m:science
every message counts

COMPLETE GUIDE TO INTEGRATING SMS

SMS Web Service

Developer's Guide

Website: www.m-science.com
Telephone: 01202 241120
Email: enquiries@m-science.com

Introduction

Product Overview

The SMS Web Service is a unique .NET enabled software product that enables organisations to integrate two-way SMS messaging functionality into their own websites, applications or web services.

As the product is .NET enabled and access to the web service uses standard Internet protocols, it is vendor, platform and language independent enabling multiple methods of connection as shown in Figure 1 below.

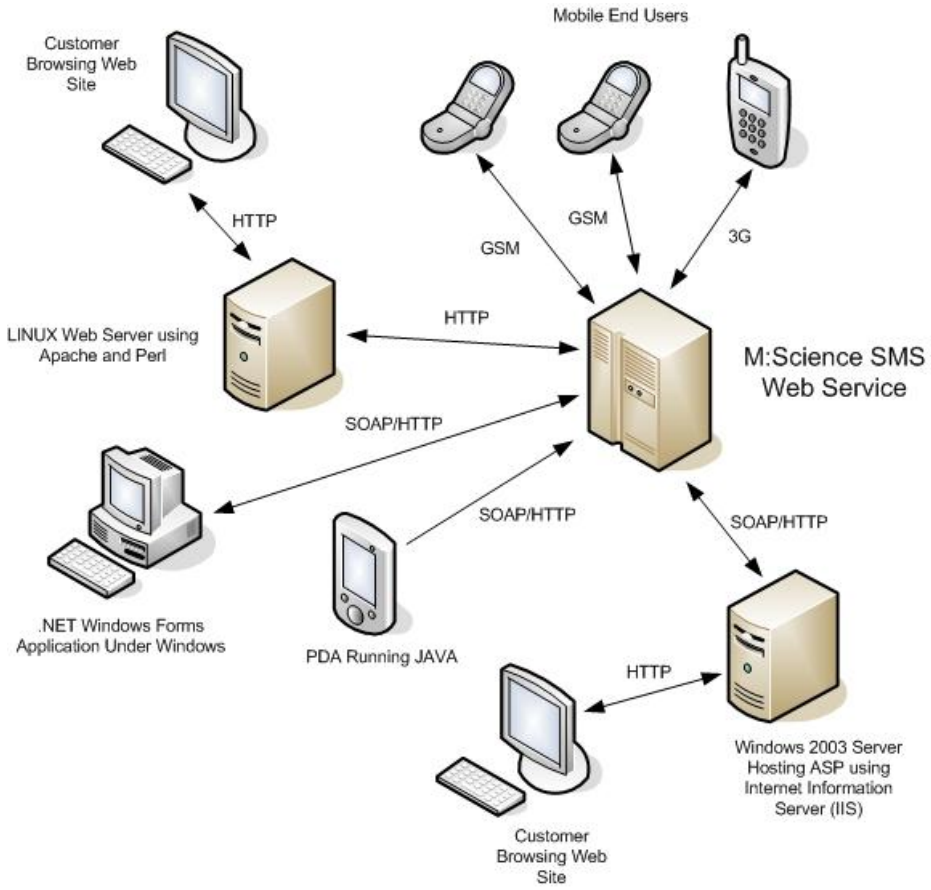

Figure 1 – Connection Methods

To begin using the SMS Web Service, the first step is to set up an online M:Science account. Go to www.m-science.com and select 'New Customer' and complete the registration form to register an account. Access to the SMS Web Service must be requested through M:Science sales (Contact details on website). A client can be anything that accepts an HTTP PUSH, such as an ASP script, a cgi script or another .NET web service. Secure communications via HTTPS are also supported.

The account may then be configured for handling inbound messages depending on the method to be used i.e. inbound messages to SMS product, Web PUSH or Web Service. It is possible to specify a different inbound method for each rented inbound number.

Product Components

Web Service

Hosted by M:Science, this .NET web service allows external users to communicate with the service via HTTP or SOAP to send, receive and monitor text messages or configure asynchronous inbound routing of text messages. This is done using SOAP method calls or HTTP GET or PUSH operations onto the Web service SMS Web Service server.

Examples of how to incorporate SOAP or HTTP using a number of different programming languages is provided in the 'Programming Language Specifics' section of this guide.

Each method provides basic security for encrypting account identifiers and passwords, but to achieve total security a secure sockets layer (SSL) should be employed.

The web service is available on two addresses. These are smswebservice1.m-science.com and smswebservice2.m-science.com. Software should be coded to use the backup address if the primary is unavailable. To utilise secure sockets communications simply prefix the address with 'https://' instead of 'http://'.

