MESSAGE MANAGER

Message Manager
Sending & Receving
Webservice API
Developer Reference

Version 6.00

PROVIDED BY

Table of Contents

I. Messenger API functions as webservice

This set of functions is designed for sending and receiving SMS/MMS messages and for Inbox operation in third-party clients. Each function comes in two variants: for UTF-8 and for Windows-1256 encoding. The function parameters universal to all functions are underlined, and you can see their full descriptions in the API parameters universal to all functions.

Messenger API uses SOAP services, so you need to add a Web Reference called ‘MessengerService’ to your project before you can compile the examples from this document.

The URL for the Web Reference for Ooredoo Messaging System is `https://messaging.ooredoo.qa/bms/soap/Messenger.asmx`

You can also access the unsecured version of the Web Reference for Ooredoo Messaging System at `http://messaging.ooredoo.qa/bms/soap/Messenger.asmx`

Please use the following code to initialize Messenger API before calling any of its functions:

```
MessengerSoapClient messenger =
      new MessengerService.MessengerSoapClient("MessengerSoap");
// "MessengerSoap" - endpoint name in configuration file
```

I.1. API parameters universal to all functions

I.1.a. customerID

The unique identifier for every customer in the Message Manager Platform database.

I.1.b. customerUserName

A user is the actual actor who uses the Messaging Platform services. A customer can have several users who access the Messaging Platform services under the customer account. A user does not have a balance of their own, when a user is sending messages, their customer account is charged. Every user has a separate user name, a password, and can have a separate set of originators.

I.1.c. originatorName

The originator is the name or number that the recipient sees as the signature for the message received.

II. Authenticating the user

The ***Authenticate*** function authenticates the user specified by the customer ID, user name, language, and password in MESSAGEMANAGER PLATFORM system.

The example provided below is the ***Authenticate*** function prototype:

```
Authenticate(SoapUser user)
```

The required parameters are:

- customerID – The unique identifier for every customer in the platform
- customerUsername - for the user for whom the list of originators is requested.
- userPassword—the password of the user specified.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.

An example of calling the ***Authenticate*** function is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 1;
user.Name = "Bob";
user.Language = "en";
user.Password = "secret";

AuthResult authData = messenger.Authenticate(user);

if (authData.Result == "OK")
{
    Console.WriteLine("Hello, Bob");
}
else
{
    Console.WriteLine("Error: " + authData.Result);
}
```

An example of responding to the ***Authenticate*** function is provided below:

```
<?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>
    <AuthenticateResponse xmlns="http://pmmsoapmessenger.com/">
      <AuthenticateResult>
        <NetPoints>string</NetPoints>
        <Originators>
          <string>string</string>
          <string>string</string>
        </Originators>
        <CustomerID>int</CustomerID>
        <CreditSMS>string</CreditSMS>
        <CreditMMS>string</CreditMMS>
      </AuthenticateResult>
    </AuthenticateResponse>
  </soap:Body>
</soap:Envelope>
```

You can also use the HTTP_***Authenticate*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/soap/Messenger.asmx/HTTP_Authenticate?customerID=string&userName=str
```

```
ing&userPassword=string

HTTP/1.1
```

III.Sending SMS messages

With the help of the ***SendSms*** function the customer can send SMS messages using the specified parameters.

The example provided below is the ***SendSms*** function prototype:

```
SendSms(SoapUser user, string originator, string smsData, string
recipientPhone, MessageType messageType, string defDate, boolean blink,
boolean flash, boolean private)
```

The required parameters are:

- customerID – The unique identifier for every customer in the platform
- customerUsername - for the user for whom the list of originators is requested.
- userPassword—the password of the user specified.
- originator—the parameter identical to originatorName
- smsText—the text for the SMS message to be sent.
- recipientPhone—the recipient's phone numbers separated by commas (the parameter can contain just one number).
- messageType—the type of the character set used in the SMS text (Latin, ArabicWithArabicNumbers, ArabicWithLatinNumbers).
- defDate—the parameter specifying the date and time of deferred delivery (must be in the **yyyyMMddhhmmss** format).

  Note: The defDate parameter has to be empty for IMMEDIATE delivery, and expects the time in GMT.
- flash—the parameter defines whether the SMS message is to be saved in the recipient phone inbox (True—the message is not saved, False—the message is saved).
- blink—the parameter defines whether the SMS message blinks when onscreen (applicable for some makes of Nokia mobile phones).
- private—the parameter defines whether the SMS message appears in the account statement.

