MMS for MMS messages and SMS-Notification for delivery notifications Added sample SOAP-XML response example in Appendices Fixed typo on length of time mms data is stored in system from 30 seconds to 30 minutes Added new error code [509]
1.07	Globe Labs	Oct 6, 2008	Added additional notes when sending binary data over SMS
1.08	Globe Labs	Oct 13, 2008	Fixed minor error in C# sample code.
1.09	Globe Labs	Nov. 6, 2008	Fixed Java example to work with API by adding named parameters is sample code.
2.0	Globe Labs	Jan. 30, 2009	Incorporate new information about registration, terms of service and a REST based interface.
2.1	Globe Labs	Feb 18, 2009	Case adjustments in the sendMMS method. From SMIL to smil and Subject to subject.

TABLE OF CONTENTS

Introduction 4

Quick Start 6

API Description 9

Sending Messages 10

XML-Based Delivery Notification 17

Receiving Messages 19

Application Container 22

Useful Tools 22

Links 22

Appendices 23

INTRODUCTION

WHAT IS IT?

The Globe SMS/MMS API allows third-party developers to integrate SMS and MMS messaging into their web-based or desktop applications via a web-accessible / web-services-based interface. You can use the API to send SMS and MMS messages to Globe subscribers via remote procedure calls using the Simple Object Access Protocol (SOAP) over HTTP as well as a REST based URL interface over HTTP. The API also gives the ability for developers to receive SMS and MMS messages from Globe subscribers via a combination of XML messaging, a short-code number (2373) and a pre-assigned 4-digit suffix/code.

The API is intended to encourage innovation by opening Globe's telecommunications infrastructure to new and unique mash-up applications and technologies. It is currently being developed and hosted by Globe Labs.

HOW DO I GET STARTED?

To get started using the API, you'll need the following:

- A web-server / service that can process HTTP POST requests on a host that is publicly accessible over the Internet via an IP address or URL
- Working knowledge of web-services / SOAP OR knowledge on how to make remote URL calls in the language of your choice
- A registered developer account with Globe Labs
- A good idea for a brand-new application you want to unleash to the world

Access to the API is granted on a per application basis. You need to register your application with Globe Labs and specify that your application will require the SMS/MMS API. You also need to provide a set of Globe numbers that will be pre-approved for the SMS/MMS service and the call-back URL your application will be using.

Globe Labs will assess your request and get back to you within 3 working days. Once you have been granted access, you'll receive an email notification saying that you have been granted access to the API. The email notification will also inform you about the application's access details like your username and password.

You are free to register more than one application.

For more detailed information on how to gain access to the API, please refer to the Globe Labs website http://www.globelabs.com.ph.

RESTRICTIONS / TERMS OF SERVICE

1. Only Globe numbers will be allowed to utilize the API, both in terms of sending messages via the 4 digit short code and for receiving messages from the API.

2. The API can process multi-part messages but only up to a maximum of 459 characters

3. The maximum size of a MMS message is up to 300 KB. The API will warn the user if the media file size is too large.

4. Because the API is still in beta, Globe Labs cannot guarantee message delivery times and API uptime.

7. While the API is still in beta, sending the receiving messages via the API will be free of charge.

COMMON TERMS / ACRONYMS

SMS -- Short Messaging System

MMS -- Multimedia Messaging System

API -- Application Programming Interface

SMSC -- Short Message Service Center

MMSC -- Multimedia Message Service

SOAP -- Simple Object Access Protocol

HTTP -- Hypertext Transfer Protocol

QUICK START

Using the API is fairly straightforward. It was designed to allow easy integration between different platforms and languages using the SOAP protocol. Let's create a simple application which allows a webpage to send an SMS message.

```
<?php
// load the nusoap libraries. These are slower than those built in PHP5 but don't require
you to recompile PHP
include_once("nusoap/lib/nusoap.php");
// create the client and define the URL endpoint
$client = new
soapclient('http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/');
// set the character encoding, utf-8 is the standard.
$client->soap_defencoding = 'UTF-8';
// check if we generated an error in creating the client / assigning the endpoint
$err = $client->getError();

if ($err)
{// Display the error
        $error_message = 'Constructor error: ' . $err;
}
// check if a message was sent
if (!empty($_POST['send']))
{// Call the SOAP method, note the definition of the xmlnamespace as the third parameter
in the call and how the posted message is added to the message string
        $result = $client->call('sendSMS', array(    'uName' => 'username',
'uPin' => 'password',
'MSISDN' => '0917xxxxxxx',
'messageString' => $_POST['msg'],
'Display' => '0',
'udh' => '',
'mwi' => '',
'coding' => '0' ),
"http://ESCPlatform/xsd");

// Check for a fault
        if ($client->fault)
        {
                $error_message = "Fault Generated: \n";
        }
        else
        {// Check for errors
                $err = $client->getError();

                if ($err)
                {// Display the error
                        $error_message = "An unknown error was generated: \n";
                }
                else
                {// Display the result
                        if ($result == "201")
                        {
                                $error_message = "Message was successfully sent!";
                        }
                        else
                        {
                                $error_message = "Server responded with a $result
message";
                        }
                }
        }
}// end if

<html>
<head></head>
<body>
<h1>SMS API Demo</h1>
<p>Type in your message below and I'll be able to get your message via SMS.</p>
```

```
<form method="post">
<textarea name="msg"></textarea>
<br>
<input type="submit" name="send" value="Send">
</form>
<div style="color: red"><?php echo $error_message ?></div>
</body>
```