Web Push Service

This service asynchronously routes incoming messages via HTTP PUSH to one or more registered URLs. A web address must be provided which interprets the parameter list specified for 'ReceiveInboundMessage'.

Sending Messages

Sending One or More Messages

Messages are sent by making calls onto ‘SendMessages’, which will allow up to five messages to be supplied. A message ID is then returned for each message which can be used to track the progress of the message.

Tracking Messages

The status of a message can be obtained by making a call onto ‘GetMessageStatus’, which will accept up to ten message IDs and then return a status for each one. See the ‘Function References’ section in this guide for full details of the message states which can be returned.

To fully track a message, it is recommended to request a delivery receipt when the original message is sent. If the client needs to interpret delivery receipts directly, the product message ID code can also be supplied with the original message. This is sent back with any delivery receipt and can hence be used to tie the two together.

Receiving Messages

Polling for Messages

If a selected inbound number has been configured to return messages to the web service, the messages can be retrieved by making calls onto 'GetMessages'. This will return up to the next five messages in the queue. To receive more messages, these must first be acknowledged. This can be done by making a call onto 'AckMessages', supplying the list of identity codes given back in 'GetMessages'.

Asynchronous Message PUSHing

It is possible to configure the M:Science SMS Web Service server to asynchronously 'PUSH' messages to a web address, calling the local software immediately the message arrives. The user must first contact M:Science at enquires@m-science.com to activate their account for this purpose and then must register one or more URLs using 'RegisterForInbound'. The web server code behind the URLs must be capable of accepting an HTTP PUSH (the same as a form submit) using the parameters specified in the 'Client Interface section of this guide.

To stop message pushing, either a specific URL can be unregistered using 'UnregisterInboundAddress' or all URLs unregistered for an account by calling 'ResetInboundCallbacks'.

The client can check to see if the web service is having problems pushing messages by calling onto 'GetURLStatus' and stating the URL.

Programming Language Specifics

Note: There are downloadable ZIP files from our website containing sample projects and demonstrating the key issues described here.

Encryption

To prevent the AccountID and Password from being in readable form, they can be encrypted. To send in an encrypted username or password, simply take each character value for strings representing both and add one. This is sufficient to render known words as garbled text. If the encrypted flag is set on a method call, the reverse of this is done inside the server software.

An example function might be:

```
private void EncryptCredentials(string accountID, string password, out string encAccountID, out string encPassword)
{
// First encrypt the account id
        char[] procArr = accountID.ToCharArray();
        char[] output = new char[procArr.Length];
        for(int i = 0; i < accountID.Length; i++)
                output[i]= (char)(procArr[i]+1);
        encAccountID = new string(output);
        // Encrypt the password
        StringBuilder sb2 = new StringBuilder();
        procArr = password.ToCharArray();
        output = new char[procArr.Length];
        for(int i = 0; i < password.Length; i++)
                output[i]= (char)(procArr[i]+1);
        encPassword = new string(output);
}
```

.NET

Microsoft's primary development environment for .NET is Visual Studio 2003. This dramatically simplifies accessing Web Services. Alternatively the .NET SDK may be used, but that is outside of the scope of this guide.

There are many languages available for the .NET platform but in this instance we shall consider the more popular languages of C#, VB and C++.

Web Services such as the M:Science SMS Web Service support an interrogation system for detailing the available interfaces and methods.

Visual Studio automatically does this for you and creates an object exposing all of the methods of the service as described in the Function Reference section of this guide. All the user then has to do is to instantiate the object and make calls onto it.

To illustrate this procedure:

1. Make a blank Windows forms project in any of the languages mentioned above.
2. Next navigate to the solution explorer, right click on references and select 'Add Web Reference'.
3. At the prompt, enter the address of the M:Science SMS Web Service as follows: http://smswebservice1.m-science.com. Note: There is a backup address at http://smswebservice2.m-science.com. Software can be programmed to use this as an optional redundant failover. Use the prefix 'https://' if you require SSL encryption (only available on the primary server).
4. Provided an internet connection is available to M:Science, the rest of the process will be done for you.

Note: The same procedure is used for writing a Windows Forms application or for a Web Application.

Example Calls

The following are examples of calling an imported Web Service from .NET languages:

Note: Exception handling should be used in all cases but has been omitted to save space.