An example of calling the ***SendSms*** function is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 1;
user.Name = "Bob";
user.Language = "en";
user.Password = "secret";

string originator = authResult.Originators[0];
string defDate = DateTime.UtcNow.AddHours(1).ToString("yyyyMMddhhmmss");
string smsData = "test";
string phone = "971505000001, 971505000002, 971505000003";

SendResult result = messenger.SendSms(user, originator, smsData, phone,
       MessageType.Latin, defDate, false, false, false);
Console.WriteLine(result.Result);
```

An example of responding to the ***SendSms*** function is provided below:

```
<?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>
    <SendSmsResponse xmlns="http://pmmsoapmessenger.com/">
      <SendSmsResult>
        <RejectedNumbers>
          <string>string</string>
          <string>string</string>
        </RejectedNumbers>
        <TransactionID>string</TransactionID>
        <NetPoints>string</NetPoints>
      </SendSmsResult>
    </SendSmsResponse>
  </soap:Body>
</soap:Envelope>
```

You can also use the ***HTTP_SendSms*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/soap/Messenger.asmx/HTTP_SendSms?customerID=string&userName=string&u
serPassword=string&originator=string&smsText=string&recipientPhone=string&mes
sageType=string&defDate=string&blink=string&flash=string&Private=string

HTTP/1.1
```

IV. Sending Bulk SMS messages

With the help of the ***SendSmsXML*** function the customer can send SMS messages using the specified parameters.

The example provided below is the ***SendSmsXML*** function prototype:

```
<?xml version='1.0' encoding='UTF-8'?>
<request><bulk_msg>
<customer_id>CustomerID</customer_id>
<user_id>username</user_id>
<password>Password</password>
<validity_period status='y'>
<day>Validity in Days</day>
<hours>Validity in Hours</hours>
<mins>Validity in Minutes</mins>
</validity_period>
<delivery_report>0</delivery_report>
<message>
<title>Originator</title>
<lang_id>Language</lang_id>
<body><![CDATA[smsText containing tags]]></body>
<values>
<msg_id>IncrementalCounter</msg_id>
<mobile_no>recipientPhone</mobile_no>
<param id='1'><![CDATA[Actual tag data]]></param>
<param id='2'><![CDATA[Data2]]></param>
<param id='3'><![CDATA[Data3]]></param>
<param id='4'><![CDATA[Data4]]></param>
</values>
</message>
</bulk_msg>
</request>
```

The required parameters are:

- customerID – The unique identifier for every customer in the platform
- customerUsername - for the user for whom the list of originators is requested.
- userPassword—the password of the user specified.
- originator—the parameter identical to originatorName

- Validity – allows you to specify the validity of the message in days, hours and minutes
- smsText—the text for the SMS message to be sent containing tags referred to by ~%number%~ where %number% is the parameter provided
- IncrementalCounter – a counter that is incremental for each data/ recipient provided
- recipientPhone—the recipient's phone number (the parameter has to contain just one number).
- Language —the type of the character set used in the SMS text (0 for Latin, 1 for Arabic).

An example of calling the ***SendSmsXML*** function is provided below:

```
<?xml version='1.0' encoding='UTF-8'?>
<request>
  <bulk_msg>
    <customer_id>74</customer_id>
    <user_id>u74</user_id>
    <password>123</password>
    <validity_period status="y">
      <day>2</day>
      <hours>3</hours>
      <mins>10</mins>
    </validity_period>
    <delivery_report>0</delivery_report>
    <message>
      <title>o74</title>
      <lang_id>0</lang_id>
      <body><![CDATA[Dear ~1~, your salary for the month of July is ~2~
~3~.]]></body>
      <values>
         <msg_id>1</msg_id>
         <mobile_no>375296509492</mobile_no>
         <param id="1"><![CDATA[Deepak]]></param>
         <param id="2"><![CDATA[QAR]]></param>
         <param id="3"><![CDATA[18764.50]]></param>
         <param id="4"><![CDATA[18764.50]]></param>
      </values>
      <values>
         <msg_id>2</msg_id>
         <mobile_no>375296509493</mobile_no>
         <param id="1"><![CDATA[Prashanth]]></param>
         <param id="2"><![CDATA[USD]]></param>
         <param id="3"><![CDATA[10890.00]]></param>
      </values>
    </message>
  </bulk_msg>
</request>
```

An example of responding to the ***SendSmsXML*** function is provided below:

```
<?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>
    <SendSmsResponse xmlns="http://pmmsoapmessenger.com/">
      <SendSmsResult>
        <RejectedNumbers>
          <string>string</string>
          <string>string</string>
        </RejectedNumbers>
        <TransactionID>string</TransactionID>
        <NetPoints>string</NetPoints>
      </SendSmsResult>
    </SendSmsResponse>
  </soap:Body>
</soap:Envelope>
```

V. Sending Binary SMS messages

With the help of the ***SendBinarySms*** function the customer can send binary SMS messages using the specified parameters.