For this example, we've used PHP and NuSOAP. Most web hosting services these days provide PHP support at very affordable prices. Some of hosting providers may not provide built-in SOAP support in PHP5 This is where the NuSOAP library comes in handy.

The first line below shows the script creating a new instance of the `soapclient` class (as defined in the NuSOAP library). The constructor takes an URL endpoint as an argument and then checks if the URL endpoint is valid. On the second line, we set the text encoding parameter to utf-8.

```
$client = new soapclient(
'http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/');
$client->soap_defencoding = 'UTF-8';
```

The code below shows the `soapclient` object executing a `call()` method. The `call()` function does the actual work of creating a SOAP-compatible XML message and also sends the XML to our predefined endpoint.

The call method takes three arguments: the remote method to be called (`sendSMS`), the parameters for the remote call (an associative array that matches the parameters of the web-service) and finally the XML namespace to be used in the SOAP call (`"http://ESCPlatform/xsd"`). Make sure to define your own username, password and MSISDN.

At the minimum, we need to define a username, password and a target MSISDN as parameters. Now to send a message that our application user had just typed, we merely set the "`messageString`" parameter to take its input from the HTTP-POST request sent by the user.

```
$result = $client->call('sendSMS', array(    'uName' => 'username',
        'uPin' => 'password',
        'MSISDN' => '0917xxxxxxx',
        'messageString' => $_POST['msg'],
        'Display' => '0',
        'udh' => '',
        'mwi' => '',
        'coding' => '0' ),
        "http://ESCPlatform/xsd");
```

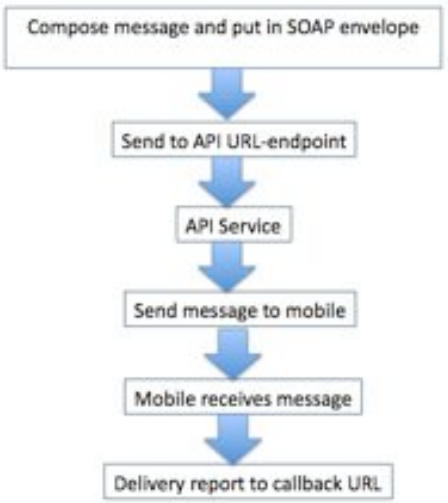
The other bits of code in the example perform error handling in case of failures. `$client->fault` is used when the SOAP service itself generates a error. The service will send SOAP envelope containing fault related elements and nodes. `$client->getError()`, on the other hand, is a method made available by NuSOAP for non standard failure modes such as timeouts. Finally, the code also processes the `$result` variable which contains the response of the web service to the request we made. In the example application, receiving a 201 code means that the message has successfully been sent.

API DESCRIPTION

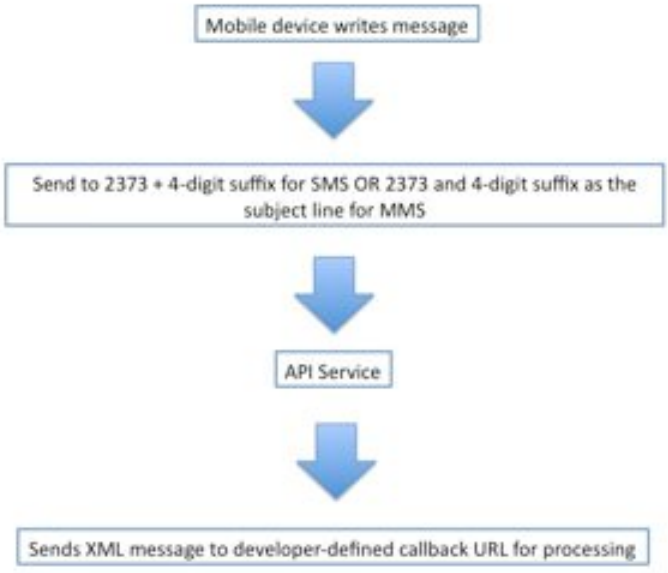
OVERVIEW

The diagrams below illustrate the general flow of the two main features of the API

Sending a message via the API:



Receiving a message via the API:



SENDING MESSAGES

sendSMS

Allows an API user to send an SMS message.