C#

```
MScienceSMSWebService ws = new MScienceSMSWebService();
string[] tag = new string[1];
tag[0] = "123";              // Tag to identify the source application
string[] address = new string[1];
address[0] = "123";          // Set this to any inbound number which you may wish the recipient to reply to
int[] receipt = new int[1];
receipt[0] = 0;   // Set this to 1 if you want a delivery receipt
int[] messageID = new int[1];
messageID[0] = 0;            // Can be set to a unique message ID. This is sent back with delivery recipts.
string[] destination = new string[1];
destination[0] = textBoxDestination.Text;
string[] message = new string[1];
message[0] = textBoxMessage.Text;
string result = ws.SendMessages(false, textBoxUserID.Text, textBoxPassword.Text, tag, address, receipt,
messageID, destination, message);
System.Diagnostics.Trace.WriteLine(result);
```

VB

```
Dim ws As New Client.localhost.MScienceSMSWebService

Dim tag(1) As String

tag(0) = "123"                '// Tag to identify the source application

Dim address(1) As String

address(0) = "123"          '// Set this to any inbound number which you may wish the recipient to reply to

Dim receipt(1) As Integer

receipt(0) = 0               '// Set this to 1 if you want a delivery receipt

Dim messageID(1) As Integer

messageID(0) = 0            '// Can be set to a unique message ID. This is sent back with delivery recipts.

Dim destination(1) As String

destination(0) = TextBoxDestination.Text

Dim message(1) As String

message(0) = TextBoxMessage.Text

Dim result As String

result = ws.SendMessages(False, TextBoxUserID.Text, TextBoxPassword.Text, tag, address, receipt,
messageID, destination, message)
```

C++

```
localhost::MScienceSMSWebService *pWS = new localhost::MScienceSMSWebService();

if(pWS)

{

System::String *tag[] = __gc new System::String*[1];

tag[0] = S"123";

System::String *address[] = __gc new System::String*[1];

address[0] = S"123";

System::Int32 receipt[] = __gc new System::Int32[1];

receipt[0] = 0;

System::Int32 messageID[] = __gc new System::Int32[1];

messageID[0] = 0;

System::String *destination[] = __gc new System::String*[1];

destination[0] = textBoxDestination->Text;

System::String *message[] = __gc new System::String*[1];

message[0] = textBoxMessage->Text;

System::String *result = pWS->SendMessages(false, textBoxUserID->Text, textBoxPassword->Text, tag,
address, receipt, messageID, destination, message);

MessageBox::Show(result);

System::Diagnostics::Trace::WriteLine(result);

}
```

Sun Java

Using Java 1.4.2 the .NET package provides a couple of useful classes which can be used. The simplest is to use the URL object to perform an HTTP GET. String manipulation can then be used to check the results. For example:

```
// Format the parameters for the URL
String parameters;
parameters = "encrypt=False";
parameters = parameters + "&accountID=" + textUserID.getText();
parameters = parameters + "&password=" + new String(textPassword.getPassword());
parameters = parameters + "&sourceTag=123";
parameters = parameters + "&sourceAddress=123";
parameters = parameters + "&deliveryReceipt=0";
parameters = parameters + "&productMessageID=0";
parameters = parameters + "&destination=" + textDestination.getText();
parameters = parameters + "&message=" + textMessage.getText();
// Note the parameter sting must not contain any spaces or the server will throw a 400 error
// So, process the parameters string to convert spaces to %20
String procParameters = "";
for(int i=0; i<parameters.length(); i++)
{
if(parameters.charAt(i) == ' ')
procParameters += "%20";
else
procParameters += parameters.charAt(i);
}
String urlString = "http://smswebservice1..m-science.com/
MScienceSMSWebService.asmx/SendMessages?";
urlString = urlString + procParameters;
URL url  = new URL(urlString);
BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
```

in.readLine();

If using this method, it is important to ensure that the string does not contain any non HTTP compliant characters. For example any spaces should be replaced with %20 and '+' should be replaced with %2b.

The more complex method is to use the URL, URLConnection and HttpURLConnection objects to connect onto the web service and to send the XML envelope for the command as specified in the 'Function References' section of this manual. See the sample code for the complete version of this listing:

```
URL url = new URL(SOAPUrl);

URLConnection connection = url.openConnection();

HttpURLConnection httpConn = (HttpURLConnection) connection;

FileInputStream fin = new FileInputStream(xmlFile2Send);

ByteArrayOutputStream bout = new ByteArrayOutputStream();

copy(fin,bout);

fin.close();

byte[] b = bout.toByteArray();

httpConn.setRequestProperty( "Content-Length", String.valueOf( b.length ) );

httpConn.setRequestProperty("Content-Type","text/xml; charset=utf-8");

httpConn.setRequestProperty("SOAPAction",SOAPAction);

httpConn.setRequestMethod( "POST" );

httpConn.setDoOutput(true);

httpConn.setDoInput(true);

OutputStream out = httpConn.getOutputStream();

out.write( b );

out.close();

InputStreamReader isr = new InputStreamReader(httpConn.getInputStream());

BufferedReader in = new BufferedReader(isr);

String inputLine;

while ((inputLine = in.readLine()) != null)

System.out.println(inputLine);

in.close();
```