The example provided below is the ***SendBinarySms*** function prototype:

```
SendBinarySms(SoapUser user, String originator, String binaryBody,
String recipientPhone, String defDate, String data_coding, String esm_class,
String PID)
```

The required parameters are:

- customerID – The unique identifier for every customer in the platform
- customerUsername - for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.
- originator—the parameter identical to originatorName.
- binaryBody—the binary body of the SMS message.
- recipientPhone— the recipient's phone numbers separated by commas (the parameter can contain just one number).
- defDate—the parameter specifying the date and time of deferred delivery (must be in the **yyyyMMddhhmmss** format).
- data_coding—the coding for the binary body of the SMS message.
- esm_class—the parameter related to the coding of the binary body of the SMS message.
- PID—the protocol ID.

An example of calling the ***SendBinarySms*** function is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 1;
user.Name = "Bob";
user.Language = "en";
user.Password = "secret";

string originator = authResult.Originators[0];
string defDate = DateTime.UtcNow.AddHours(1).ToString("yyyyMMddhhmmss");
string smsData = "test";
string phone = "971505000001";

SendResult binSmsSendResult =
       messenger.SendBinarySms(user, originator, smsData , phone, defDate,
              string.Empty, string.Empty, string.Empty);
Console.WriteLine();
```

An example of responding to the ***SendBinarySms*** function is provided below:

```
<?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>
    <SendBinarySmsResponse xmlns="http://pmmsoapmessenger.com/">
      <SendBinarySmsResult>
        <RejectedNumbers>
```

```
          <string>string</string>
          <string>string</string>
        </RejectedNumbers>
        <TransactionID>string</TransactionID>
        <NetPoints>string</NetPoints>
      </SendBinarySmsResult>
    </SendBinarySmsResponse>
  </soap:Body>
</soap:Envelope>
```

You can also use the ***HTTP_SendBinarySms*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/soap/Messenger.asmx/HTTP_SendBinarySms?customerID=string&userName=st
ring&userPassword=string&originator=string&binaryBody=string&recipientPhone=s
tring&defDate=string&data_coding=string&esm_class=string&PID=string

HTTP/1.1
```

VI. Sending MMS messages

With the help of the ***SendMMS*** function the customer can send MMS messages using the specified parameters.

The example provided below is the ***SendMMS*** function prototype:

```
SendMMS(byte[] mm7Data)
```

The required parameter is:

- mm7Data—the standard format for sending MMS messages. Along with media files and other data, the parameter contains the recipient phone numbers separated by commas (one number is possible).

An example of calling the ***SendMMS*** function is provided below:

```
Byte[] mm7Data;
using (FileStream fileStream = new FileStream(path, FileMode.Open))
{
mm7Data = new Byte[fileStream.Length];
fileStream.Read(mm7Data, 0, mm7Data.Length);
}

SendMMSResult mmsSendResult = messenger.SendMMS(mm7Data);
Console.WriteLine(mmsSendResult.Result);
```

An example of responding to the ***SendMMS*** function is provided below:

```
<?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>
    <SendMMSResponse xmlns="http://pmmsoapmessenger.com/">
      <SendMMSResult>
        <MESSAGEMANAGER PLATFORMTransactionID>string</MESSAGEMANAGER
PLATFORMTransactionID>
      </SendMMSResult>
    </SendMMSResponse>
  </soap:Body>
</soap:Envelope>
```

VI.1. Transforming MMS content into ‘mm7’ format and sending the resulting mm7 file

The three pieces of code provided below show how to compile and send an MMS message using the ***SendMMS*** function. The second piece details one of the methods (*GenerateMIME*) mentioned in the first part, and the third piece details one of the methods (*WriteToMime*) mentioned in the second piece.