URL endpoint: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/

REST URL: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/sendSMS

<table>
<tr><th>Parameters</th><th>Description</th></tr>
<tr><td>uName</td><td>A valid username – same as the username used during registration. (Required)</td></tr>
<tr><td>uPin</td><td>The PIN that was issued to the user during the API registration process. (Required)</td></tr>
<tr><td>MSISDN</td><td>The target cellphone number intended to receive the SMS message. This number must have been defined during the registration process or added subsequently via the Globe Labs website. (Required)</td></tr>
<tr><td>messageString</td><td>The actual message string.</td></tr>
<tr><td>Display</td><td>Sets how the message is sent to a mobile device. By default, use 1 (Send to Phone) to send standard text messages. The table below lists various Display options. (Required)
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>0</td><td>Send Directly to Display</td></tr>
<tr><td>1</td><td>Send to Phone</td></tr>
<tr><td>2</td><td>Send to SIM</td></tr>
</table>
</td></tr>
<tr><td>udh</td><td>The user data header. This information is commonly used in delivering binary data such as in transmitting ringtones or in sending multi-part messages. Must be URL encoded (Can be blank). If you are not sending multi-part messages or binary data, it’s best to leave this field blank.</td></tr>
<tr><td>mwi</td><td>The message waiting indicator (Can be blank). For certain types of messages, you may want an additional icon / indicator to appear on the user’s phone. However, be careful in using this parameter, as you need to send another SMS to remove the icon. Codes 0 (zero) to 3 (three) show the icons while the others hide them.
<table>
<tr><th>MWI</th><th>Description</th></tr>
<tr><td>0</td><td>Voice Mail Icon Activate</td></tr>
<tr><td>1</td><td>Fax Icon Activate</td></tr>
<tr><td>2</td><td>Email Icon Activate</td></tr>
</table>
</td></tr>
</table>

	3	Other Activation
	4	Deactivate Voice Mail Icon
	5	Deactivate Fax Icon
	6	Deactivate Email Icon
	7	Deactivate Other Icon
`coding`	Sets the (text) coding scheme of the SMS message to be sent (Can be blank). The options are listed below. If you want to send standard text messages, use 0 (zero).	

Coding	Description
0	7-Bit
1	8-Bit
2	USC-2

After delivering the SOAP payload to the API interface, the API will respond with a SOAP response containing a return code (e.g. `<return>201</return>`). The return codes and their descriptions are as follows:

Return Code	Description
301	User is not allowed to access this service
302	User exceeded daily cap
303	Invalid message length
304	Maximum Number of simultaneous connections reached
305	Invalid login credentials
401	SMS sending failed
402	MMS sending failed
501	Invalid target MSISDN
502	Invalid display type
503	Invalid MWI
506	Badly formed XML in SOAP request

504	Invalid Coding
505	Empty value given in required argument
507	Argument given too large
201	SMS accepted for delivery
202	MMS Accepted for delivery

Using SOAP for Sending SMS messages

WSDL

WSDL or the Web-Services Description Language is an XML document used in describing the services available in a web service. This is useful in determining what services are available and which network protocols they bound to. You'll need the WSDL file for creating local proxies for your code to interact with. The WSDL file for the API, is accessible via the URL: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform?wsdl

Important notes:

- In using this service, all fields are necessary in the SOAP envelope, so make sure to include all arguments even if their contents are empty (blankable).
- Generally, the arguments are strings although the web service does not necessarily do strong type enforcement.
- It's important to include a XML namespace in the function call. The namespace is "`http://ESCPlatform/xsd/`". This should be defined as an xmlns attribute in the `sendSMS` soap function call / XML node (e.g. `<soapenv:Body><sendSMS xmlns="http://ESCPlatform/xsd/"> ... </sendSMS></soapenv:Body>`).
- If you are sending binary data over SMS, such a ringtones or operator logos, you need to use 8-bit encoding and url-escape the strings for the messageString and the udh.

Appendix 1 defines a sample SOAP envelope for the sendSMS method and the SOAP-based response.

Below is an example application written in the C# language and developed using the tools made available via the Mono project.

1) To begin, we first obtain the WSDL file and run the wsdl command on the file.

```
> wsdl http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform?wsdl
```

2) This should generate a file called Platform.cs. We need to compile this file to generate a library and the library needs to reference (-r:) the System.Web.Services library.

```
> mcs /target:library Platform.cs -r:System.Web.Services
```

3) This should generate a file called Platform.dll, which is a stub assembly for integration with other code.