XML

An example XML file might be:

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

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<SendMessages xmlns="http://www.m-science.com/MScienceSMSWebService/">

<encrypt>false</encrypt>

<accountID>1</accountID>

<password>password</password>

<sourceTag>

<string>123</string>

</sourceTag>

<sourceAddress>

<string></string>

</sourceAddress>

<deliveryReceipt>

<int>1</int>

</deliveryReceipt>

<productMessageID>

<int>0</int>

</productMessageID>

<destination>

<string>+447736239613</string>

</destination>

<message>

<string>XMLtest</string>

</message>
```

```
</SendMessages>

</soap:Body>

</soap:Envelope>
```

ASP

The Microsoft.XMLHTTP object can be used to perform HTTP GET or PUSH operations on the Web Service. The formatting for the HTTP GET function should be a question mark between the function address and the parameter list. Each parameter separated from the next using a question mark. The example used here used VB script embedded is the ASP to perform an HTTP GET on the web service. SOAP could be used by PUSHing XML envelopes as with the Java example.

```
<%@ Language=VBScript %>

<HTML>

<HEAD>

<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">

</HEAD>

<BODY>

<%

'VBScript goes in here

Dim xml

Dim Parameters

Parameters = "encrypt=" & Request.Form.Item("encrypt")

Parameters = Parameters & "&accountID=" & Request.Form.Item("accountID")

Parameters = Parameters & "&password=" & Server.UrlEncode(Request.Form.Item("password"))

Parameters = Parameters & "&sourceTag=" & Request.Form.Item("sourceTag")

Parameters = Parameters & "&sourceAddress=" & Request.Form.Item("sourceAddress")

Parameters = Parameters & "&deliveryReceipt=" & Request.Form.Item("deliveryReceipt")

Parameters = Parameters & "&productMessageID=" & Request.Form.Item("productMessageID")

Parameters = Parameters & "&destination=" & Server.UrlEncode(Request.Form.Item("destination"))

Parameters = Parameters & "&message=" & Server.UrlEncode(Request.Form.Item("message"))

Set xml = Server.CreateObject("Microsoft.XMLHTTP")

xml.open "GET", "http://smswebservice1.m-science.com/
MScienceSMSWebService.asmx/SendMessages?" & Parameters, false

xml.send

Dim WhereAt
```

```
WhereAt = InStr(1, xml.responseText, "OK-", 1)

If WhereAt = 0 then

Response.Write "Failed to send message: " & xml.responseText

Else

Response.Write "Successfully sent message: " & xml.responseText

End If

Set xml = nothing

%>

<P> </P>

<p><a href = "index.htm">Return to Index</a></p>

</BODY>

</HTML>
```

VB6

Very much the same procedure as for ASP, the Microsoft.XMLHTTP can be used. The same principal applies with respect to using XML envelopes to make SOAP calls.

```
Parameters = "encrypt=False"

Parameters = Parameters & "&accountID=" + TextUserID.Text

Parameters = Parameters & "&password=" + TextPassword.Text

Parameters = Parameters & "&sourceTag=123"

Parameters = Parameters & "&sourceAddress=123"

Parameters = Parameters & "&deliveryReceipt=0"

Parameters = Parameters & "&productMessageID=0"

Parameters = Parameters & "&destination=" & TextDestination.Text

Parameters = Parameters & "&message=" & TextMessage.Text


Set xml = CreateObject("Microsoft.XMLHTTP")

xml.open "GET", "http://smswebservice1.m-science.com/
MScienceSMSWebService.asmx/SendMessages?" & Parameters, False

xml.send

Dim WhereAt

WhereAt = InStr(1, xml.responseText, "OK-", 1)

If WhereAt = 0 Then

   MsgBox "Failed to send message: " & xml.responseText

Else

   MsgBox "Successfully sent message: " & xml.responseText

End If

Set xml = Nothing
```

HTML

Straight HTML is not the ideal protocol for accessing the web service as the end user will be presented with the direct output of the web service which is not in an easily readable format. However, it is ideal for a quick test with very little required in the way of dependencies:

```
<html>

<head>

<meta http-equiv="Content-Language" content="en-gb">

<meta name="GENERATOR" content="Microsoft FrontPage 5.0">

<meta name="ProgId" content="FrontPage.Editor.Document">

<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">

<title>UserID</title>

</head>

<body>

<form method="POST" action="http://smswebservice1.m-science.com/
MScienceSMSWebService.asmx/SendMessages">