First, the paths where to put the ‘SMIL’ file and the MMS content are defined. Then the *GenerateMIME* method uses the paths to generate the resulting mm7 file and save it to the path specified. Then the *FileStream* method takes the mm7 file and transforms it into a datastream to be transferred to the ***SendMMS*** function.

```
String mm7FilePath = "D:\\<some_folder_1>\\<some_folder_2>\\<mm7FileName>";

String mmsContentFolderPath = "D:\\<some_folder_1>\\<some_folder_2>\\";

GenerateMIME( mmsContentFolderPath, mm7FilePath );

Byte[] mm7data = null;
```

```
using (FileStream fs = new FileStream(mm7FilePath, FileMode.Open,
FileAccess.Read))
{
    mm7data = new Byte[fs.Length];
    fs.Read(mm7data, 0, mm7data.Length);
    String mm7DataString = Convert.ToBase64String(mm7data,
                                            Base64FormattingOptions.None);
}

Messenger.Messenger messengerService = new Messenger.Messenger();
SendMMSResult sendMMSresult = messengerService.SendMMS(mm7data);
```

The *GenerateMIME* method mentioned in the example above puts the SMIL file into the mm7 file. The *GenerateMIME* method is detailed in the lengthy example below:

```
public static void GenerateMIME(String dataDirectoryName, string
                                outputFileName)
{
    DateTime dt = DateTime.Now;
    String sMonth = String.Empty;
    String sDay = String.Empty;
    String sYear = String.Empty;
    String sHour = String.Empty;
    String sMinute = String.Empty;
    String sSecond = String.Empty;
    String sMillisecond = String.Empty;

    // Getting data for boundary string
    int iMonth = dt.Month;
    if (iMonth <= 9) sMonth = "0" + iMonth.ToString();
    else sMonth = iMonth.ToString();

    int iDay = dt.Day;
    if (iDay <= 9) sDay = "0" + iDay.ToString();
    else sDay = iDay.ToString();

    int iYear = dt.Year;
    sYear = iYear.ToString().Substring(2, 2);

    int iHour = dt.Hour;
    if (iHour <= 9) sHour = "0" + iHour.ToString();
    else sHour = iHour.ToString();

    int iMinute = dt.Minute;
    if (iMinute <= 9) sMinute = "0" + iMinute.ToString();
    else sMinute = iMinute.ToString();

    int iSecond = dt.Second;
    if (iSecond <= 9) sSecond = "0" + iSecond.ToString();
    else sSecond = iSecond.ToString();

    int iMillisecond = dt.Millisecond;
    if (iMillisecond <= 99)
    {
        if (iMillisecond <= 9) sMillisecond = "00" + iMillisecond.ToString();
        else sMillisecond = "0" + iMillisecond.ToString();
    }
    else sMillisecond = iMillisecond.ToString();

    String sDt = sMonth + sDay + sYear + sHour + sMinute + sSecond +
                                                        sMillisecond;
    String content_ref = "11351";
    String smil_reference = "20412";
    // Set up boundary string
    String boundary = "--------" + sDt;
    String[] files = Directory.GetFiles(dataDirectoryName, "*.smil");

    String smilFileFull = files[0];

    String[] a = smilFileFull.Split('\\');
    int count = a.Length;
    String smilFileName = a[count - 1];


    // Writing data into mime-file
    StreamWriter sw = File.CreateText(outputFileName);
    StreamReader sr = new StreamReader(smilFileFull);

    try
    {
        // header generation
        sw.WriteLine("Content-Type: multipart/related;");
        sw.WriteLine("\t boundary=\"" + boundary + "\";");
        sw.WriteLine("\t start=\"smil_reference_" + smil_reference + "\";");
        sw.WriteLine("\t type=\"application/smil\"");
        sw.WriteLine("Content-Id: <content_ref_" + content_ref + ">");

        // smil inserting
        sw.WriteLine("");
        sw.WriteLine("--" + boundary);
        sw.WriteLine("Content-Id: <smil_reference_" + smil_reference + ">");
        sw.WriteLine("Content-Type: application/smil; name=\"" + smilFileName
                                                        + "\"");
        sw.WriteLine("Content-location: " + smilFileName);
        sw.WriteLine("");
        sw.Write(sr.ReadToEnd());
        sw.WriteLine("");

        //attachments inserting

        DirectoryInfo di = new DirectoryInfo( dataDirectoryName );
        FileInfo[] fInfos = di.GetFiles();
        String StringArray = String.Empty;

        for( int i = 0; i < fInfos.Length; i++ )
        {
            if (!fInfos[i].Extension.Equals("smil"))
            {
                continue;
            }

            using (FileStream fs = new FileStream(fInfos[i].FullName,
                                           System.IO.FileMode.Open))
            {
                if (fInfos[i].Extension.Equals("txt"))
                {
                    StreamReader reader = new StreamReader(fs,
                                 System.Text.Encoding.GetEncoding(1256));
                            StringArray = reader.ReadToEnd();
                }
                else
                {
                    Byte[] ByteArray = new Byte[fs.Length];
                    fs.Read(ByteArray, 0, ByteArray.Length);
                    StringArray = Convert.ToBase64String(ByteArray, 0,
                                                     ByteArray.Length);
                }

                WriteToMime( sw, boundary, contentType, fInfos[i].Name,
                            StringArray );
            }


        }
        // footer generation
        sw.WriteLine("");
        sw.WriteLine("--" + boundary + "--");
    }
    catch (Exception ex)
    {
        //Exception handling
    }
    finally
    {
        sw.Close();
        sr.Close();
     }
}
```