4) The code below shows an application that allows us to send an SMS message from the command line

```
using System;
using System.Net;

class SendSMS {

        public static void Main(string [] args) {

                /* some of the paramters required to use to API */
                const string uName = "username";
                const string uPin = "password";
                const string MSISDN = "091xxxxxxx";
                const string Display = "0";
                const string udh = "";
                const string mwi = "";
                const string coding = "0";

                /* Platform is a class generated by the wdsl tool. It encapsulates the
web-service */
                Platform Service1 = new Platform();

                /* actual perform a remote procedure call via the interface */
                String result = Service1.sendSMS(uName, uPin, MSISDN, args[0], Display,
udh, mwi, coding);

                if (result == null) {
                        Console.WriteLine("[No response]");
                }
                else {
                        Console.WriteLine(suggestion);
                }
        }
}
```

5) To compile the code using Mono, we just need to reference our stub assembly.

```
> mcs /r:Platform.dll sendsms.cs
```

6) Accessing the service is as simple as:

```
> mono sendsms.exe "This is a test message via the Globe SMS API"
```

We can also use Java to send SMS messages via the API*. We'll need to download Axis (for SOAP messaging, (http://www.apache.org/dyn/closer.cgi/ws/axis/1_4) and we'll need to set the following libraries in your `CLASSPATH` to compile and compile/run the sample code below.

```
import java.net.URL;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class JavaClient {
```

```
        /*
         * Declare connection properties
         */
        private static final String ENDPOINT_URI =
"http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/";
        private static final String NAMESPACE = "http://ESCPlatform/xsd";
        private static final String uName = "username";
        private static final String uPin = "password";

        public static void main(String args[]) {

                /*
                 * Declare variables
                 */
                String MSISDN = "0915XXXXXXX";//The target cellphone number
                String messageString = "SMS from Globe API";//The actual message string.
                String display = "1";//Sets how the message is sent to a mobile device.
                String udh = "";//The user data header
                String mwi = "";//The message waiting indicator
                String coding = "0";//Sets the (text) coding scheme of the SMS message to
be sent

                try {
                        URL endpoint = new URL(ENDPOINT_URI);
                        Service service = new Service();
                        Call call = (Call) service.createCall();

                        call.setTargetEndpointAddress(endpoint);
                        call.setUseSOAPAction(true);
                        call.setOperationName(new javax.xml.namespace.QName(NAMESPACE,
                                        "sendSMS"));

                        call.addParameter("uName", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("uPin", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("MSISDN", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("messageString",
org.apache.axis.Constants.XSD_STRING, javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("display", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("udh", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("mwi", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.addParameter("coding", org.apache.axis.Constants.XSD_STRING,
javax.xml.rpc.ParameterMode.IN);
                        call.setReturnType(org.apache.axis.Constants.XSD_STRING);


                        String returnCode = (String) call.invoke(new java.lang.Object[] {
                                        uName, uPin, MSISDN, messageString, display, udh,
mwi,
                                        coding });//cast to String to get return code

                        System.out.println("RETURN CODE = " + returnCode);

                } catch (Exception e) {
                        System.err.println("Exception encountered :" + e.getStackTrace());
                }

        }
```

*Thanks to Melvin Dave P. Vivas, who graciously provided the code above.

Using REST to send SMS messages

If you plan on using the REST-based interfaces to send SMS messages, you simply have to call the appropriate URL. In the example below, we use the command line tool curl to send a simple SMS message.

```
> curl -d uName=username \
-d uPin=password \
-d MSISDN=0917XXXXXX \
-d messageString='Hellow World' \
-d Display=1 \
-d udh='' \
-d mwi='' \
-d coding='0' \
http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/sendSMS
```

You simply need to call the URL with the appropriate arguments as POST variables. If successful, you should see the following response:

```
<ns:sendSMSResponse
xmlns:ns="http://ESCPlatform/xsd"><ns:return>201</ns:return></ns:sendSMSResponse>
```

You can even call the API from a simple html form.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta name="generator" content="HTML Tidy for Linux/x86 (vers 11 February 2007),
see www.w3.org" />
        <title></title>
    </head>
    <body>
        <form method="post"
action="http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/sendSMS?">
            <table border="0">
                <tr>
                    <td>
                        uName
                    </td>
                    <td>
                        <input name="uName" />
                    </td>
                </tr>
                <tr>
                    <td>
                        uPin
                    </td>
                    <td>
                        <input name="uPin" />
                    </td>
                </tr>
                <tr>
                    <td>
                        MSISDN
                    </td>
                    <td>
                        <input name="MSISDN" />
                    </td>
                </tr>
                <tr>
                    <td>
                        Display
                    </td>
                    <td>
                        <input name="Display" value="1" />
                    </td>
                </tr>
```

```
                <tr>
                    <td>
                        udh
                    </td>
                    <td>
                        <input name="udh" />
                    </td>
                </tr>
                <tr>
                    <td>
                        mwi
                    </td>
                    <td>
                        <input name="mwi" />
                    </td>
                </tr>
                <tr>
                    <td>
                        coding
                    </td>
                    <td>
                        <input name="coding" value="0" />
                    </td>
                </tr>
                <tr>
                    <td>
                        Message
                    </td>
                    <td>
                        <textarea name="messageString">
</textarea>
                    </td>
                </tr>
                <tr>
                    <td colspan="2">
                        <input type="submit" />
                    </td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

The example above will output the same XML document containing the response code.

sendMMS

Allows an API user to send a MMS message. The parameters used are similar to the sendSMS function but this SOAP call uses SMIL information instead of text to define content.