<p>

 UserID</p>

 <p><input type="text" name="accountID" size="20" tabindex="1"></p>

 <p>Password</p>

 <p><input type="password" name="password" size="20" tabindex="2"></p>

 <p>Destination</p>

 <p><input type="text" name="destination" size="20" tabindex="3"></p>

 <p>Message Text</p>

 <p><textarea rows="2" name="message" cols="20" tabindex="4"></textarea></p>

 <p><input type="submit" value="Submit" name="B1" tabindex="5"><input type="reset" value="Reset"
name="B2"></p>

<input type="hidden" name="deliveryReceipt" value="0">

<input type="hidden" name="encrypt" value="false">

<input type="hidden" name="productMessageID" value="0">

<input type="hidden" name="sourceAddress" value="123">
```

```
<input type="hidden" name="sourceTag" value="123">

</form>

</body>
```

Python

Python is a simple and yet powerful open source language freely available for most platforms. It has an easy direct interpreter mode for quick coding tests and extensive libraries for more complex operations. It can be downloaded for free from http://www.python.org. It can also be used in much the same way as Perl with Apache web server (http://www.apache.org) by using the additional plugin Mod_Python (http://httpd.apache.org/modules/python-download.cgi).

The same principals apply as with the other languages for accessing the Web Service. Make an HTTP GET or PUSH (optionally with the SOAP envelope) and use string manipulation to interpret the response.

An example would be (cut and past this into the Python interpreter):

```
>>> import urllib

>>> f = urllib.urlopen("http://smswebservice1.m-science.com/
MScienceSMSWebService.asmx/SendMessages?encrypt=False&accountID=1&password=password&sourceTag=123&sourceAddress=123&deliveryReceipt=0&productMessageID=0&destination=123456&message=Message")

>>> print f.read()
```

Perl

Perl is another free scripting language, commonly used on a wide range of platforms. It is frequently used for implementing Common Gateway Interfacing (or cgi), using web servers such as Apache. There are various versions of Perl and a good starting point for investigation is http://www.perl.org.

The version used to develop the scripts provided here was 5.8.6 build 811 ActivePerl from ActiveState at http://www.activestate.com/Products/ActivePerl/ . This is available for a range of platforms.

To make SOAP calls onto the M:Science SMS Web Service, the free SOAP::Lite module was used. This can be downloaded for various platforms from http://www.soaplite.com/.

For writing active web content it recommended that the reader investigates mod_perl, which is a persistent interpreter for Apache web server, at http://perl.apache.org/.

SOAP::Lite comes as an archive which must be extracted to a folder. Once there, it must be built and installed using your existing perl compiler. This can be achieved through the following steps (correct as of V0.55 – check for changes) from the SOAP:Lite folder, ensuring that the perl compiler is in the command path:

Non Windows:

perl Makefile.PL

make

make test

make install

Windows:

perl Makefile.PL

nmake

nmake test

nmake install

note: 'nmake' is a utility available in Microsoft Visual Studio.

Having done this, the sample files can be run using the format 'perl sample.pl'. Note that the sample files parameter data must be customised to the specific user account details, source and destination numbers etc first.

An example of sending a message:

```
use SOAP::Lite (maptype => {} );

my $soap = SOAP::Lite

   -> uri('http://www.m-science.com/MScienceSMSWebService')
```

```
   -> on_action( sub { join '/', 'http://www.m-science.com/MScienceSMSWebService', $_[1] } )
   -> proxy('http://smswebservice1.m-science.com/MScienceSMSWebService.asmx');
my $method = SOAP::Data->name('SendMessages')
   ->attr({xmlns => 'http://www.m-science.com/MScienceSMSWebService/'});
my @params = (          SOAP::Data->name(encrypt => true),
                        SOAP::Data->name(accountID => 5),
                        SOAP::Data->name(password => dfhgjdfg),
                        SOAP::Data->name(sourceTag => \SOAP::Data->name(string => 123)),
                        SOAP::Data->name(sourceAddress => \SOAP::Data->name(string => )),
                        SOAP::Data->name(deliveryReceipt => \SOAP::Data->name(int => 0)),
                        SOAP::Data->name(productMessageID => \SOAP::Data->name(int => 0)),
                        SOAP::Data->name(destination => \SOAP::Data->name(string)->type(string =>
"+447886170298")),
                        SOAP::Data->name(message => \SOAP::Data->name(string => "Test Message")));
my $result = $soap->call($method => @params);
print $result->result();
```

Notice the extra code for the parameters which are passed as arrays.

Also case sensitivity is an issue with Perl and SOAP take care in getting it right otherwise it will not work.

PHP

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

PHP can be downloaded from http://www.php.net/. The source code given here was tested with version 5.0.3. NuSOAP is also required to communicate with the web service. This is a php file which can be downloaded from http://dietrich.ganx4.com/nusoap/. V 1.8.2 was tested here. It is recommended to put this source file into the same directory as the source code for ease of location.

The following code is an example of sending a text message:

```
<?php

require_once('nusoap.php');

$client = new soapclient('http://smswebservice1.m-science.com?WSDL', true);


$sourceTag = array('string' => '');

$sourceAddress = array('string' => '123');

$deliveryReceipt = array('int' => 0);

$productMessageID = array('int' => 0);

$destination = array('string' => '+447736123456');

$message = array('string' => 'A message from mscience...');


$params = array('encrypt'=> false, 'accountID' => 1, 'password' => 'mypassword', 'sourceTag' =>
$sourceTag, 'sourceAddress' => $sourceAddress, 'deliveryReceipt' => $deliveryReceipt,
'productMessageID' => $productMessageID, 'destination' => $destination, 'message' => $message);

$result = $client->call('SendMessages', array('parameters' => $params));

print_r($result);

echo '<h2>Response</h2><pre>' . htmlspecialchars($client->response, ENT_QUOTES) . '</pre>';

?>
```

Function References

Web Service Interface

GetNextInboundMessages

This string is used to retrieve up to the next five inbound messages. See below for the format which repeats for each message.

Parameters

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.

Return Codes

- OK- MESSAGEID - Retrieved message ok.
- SOURCEADDRESS - Source phone number.
- DESTINATIONADDRESS - Destination phone number.
- RECIEVEDATE - Date and time the inbound message was received in the format DDMMYYHHMMSS where DD – days, MM- months, YY – year, HH hours, MM minutes, SS seconds.
- PRODUCTMESSAGEID - Optional message id specified in SendMessage and returned with delivery receipts for correlation purposes.
- DELIVERYRECEIPT - True if it's a delivery receipt.
- RECEIPTDATE - Date of the delivery receipt (see receive date for format).
- MESSAGELENGTH - Number of bytes in the message text. Note that there is a comma between the end of the message text and the next message.
- MESSAGETEXT - Body of the message text
- OK-NOMESSAGES - No messages present
- FAILED–BADLOGIN - Cannot log in.
- FAILED-NULLPARAM - Some of the parameters supplied are null.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/GetNextInboundMessages"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetNextInboundMessages xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
    </GetNextInboundMessages>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetNextInboundMessagesResponse xmlns="http://www.m-
science.com/MScienceSMSWebService/">
      <GetNextInboundMessagesResult>string</GetNextInboundMessagesResult>
```

```
  </GetNextInboundMessagesResponse>
 </soap:Body>
</soap:Envelope>
```

AckMessages

Before retrieving more messages, it is first necessary to acknowledge receipt of each message already received. This is achieved by calling AckMessages and providing the message ID for each message.

Note: This function accepts an array of message IDs (max 10). For HTTP GET, this means that the MessageID parameter is being repeated in the URL parameter list for each array entry.

Parameters

[] – Indicates an array.

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.
- MessageID (Int 32[]) - Message ID array passed back by GetNextInboundMessages.

Return Codes

Note: In the case of multiple message IDs, there will be an array of return codes. E.g. OK-SUCCEEDED,FAILED-BADIDENT,OK-SUCC...

- OK-SUCCEEDED - OK
- FAILED–BADLOGIN - Unable to log in.
- FAILED-NULLPARAM - One or more of the parameters are NULL
- FAILED–BADIDENT - Unknown message ID.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/AckMessages"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AckMessages xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <messageID>
        <int>int</int>
        <int>int</int>
      </messageID>
    </AckMessages>
  </soap:Body>
```