(the end of the *GenerateMIME* method description)

The *WriteToMime* method mentioned in the example above is responsible for putting the media files into the same mm7 file. The method is detailed in the example below:

```
public static void WriteToMime(StreamWriter sw, String boundary,
                               String contentType, String contentLocation,
                               String stringArray)
{
    sw.WriteLine("");
    sw.WriteLine("--" + boundary);
    sw.WriteLine("Content-Id: <" + contentLocation + ">;
                 " + "name=\"" + contentLocation + "\"");
    sw.WriteLine("Content-Type: " + contentType);
    sw.WriteLine("Content-location: " + contentLocation);
    if ( !contentType.StartsWith("text"))
    {
        sw.WriteLine( "Content-Transfer-Encoding: base64" );
    }
    sw.WriteLine("");

    int RecordsNum = 0;
    int iRes = stringArray.Length / 76;
    if ((iRes * 76) < stringArray.Length) RecordsNum = iRes + 1;
    else RecordsNum = iRes;
    String[] a = null;
    a = new String[RecordsNum];
    int j = 0;
    for (int i = 0; i < RecordsNum; i++)
    {
        if (!(i == RecordsNum - 1)) a[i] = stringArray.Substring(j, 76);
        else
        {
            int iLengthRemain = stringArray.Length - ((RecordsNum - 1) * 76);
            a[i] = stringArray.Substring(j, iLengthRemain);
        }

        sw.WriteLine(a[i]);

        j += 76;
    }
}
```

VII. Getting SMS/MMS message status

To get statuses for both SMS and MMS messages, the ***GetSmsStatus*** function is used. Actually, it operates with the message transactionID which does not depend on the message type. The ***GetSmsStatus*** function returns the delivery status of the specified SMS/MMS message.

The example provided below is the ***GetSmsStatus*** function prototype:

```
GetSmsStatus(SoapUser user, String transactionID, Boolean detailed)
```

The required parameter is:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.
- transactionID—the unique identification of the transaction.
- detailed—Boolean value which defines if detailed statuses for multiple recipients are to be show. It can be 'True' or 'False'.

An example of calling the ***GetSmsStatus*** function is provided below:

```
// send sms
SendResult sendResult = messenger.SendSms(
               user, authResult.Originators[0], "test", "971505000001",
               MessageType.Latin, DateTime.UtcNow.ToString("yyyyMMddhhmmss"),
               false, false, false);
// retrieve its' status
SmsStatus status = GetSmsStatus(user, sendResult.TransactionID, true);
Console.WriteLine(status.Result);
```

An example of responding to the ***GetSmsStatus*** function is provided below:

```
<?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>
    <GetSmsStatusResponse xmlns="http://pmmsoapmessenger.com/">
      <GetSmsStatusResult>
        <Statistics>xml</Statistics>
        <Details>xml</Details>
        <NetPoints>string</NetPoints>
      </GetSmsStatusResult>
    </GetSmsStatusResponse>
  </soap:Body>
</soap:Envelope>
```

You can also use the ***HTTP_GetSmsStatus*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/soap/Messenger.asmx/HTTP_GetSmsStatus?customerID=string&userName=str
ing&userPassword=string&transactionID=string&detailed=string