URL endpoint: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/

REST URL: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/sendMMS

Parameters	Description
uName	A valid username – same as the username used during registration. **(Required)**
uPin	The PIN that was issued to the user during the API registration process. **(Required)**
MSISDN	The target cellphone number intended to receive the SMS message. This number must have been defined during the registration process or added subsequently

	via the Globe Labs website. **(Required)**
`subject`	The subject matter of the message. **(Required)**
`smil`	A properly formed SMIL XML document. SMIL stands for "Synchronized Multimedia Integration Language" and is a w3c standard for defining how content should appear on a mobile phone. More information about developing using SMIL and details about the document specification can be found at http://www.w3.org/AudioVideo/. See Appendix 2 for some sample XML to get you started.

Similar to the SMS API, after delivering the SOAP payload, the API will respond with a SOAP response containing a return code. The return codes are as follows:

Return Code	Description
301	User is not allowed to access this service
302	User exceeded daily cap
303	Invalid message length
304	Maximum Number of simultaneous connections reached
305	Invalid login credentials
401	SMS sending failed
402	MMS sending failed
501	Invalid target MSISDN
502	Invalid display type
503	Invalid MWI
504	Invalid Coding
505	Empty value given in required argument
506	Badly formed XML in SOAP request
507	Argument too large

509	Malformed SMIL
202	MMS Accepted for delivery

Using SOAP for Sending SMS messages

WSDL

WSDL or the Web-Services Description Language is an XML document used in describing the services available in a web service. This is useful in determining what services are available and which network protocols they bound to. You'll need the WSDL file for creating local proxies for your code to interact with. The WSDL file for the API, is accessible via the URL: http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform?wsdl

Important notes:

- In using this service, all fields are necessary in the SOAP envelope, so make sure to include all arguments even if their contents are empty (blankable).
- Generally, the arguments are strings although the web service does not necessarily do strong type enforcement.
- It's important to include a XML namespace in the function call. The namespace is "`http://ESCPlatform/xsd/`". This should be defined as an xmlns attribute in the `sendSMS` soap function call / XML node (e.g. `<soapenv:Body><sendSMS xmlns="http://ESCPlatform/xsd/"> ... </sendSMS></soapenv:Body>`).

Appendix 2 defines a sample SOAP envelope for the sendMMS method including an example SMIL argument.

The example below extends the previous SMS example. But instead of providing an SMS argument, the application expects an SMIL string. The console application returns an integer code upon completing the request.

```
using System;
using System.Net;

class SendMMS {

        public static void Main(string [] args) {

                const string uName = "user";
                const string uPin = "password";
                const string MSISDN = "0917xxxxxxx";
                const string Subject = "Test Message";

                Platform Service1 = new Platform();
                 /* uncomment this code block if you need http proxy support
                NetworkCredential myCred = new NetworkCredential("username", "password",
"domain");
                WebProxy proxyObject = new WebProxy("http://proxy.address:8080/");

                proxyObject.Credentials = myCred;
                Service1.Proxy = proxyObject;
                */
```

```
                String suggestion = Service1.sendMMS(uName, uPin, MSISDN, Subject,
args[0]);

                if (suggestion == null) {
                        Console.WriteLine("[No response]");
                }
                else {
                        Console.WriteLine(suggestion);
                }
        }
}
```

Compiling the code using mono is also similar

```
> mcs /r:Platform.dll sendmms.cs
```

And then using the service is as simple as:

```
> mono sendmms.exe "<smil>*smil</smil>"
```

To get you started, below is an example SMIL file that contains a Globe logo and some text.

```
<smil>
  <head>
    <layout>
      <root-layout height='96' width='122' />
      <region height='67%' fit='meet' id='Image' width='100%' left='0%' top='0%' />
      <region height='33%' fit='scroll' id='Text' width='100%' left='0%' top='67%' />
    </layout>
  </head>
  <body>
    <par dur='8000ms'>
      <img src='http://freeformsoftware.org/demo/resource/globelogo.gif' region='Image'
/>
      <text src='http://freeformsoftware.org/demo/resource/helloworld.txt' region='Text'
/>
    </par>
  </body>
</smil>
```

Using REST to send MMS Messages

Sending SMS messages via REST is done just like sending a MMS message. You just need to call the prescribed URL:

```
http://iplaypen.globelabs.com.ph:1881/axis2/services/Platform/sendMMS
```

with the appropriate parameters included as POST variables. Use the example HTML for to send a MMS message. Just add the SMIL example above in the space provided.