```
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AckMessagesResponse xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <AckMessagesResult>string</AckMessagesResult>
    </AckMessagesResponse>
  </soap:Body>
</soap:Envelope>
```

SendMessages

This string is used to send SMS messages to the selected addresses. The messages will be filtered for unknown characters and split into 160 byte SMS messages. The source addresses should be the numbers that will appear on the end user's mobile handset, and subsequently be for any reply. If a delivery receipt is requested the caller must pass a unique message identifier in the ProductMessageID for each message. This will be sent back with the receipt to allow it to be correlated to the original message.

Parameters

[] – denotes an array (max 5). Note that all of the arrays passed must be the same size.

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.
- SourceTag (String []) - Not presently used but should be set to some identifying text describing the source of the message. E.g. SMS User Name.
- SourceAddress (String []) - Return Address (inbound number)
- DeliveryReceipt (Int 32 []) - 0 for no, 1 for Yes
- ProductMessageID (Int 32 []) - A user defined message identifier sent back with the delivery receipt as a cross reference to the original message.
- Destination (String []) - Destination phone number.
- Message (Sring []) - Message text. Max 160 characters or the message will be split into multiple messages.

Return Codes

Note: The actual string sent back will be a comma separated list of the return codes defined below.

The number of return codes will be the same as the number of messages sent unless there is a parameter inconsistency or any particular message is longer than 160 bytes, in which case there will be a return for each sub-message.

If the message send is successful then the following code will be returned:

- MESSAGEID - Unique identification ID for the message.
- BALANCE - New outbound message balance (number of messages).
- PENDING - Number of messages pending being sent.

- SURCHARGEBALANCE - Available funds in the surcharge account in the format ££££££PP where £ is Pounds Sterling and PP is pence.

If the message send fails then one of the following return codes will be presented:

- FAILED-NOMESSAGES - Insufficient message balance.
- FAILED-NOINTERCONNECTFUND - Insufficient funds in the interconnect account.
- FAILED-BADLOGIN - Unable to log in
- FAILED-BADPARAMERROR - Bad parameter or account blocked.
- FAILED-NULLPARAM - One or more parameters are NULL
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.
- FAILED-BADSOURCEADDRESS – The source address specified contains characters other than numeric and '+'.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/SendMessages"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SendMessages xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <sourceTag>
        <string>string</string>
        <string>string</string>
```

```
      </sourceTag>
      <sourceAddress>
        <string>string</string>
        <string>string</string>
      </sourceAddress>
      <deliveryReceipt>
        <int>int</int>
        <int>int</int>
      </deliveryReceipt>
      <productMessageID>
        <int>int</int>
        <int>int</int>
      </productMessageID>
      <destination>
        <string>string</string>
        <string>string</string>
      </destination>
      <message>
        <string>string</string>
        <string>string</string>
      </message>
    </SendMessages>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

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

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SendMessagesResponse xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <SendMessagesResult>string</SendMessagesResult>
    </SendMessagesResponse>
  </soap:Body>
</soap:Envelope>
```

GetMessageStatus

This string returns the current status of the message specified by the MessageID as sent back by SendMessage. Note that this does not include information from delivery receipts. The purpose of this function is to determine the message status against the M:Science host. If you wish to determine message status from the mobile device then you will to request and evaluate the relevant delivery receipt.

Parameters

[] – Indicates an Array.

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String_) - Password used to login to the website.
- MessageID (Int 32 []) - Message identifiers passed back from SendMessage.

Return Codes

Note: The actual string sent back for each message will be a comma separated list of the return codes defined below.

- OK-SENTOK - Message has been sent ok.
- OK-PENDING - Trying to send message.
- OK-FAILED - Message send has failed.
- FAILED-BADID - Unknown message ID.
- FAILED-NULLPARAM - One or more parameters is NULL
- FAILED-BADLOGIN - Unable to log in.

- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/GetMessageStatus"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetMessageStatus xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <messageID>
        <int>int</int>
        <int>int</int>
      </messageID>
    </GetMessageStatus>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

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

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetMessageStatusResponse xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <GetMessageStatusResult>string</GetMessageStatusResult>
    </GetMessageStatusResponse>
  </soap:Body>
</soap:Envelope>
```

RegisterForInbound

Allows a client to register for asynchronous inbound message calls. To accept these, there must be http servers at the addresses specified capable of accepting a POST with the parameters described for ReceiveInboundMessage.

Parameters

[] – Denotes an array.

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.
- Address (String []) - Array of fully qualified web addresses. E.g. http://www.mycompany.com/ResponseService.amsx/ReceiveInboundMessage

Return Codes

Note: The actual return code will be a comma separated list of return codes; one for each attempted registration unless there is a global failure. E.g. 'OK-REGISTERED,OK-REGISTERED, FAILED-ALREADYUSED,OK-REG....'

- OK-REGISTERED - Registered successfully.
- FAILED-BADLOGIN - Unable to log in.
- FAILED-NULLPARAM - One or more parameters is NULL.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.
- FAILED-ALREADYUSED - The specified address is already registered.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/RegisterForInbound"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <RegisterForInbound xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <address>
        <string>string</string>
        <string>string</string>
      </address>
    </RegisterForInbound>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
```

<RegisterForInboundResponse xmlns="http://www.m-science.com/MScienceSMSWebService/">

<RegisterForInboundResult>**string**</RegisterForInboundResult>

</RegisterForInboundResponse>

</soap:Body>

</soap:Envelope>

UnregisterInboundAddress

Unregisters asynchronous callback addresses specified in RegisterForInbound. The addresses must exactly match the ones specified when the callbacks were initially registered.

Parameters

[]- Denotes an array.