HTTP/1.1
```

VIII. Receiving SMS messages

To receive SMS messages, the ***InboxProcessing*** function is used. With the help of the operations with the 'list' prefix SMS messages can be listed, when the actual texts of the SMS messages are not returned. With the help of the operations with the 'fetch' prefix SMS messages can be returned in full, with the texts as well as all the other parameters. After the SMS messages are fetched, they are marked as 'read'.

The example provided below is the ***InboxProcessing*** function prototype:

```
InboxProcessing(SoapUser user, String operation, String messageId)
```

The required parameters are:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.
- operation—the operation to be performed with SMS messages. For receiving SMS messages the operations with the 'list' and 'fetch' prefixes are used (see their descriptions below).
- messageId—the unique identifier for the message, to be used for the 'fetch-id' operation only.

You can also use the ***HTTP_InboxProcessing*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/SOAP/Messenger.asmx/HTTP_InboxProcessing?customerID=string&userName=
     string&userPassword=string&operation=string&messageId=string

HTTP/1.1
```

VIII.1. Listing SMS messages available for the Inbox

The operations with the 'list' prefix for SMS messages are:

- list-all;
- list-new.

VIII.1.a. Listing all SMS messages

An example for the request for listing all SMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "list-all";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for listing all SMS messages are provided below:

If the user is

```
<inboxlist xmlns="">
```

unknown

```
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no messages

```
 <inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are messages available (the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="b7d7508c-0d68-4e0f-acaa-6f2ec27947e7" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>14</size>
        <subject />
        <msgtype>SMS</msgtype>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the number of characters for the SMS messages.

VIII.1.b. Listing new SMS messages

An example for the request for listing new SMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "list-new";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for listing new SMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no new messages

```
 <inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are new messages available (the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="b7d7508c-0d68-4e0f-acaa-6f2ec27947e7" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>14</size>
        <subject />
        <msgtype>SMS</msgtype>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the number of characters for the SMS messages.

VIII.2. Fetching SMS messages available for the Inbox

The operations with the 'fetch' prefix for SMS messages are:

- fetch-all;
- fetch-new;
- fetch-id.

Please note that after any of the 'fetch' operations is performed, the fetched messages are marked as 'read'.

VIII.2.a. Fetching all SMS messages

An example for the request for fetching all SMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-all";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for fetching all SMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no messages

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are messages available (the messages are fetched with the text and then marked as read, the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="b7d7508c-0d68-4e0f-acaa-6f2ec27947e7" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>19</size>
        <subject />
        <msgtype>SMS</msgtype>
        <text>Please call me back</text>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the number of characters for the SMS messages.

VIII.2.b. Fetching new SMS messages

An example for the request for fetching new SMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-new";
String messageID = String.Empty;
```

```
Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                      messageID);
```

Examples of possible responses for fetching new SMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no new messages

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are new **SMS** messages available (the messages are fetched with the text and then marked as read, the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="b7d7508c-0d68-4e0f-acaa-6f2ec27947e7" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>18</size>
        <subject />
        <msgtype>SMS</msgtype>
        <text>Contract is signed</text>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the number of characters for the SMS messages.

VIII.2.c. Fetching SMS messages by their IDs

An example for the request for fetching SMS messages by the IDs is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-id";
String messageID = "b7d7508c-0d68-4e0f-acaa-6f2ec27947e7";

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                      messageID);
```

Please note that in case there are several SMS message IDs, they must be separated by semicolon.

Example of possible responses for fetching SMS messages by the IDs are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there is no messages with this ID

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>88000</credits>
</inboxlist>
```

If there is the **SMS** message with the ID specified (the message is fetched with the text and then

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="b7d7508c-0d68-4e0f-acaa-6f2ec27947e7" new="1">
        <from>20108000000</from>
        <to>xenon</to>
        <timestamp>20090421172256</timestamp>
        <size>36</size>
```

marked as read, the number of message blocks equals the number of messages).	`        <subject />`
	`        <msgtype>SMS</msgtype>`
	`        <text>The cargo is to be shipped in 2 days</text>`
	`    </message>`
	`    <credits>87000</credits>`
	`</inboxlist>`

The Size node contains the number of characters for the SMS messages.

IX. Receiving MMS messages

To receive MMS messages, the ***InboxProcessing*** function is used. With the help of the operations with the 'list' prefix MMS messages can be listed, when the actual multimedia contents of the MMS messages are not returned. With the help of the operations with the 'fetch' prefix an MMS message can be returned in full, with all the multimedia content, as well as the subject and all the other parameters. After the MMS messages are fetched, they are marked as 'read'.

The example provided below is the ***InboxProcessing*** function prototype:

```
InboxProcessing(SoapUser user, String operation, String messageId)
```

The required parameters for the ***InboxProcessing*** function are:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.
- operation—the operation to be performed with MMS messages. For receiving MMS messages the operations with the 'list' and 'fetch' prefixes are used (see their descriptions below).
- messageId—the unique identifier for the message, to be used for the 'fetch-id' operation only. If there are several identifiers, they must be separated by semicolon.

You can also use the ***HTTP_InboxProcessing*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/SOAP/Messenger.asmx/HTTP_InboxProcessing?customerID=string&userName=
     string&userPassword=string&operation=string&messageId=string

HTTP/1.1
```

IX.1. Listing MMS messages available for the Inbox

The operations with the 'list' prefix for MMS messages are:

- list-all;
- list-new.

IX.1.a. Listing all MMS messages

An example for the request for listing all MMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "list-all";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for listing all MMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no messages

```
 <inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are messages available (the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="082bbe87-4b76-4cfb-b4ef-236daf429a04" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>45</size>
        <subject>Meet me at airport</subject>
        <msgtype>MMS</msgtype>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the size in KB for the MMS messages.

IX.1.b. Listing new MMS messages

An example for the request for listing new MMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";
String operation = "list-new";
String messageID = String.Empty;
Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for listing new MMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no new messages

```
 <inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are new messages available (the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="082bbe87-4b76-4cfb-b4ef-236daf429a04" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>4</size>
        <subject>Meet me at airport</subject>
        <msgtype>MMS</msgtype>
```

```
     </message>
     <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the size in KB for the MMS messages.

IX.2. Fetching MMS messages available for the Inbox

The operations with the 'fetch' prefix for MMS messages are:

- fetch-all;
- fetch-new;
- fetch-id.

Please note that after any of the 'fetch' operations is performed, the fetched messages are marked as 'read'.

IX.2.a. Processing an MMS message request

An example of a typical request to fetch an MMS message (a request to fetch an MMS message by ID is shown):

```
SoapUser user = new SoapUser();
     user.CustomerID = 2;
     user.Name = "q";
     user.Language = "en";
     user.Password = "q";

     String operation = "fetch-id";
     String messageID = "b7d7508c-0d68-4e0f-acaa-6f2ec27947e7";

     Messenger.Messenger messengerService = new Messenger.Messenger();
     XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                        messageID);
```

An XML response for the request contains the 'content' tag, which in its turn contains the string which is a MIME-encoded zip file. The operation described below saves the zip file to the hard disk, thus making the file available for the further operations.

```
//getting content as Base64 encoded String
XmlNode contentNode = xmlResponse.SelectSingleNode( "content" );
String contentAsBase64String = xmlResponse.InnerText;

Byte[] contentReceived = Convert.FromBase64String( contentAsBase64String );

//saving received file on disk
String incomingFileFullName =
       "D:\\<some_folder_1>\\<some_folder_2>\\<some_filename>.zip";

using ( FileStream fs = new FileStream(incomingFileFullName, FileMode.Create,
                                       FileAccess.Write) )
{
    fs.Write(contentReceived, 0, contentReceived.Length);
}

UnZipMMS (incomingFileFullName);
```

You as a developer must implement some 'UnZipMMS' method to be used to unzip the previously saved zip file with the MMS content. Then the MMS content is available for the further operations.

IX.2.b. Fetching all MMS messages

An example for the request for fetching all MMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-all";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for fetching all MMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no messages

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are messages available (the messages are fetched with the text and then marked as read, the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="082bbe87-4b76-4cfb-b4ef-236daf429a04" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>4</size>
        <subject>Meet me at airport</subject>
        <msgtype>MMS</msgtype>
        <text>Road map to airport</text>
        <content>UEsDBBQAAAAIAB1YlzpRB98gKCcBALonA...</content>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the size in KB for the MMS messages. The Content node for the MMS message in the example above contains a Base64 encoded zipped smil file and the multimedia content (the string shown incomplete).

IX.2.c. Fetching new MMS messages

An example for the request for fetching new MMS messages is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-new";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for fetching new MMS messages are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there are no new messages

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>POSTPAID</credits>
</inboxlist>
```

If there are new **MMS** messages

```
<inboxlist xmlns="">
    <result>OK</result>
```

available (the new messages are fetched with the text and then marked as read, the number of message blocks equals the number of messages).