XML-BASED DELIVERY NOTIFICATION

During the API registration process, you were asked to provide a URL. This URL is used by the API service as a callback address to send messages to. Specifically, the API service sends Delivery Notification messages to this URL using standard HTTP-POST requests. These requests contain structured meta-data about messages in an XML format. Currently, only the sendSMS service have Delivery Notifications in place.

The XML format of the Delivery Report message is a follows:

```
<?xml version="1.0" encoding="utf-8"?>
<message>
  <param>
    <name>messageType</name>
    <value>SMS-NOTIFICATION</value>
  </param>
  <param>
    <name>source</name>
    <value>2373</value>
  </param>
  <param>
    <name>type</name>
    <value>1</value>
  </param>
  <param>
    <name>msg</name>
    <value>Message for 09178101512, with identification
080623135320 has been delivered on 2008-06-23 at
13:53:24.</value>
  </param>
  <param>
    <name>target</name>
    <value>09178101512</value>
  </param>
</message>
```

This XML document contains just a series of parameter (param) nodes which you can parse and then iterate though. Each parameter should contain a corresponding `<name>` and `<value>` node. These parameters are as follows:

<table>
<tr><th>Parameters</th><th>Description</th></tr>
<tr><td>messageType</td><td>Typically this will always be “SMS-NOTIFICATION”</td></tr>
<tr><td>source</td><td>The source / sending MSISDN initiating the message. For SOAP-based messages this will be 2373 + your assigned 4-digit suffix code.</td></tr>
<tr><td>type</td><td>The actual delivery-status report result. These are integer codes. Their descriptions are follows:
<table>
<tr><th>Status Code</th><th>Definition</th></tr>
<tr><td>1</td><td>Delivery Success</td></tr>
<tr><td>2</td><td>Delivery Failure</td></tr>
<tr><td>4</td><td>Message Buffered</td></tr>
<tr><td>8</td><td>SMSC Submit</td></tr>
<tr><td>16</td><td>SMSC Reject</td></tr>
</table>
</td></tr>
<tr><td>msg</td><td>Normally, this should just contain the details of the message sent. In the case of the sendSMS SOAP interface, it would normally read "Message for 091XXXXXXX, with identification 080716100622 has been delivered on 2008-07-16 at 10:06:26."</td></tr>
</table>

target	the target / receiving MSISDN or cellular number

In order to process the XML, you'll need to be able to access the raw HTTP POST request itself, extract the XML data and finally process the XML.

In the example below, we use PHP and mySQL to store the notification messages sent by the API service. First we create a simple mySQL table to store the message.

```
CREATE TABLE IF NOT EXISTS `notifications` (
  `messageType` varchar(255) default NULL,
  `source` varchar(255) default NULL,
  `type` varchar(255) default NULL,
  `msg` varchar(255) default NULL,
  `target` varchar(255) default NULL
)
```

The sample PHP code processes the POST request by extracting the raw POST information and then parsing the XML to extract data. After parsing, the data is stored in the MySQL database. If GET request was received, then the script queries the database and displays all of the previous notification messages.

```
<?php

/* application paramters */
$dbname = "dbname";
$user = "user";
$pass = "password";

/* connect and  select database */
$db = mysql_connect  ("localhost", $user, $pass);
if (!$db) {
  die (mysql_error());
}
mysql_select_db  ( $dbname, $db);

/* if we get a post request */
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
  /* read raw POST data */
  $in = file_get_contents ("php://input");

  /* list is just a container we'll use to store the data */
  $list = array();

  /* parse the input using the DOM */
  $doc = new DOMDocument();
  $doc->loadXML($in);
  $params = $doc->getElementsByTagName( "param" );
  foreach( $params as $param )
  {
    $names = $param->getElementsByTagName( "name" );
    $name = $names->item(0)->nodeValue;

    $values = $param->getElementsByTagName( "value" );
    $value = $values->item(0)->nodeValue;

    $list[$name] = "'".mysql_escape_string($value)."'";
  }

  $sql = "insert into notifications(".join(",", array_keys($list)).") values (" .
join("," , $list) . ")";
  mysql_query($sql, $db); }
else {
  /* if it's a GET request then just display the notification messages */
  header( 'refresh: 30;');
  print "<html><head></head><body>";
  $sql = "select * from notifications";
```

```
  $result = mysql_query($sql, $db);
  print "<table border=\"1\">";
  while ($arr = mysql_fetch_array($result, MYSQL_ASSOC)) {
    print "<tr>";
    foreach ($arr as $val) {
      print "<td>".htmlspecialchars($val)."</td>";
    }
    print "</tr>";
  }
  print "</table>";
  print "</body>";
}
?>
```

SMS CLIENT INTERFACE - USING 2373 + 4-DIGIT SUFFIX CODE

The API not only allows users to send SMS and MMS messages, but also provides a mechanism by which developers can receive messages.