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID (as used to login to the website)
- Password (String) - Password used to log into the website.
- Address (String []) - Fully qualified web address. E.g. http://www.mycompany.com/ResponseService.amsx/ReceiveInboundMessage

Return Codes

Note: The actual return code will be a comma separated list of result codes. One for each attempted registration unless there is a global failure. E.g. 'OK-UNREGISTERED, FAILED-NOTFOUND,OK-UNR...'

- OK-UNREGISTERED - Unregistered successfully.
- FAILED-BADLOGIN' - Unable to log in.
- FAILED-NULLPARAM - One or more parameters is NULL.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.
- FAILED-NOTFOUND - The specified address was not registered.

XML

POST /MScienceSMSWebService.asmx HTTP/1.1

Host: smswebservice1.m-science.com

Content-Type: text/xml; charset=utf-8

Content-Length: **length**

```
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/UnregisterInboundAddress"


<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <UnregisterInboundAddress xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <address>
        <string>string</string>
        <string>string</string>
      </address>
    </UnregisterInboundAddress>
  </soap:Body>
</soap:Envelope>


HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length


<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <UnregisterInboundAddressResponse xmlns="http://www.m-
science.com/MScienceSMSWebService/">
      <UnregisterInboundAddressResult>string</UnregisterInboundAddressResult>
    </UnregisterInboundAddressResponse>
  </soap:Body>
```

</soap:Envelope>

ResetInboundCallbacks

Removes all callback registrations specified by RegisterForInbound.

Parameters

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.

Return Codes

The actual string sent back for each message will be a comma separated list of the return codes defined below.

- OK-UNREGISTERED - Unregistered successfully.
- FAILED-BADLOGIN - Unable to log in.
- FAILED-NULLPARAM - One or more parameters is NULL.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/ResetInboundCallbacks"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ResetInboundCallbacks xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
```

```
  </ResetInboundCallbacks>
 </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <ResetInboundCallbacksResponse xmlns="http://www.m-
science.com/MScienceSMSWebService/">
   <ResetInboundCallbacksResult>string</ResetInboundCallbacksResult>
  </ResetInboundCallbacksResponse>
 </soap:Body>
</soap:Envelope>
```

GetURLStatus

Returns the communications status of the selected URL.

Parameters

- Encrypt (Boolean) - Specifies that the account and password are encrypted.
- AccountID (String) - User ID used to login to the website.
- Password (String) - Password used to login to the website.
- URL (String) - URL to check the status of (must be registered).

Return Codes

The actual string sent back for each message will be a comma separated list of the return codes defined below.

- OK-READY - URL Ready
- OK-ACTIVE - Currently sending
- OK-RETRY - Retrying on URL

- OK-FAILED - Link is down
- OK-STATENOTSET - State is null
- FAILED-BADLOGIN - Unable to log in
- FAILED-NULLPARAM - One or more parameters is NULL
- FAILED-UNKNOWNURL - URL not recognized.
- FAILED-DBERR - Internal database error.
- FAILED-NOTLICENSED - Not licensed for this functionality.

XML

```
POST /MScienceSMSWebService.asmx HTTP/1.1
Host: smswebservice1.m-science.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.m-science.com/MScienceSMSWebService/GetURLStatus"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetURLStatus xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <encrypt>boolean</encrypt>
      <accountID>string</accountID>
      <password>string</password>
      <address>string</address>
    </GetURLStatus>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
```

Content-Length: **length**

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetURLStatusResponse xmlns="http://www.m-science.com/MScienceSMSWebService/">
      <GetURLStatusResult>string</GetURLStatusResult>
    </GetURLStatusResponse>
  </soap:Body>
</soap:Envelope>
```

Client Interface

Called asynchronously using an HTTP POST from the M:Science SMS Web Service server when an inbound message arrives. The address must have been registered using RegisterForInbound and must conform to the method call below. E.g. http://www.mycompany.com/ResponseService.amsx/ReceiveInboundMessage

ReceiveInboundMessage

Parameters

- Destination (String) - Inbound number originating the message.
- Source (String) - Number of the mobile from which the message originated.
- ProductMessageID (String) - ID code passed in with SendMessage. Can be used to cross reference delivery receipts with sent messages.
- ReceiveDate (String) - Date and time the message was originally received.
- DeliveryReceipt (String) - Indicates if the message is a receipt for previously sent one.
- ReceiptDate (String) - Date of the receipt.
- Message (String) - Text of the message.
- MessageID (String) - Identifier of the message.

Return Codes

The actual string sent back for each message will be a comma separated list of the return codes defined below.

- OK-RECEIVED - Message has been received ok and can be cleared from the InboundWebQueue.
- FAILED-RETRY - Message has not been received properly (for example destination database unreachable), try again on the next retry interval.