For applications using SMS, this is done via the use of the 2373 short-code prefix number and a 4-digit suffix code that is unique for every registered API developer. For example, if your assigned suffix / access code is 1435, a registered Globe subscriber will be able to access your SMS application/service via 23721435.

Valid cellular numbers (those listed/added via the registration process), can send standard 160 character or multi-part SMS messages to your assigned short-code. Multi-part messages will be split accordingly into individual (XML) messages.

However for MMS applications, instead of using the 4-digit suffix code when sending the message, you instead use only the short-code (2373) and use the 4-digit suffix code in the subject line of the MMS message.

Once the API service receives the SMS message, it sends an HTTP-POST request to your registered callback URL (The one you set during the registration process). Again, you can alter the default port via the callback URL definition. (e.g. http://my.url.com:8080/).

Similar to Delivery Notifications, in order to process the XML, you'll need to be able to access the raw HTTP POST request and parse it accordingly.

The format of the XML data is slightly different between SMS and MMS messages. For SMS, it is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<message>
  <param>
    <name>messageType</name>
    <value>SMS</value>
  </param>
  <param>
    <name>id</name>
    <value>xxxxxxxxxxxxxx</value>
  </param>
  <param>
    <name>source</name>
    <value>xxxxxxxxxxx</value>
  </param>
  <param>
    <name>target</name>
    <value>xxxxxxxxxxxxx</value>
```

```
  </param>
  <param>
    <name>msg</name>
    <value>xxxxxxxxxxxxx</value>
  </param>
  <param>
    <name>udh</name>
    <value></value>
  </param>
</message>
```

For MMS, it is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<message>
  <param>
    <name>messageType</name>
    <value>MMS</value>
  </param>
  <param>
    <name>subject</name>
    <value>subject123</value>
  </param>
  <param>
    <name>source</name>
    <value>123</value>
  </param>
  <param>
    <name>target</name>
    <value>123</value>
  </param>
  <param>
    <name>file</name>
    <value>
      <file>http://localhost:1234/testing.jpg</file>
      <file>http://localhost:1234/testing.txt</file>
    </value>
  </param>
</message>
```

Below are the parameter descriptions:

Parameters	Description
Common for SMS and MMS	
source	The MSISDN / cellular number sending the message.
target	The shortcode + 4-digit suffix intended to receive the message
messageType	the type of message (either SMS or MMS)
SMS Only	
msg	the actual message received
id	a unique message identifier
udh	the user data header, useful in re-assembling multi-part messages
MMS Only	
subject	the subject used by the sender of the message
file	this data point can contain multiple <file> nodes with each node describing the

	location of a file. Normally, this will be in a URL format using the HTTP protocol.

Finally, you should note that files sent via MMS have a maximum lifetime of 30 minutes in the system. After this prescribed period, the data will be purged from the API service, so we encourage developers to design their applications with this in mind.

Below is an example of how we can process a message sent via the 2373 + suffix interface. Basically, if the script is called via the API to receive / process a message (through a POST request), the XML is parsed and the results are saved to a mysql database. However, if we access the page via a GET request, we should be able to see the various messages sent to us via the API.

First we create a table to store the data we are parsing. The table definition script is below:

```
CREATE TABLE IF NOT EXISTS `incoming` (
  `id` varchar(255) NOT NULL default '',
  `source` varchar(255) default NULL,
  `target` varchar(255) default NULL,
  `msg` varchar(255) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM
```

And the actual PHP script to demonstrate parsing is listed below:

```
<?php

/* this just sets the database paramters */
 $dbname = "sms";
$user = "username";
$pass = "password";

 /* connect to the database using the paramters above */
$db = mysql_connect  ("localhost", $user, $pass);
if (!$db) {
  die (mysql_error());
}

/* select the database to use */
mysql_select_db  ( $dbname, $db);

/* change behavior depending on the type of request. If we receive a POST request, then
parse the XML document embedded within and save to the database. We can add additional
validation logic later */
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
  /* the php://input file is a PHP built-in that returns raw HTTP-POST data */
  $in = file_get_contents ("php://input");

  /* temporary storage */
  $list = array();

  /* available in PHP5 is the DOMDocument class which we can use to parse XML */
  $doc = new DOMDocument();
  $doc->loadXML($in);

  /* here we just iterate through the XML and store the data points into the $list array
*/
  $params = $doc->getElementsByTagName( "param" );
  foreach( $params as $param )
  {
    $names = $param->getElementsByTagName( "name" );
    $name = $names->item(0)->nodeValue;

    $values = $param->getElementsByTagName( "value" );
```

```
    $value = $values->item(0)->nodeValue;

    $list[$name] = "'".mysql_escape_string($value)."'";
  }

  /*
  $sql = "insert into incoming(".join(",", array_keys($list)).") values (" . join("," ,
$list) . ")";
  mysql_query($sql, $db);
}
else {
  /* here we just display the sent messages. The header function just loads directs the
browser to reload the page every 5 seconds*/
  header( 'refresh: 5;');
  print "<html><head></head><body>";
  $sql = "select * from incoming order by id";
  $result = mysql_query($sql, $db);
  print "<table border=\"1\">";
  while ($arr = mysql_fetch_array($result, MYSQL_ASSOC)) {
    print "<tr>";
    foreach ($arr as $val) {
      print "<td>".htmlspecialchars($val)."</td>";
    }
    print "</tr>";
  }
  print "</table>";
  print "</body>";
?>
```

Similar to processing notifications, the script processes RAW POST data by parsing the XML data and then adding the data to the database. Also, with a GET request, the script just obtains the data and displays the data on-screen.

USEFUL TOOLS

Here are some Open source tools and libraries to help you get started in using the SMS/MMS API

- Java: Axis -- http://ws.apache.org/axis/
- Perl: SOAP::Lite -- http://www.soaplite.com/ http://cookbook.soaplite.com/
- Python: Soap.py & ZSI -- http://pywebsvcs.sourceforge.net/
- PHP: NuSOAP and PHP5 Support -- http://sourceforge.net/projects/nusoap/ , http://www.php.net/soap
- C/C++: WASP Server Lite, EasySoap++, gSOAP -- http://www.cs.fsu.edu/~engelen/soap.html, http://sourceforge.net/projects/easysoap/,
- Ruby: SOAP4R -- http://dev.ctor.org/soap4rp

LINKS

SOAP envelope standard -- http://www.w3schools.com/soap/soap_envelope.asp

SMIL standard -- http://www.w3.org/AudioVideo/

Globe Labs - http://www.globelabs.com.ph/

APPENDICES

Appendix 1. Sample SOAP Messages for the sendSMS web service.

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

<SOAP-ENV:Envelope xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/' SOAP-
ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' xmlns:SOAP-
ENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <SOAP-ENV:Body>
    <ns3363:sendSMS xmlns:ns3363='http://ESCPlatform/xsd'>
      <uName xsi:type='xsd:string'>username</uName>
      <uPin xsi:type='xsd:string'>password</uPin>
      <MSISDN xsi:type='xsd:string'>0917XXXXXXX</MSISDN>
      <messageString xsi:type='xsd:string'>test message</messageString>
      <Display xsi:type='xsd:string'>1</Display>
      <udh xsi:type='xsd:string'></udh>
      <mwi xsi:type='xsd:string'></mwi>
      <coding xsi:type='xsd:string'>0</coding>
    </ns3363:sendSMS>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Appendix 2. Sample SOAP Messages for the sendMMS web service along with sample SMIL.

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

<SOAP-ENV:Envelope xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/' SOAP-
ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' xmlns:SOAP-
ENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <SOAP-ENV:Body>
    <ns3499:sendMMS xmlns:ns3499='http://ESCPlatform/xsd'>
      <uName xsi:type='xsd:string'>username</uName>
      <uPin xsi:type='xsd:string'>password</uPin>
      <MSISDN xsi:type='xsd:string'>0917XXXXXX</MSISDN>
      <Subject xsi:type='xsd:string'>test message</Subject>
      <SMIL xsi:type='xsd:string'>
        <smil>
          <head>
            <layout>
              <root-layout height='96' width='122' />
              <region height='67%' fit='meet' id='Image' width='100%' left='0%' top='0%'
/>
              <region height='33%' fit='scroll' id='Text' width='100%' left='0%'
top='67%' />
            </layout>
          </head>
          <body>
            <par dur='8000ms'>
              <img src='http://freeformsoftware.org/demo/resource/globelogo.gif'
region='Image' />
              <text src='http://freeformsoftware.org/demo/resource/helloworld.txt'
region='Text' />
            </par>
          </body>
        </smil>
      </SMIL>
    </ns3499:sendMMS>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Appendix 3. Sample SOAP response message.

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

<soapenv:Envelope xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'>
  <soapenv:Body>
    <example1:sendMMS xmlns:example1='http://ESCPlatform/xsd'>
      <example1:text>202</example1:text>
    </example1:sendMMS>
  </soapenv:Body>
</soapenv:Envelope>
```