```
    <message id="082bbe87-4b76-4cfb-b4ef-236daf429a04" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>4</size>
        <subject>Meet me at airport</subject>
        <msgtype>MMS</msgtype>
        <text>Road map to airport</text>
        <content>UEsDBBQAAAAIAB1YlzpRB98gKCcBALonA...</content>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the size in KB for the MMS messages. The Content node for the MMS message in the example above contains a Base64 encoded zipped smil file and the multimedia content (the string shown incomplete).

IX.2.d. Fetching MMS messages by their IDs

An example for the request for fetching MMS messages by the IDs is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "fetch-id";
String messageID = "b7d7508c-0d68-4e0f-acaa-6f2ec27947e7";

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Please note that in case there are several MMS message IDs, they must be separated by semicolon.

Examples of possible responses for fetching MMS messages by the IDs are provided below:

If the user is unknown

```
<inboxlist xmlns="">
    <result>404.2 FAILURE (User is unknown)</result>
</inboxlist>
```

If there is no messages with this ID

```
<inboxlist xmlns="">
    <result>OK</result>
    <credits>88000</credits>
</inboxlist>
```

If there is the **MMS** message with the ID specified (the **MMS** message is fetched with the content and then marked as read, the number of message blocks equals the number of messages).

```
<inboxlist xmlns="">
    <result>OK</result>
    <message id="082bbe87-4b76-4cfb-b4ef-236daf429a04" new="1">
        <from>971505000001</from>
        <to>fdm</to>
        <timestamp>20090421172256</timestamp>
        <size>4</size>
        <subject>Meet me at airport</subject>
        <msgtype>MMS</msgtype>
        <text>Road map to airport</text>
        <content>UEsDBBQAAAAIAB1YlzpRB98gKCcBALonA...</content>
    </message>
    <credits>POSTPAID</credits>
</inboxlist>
```

The Size node contains the number of characters for the MMS messages. The Content node for the MMS message in the example above contains a Base64 encoded zipped SMIL file and the multimedia content (the string shown incomplete).

X. Deleting Inbox SMS/MMS messages

To delete SMS/MMS messages, the ***InboxProcessing*** function is used. The corresponding operations with the 'kill' prefix are:

- kill-all;
- kill-old;
- kill-id.

The operations listed above delete all messages, messages which have been fetched, or messages with specified IDs respectively. If there are several IDs specified, they must be separated by semicolon.

The example provided below is the ***InboxProcessing*** function prototype:

```
InboxProcessing(SoapUser user, String operation, String messageId)
```

The required parameters for the ***InboxProcessing*** function are:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.
- operation—the operation to be performed with MMS messages (the 'kill' prefix is used for the current operations).
- messageId—the unique identifier for the message, to be used for the 'kill-id' operation only. If there are several identifiers, they must be separated by semicolon.

You can also use the ***HTTP_InboxProcessing*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/SOAP/Messenger.asmx/HTTP_InboxProcessing?customerID=string&userName=
     string&userPassword=string&operation=string&messageId=string
HTTP/1.1
```

X.1.a. Deleting all messages

To delete all messages, the ***kill-all*** operation is used. An example for the request is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "kill-all";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                        messageID);
```

An example of a possible response for deleting all messages is provided below:

```
<inboxdel xmlns="">
    <result>OK</result>
    <deleted>17</deleted>
    <credits>POSTPAID</credits>
```

```
</inboxdel>
```

The Deleted node contains the number of messages deleted.

X.1.b. Deleting old messages

To delete old messages, the ***kill-old*** operation is used. An example for the request is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "kill-old";
String messageID = String.Empty;

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

An example of a possible response for deleting old messages is provided below:

```
<inboxdel xmlns="">
    <result>OK</result>
    <deleted>10</deleted>
    <credits>POSTPAID</credits>
</inboxdel>
```

The Deleted node contains the number of messages deleted.

X.1.c. Deleting messages by their IDs

To delete messages by their IDs, the ***kill-id*** operation is used. An example for the request is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 2;
user.Name = "q";
user.Language = "en";
user.Password = "q";

String operation = "kill-id";
String messageID = "b7d7508c-0d68-4e0f-acaa-6f2ec27947e7";

Messenger.Messenger messengerService = new Messenger.Messenger();
XmlNode xmlResponse = messengerService.InboxProcessing(user, operation,
                                                       messageID);
```

Examples of possible responses for deleting messages by their IDs are provided below:

If the message ID format is correct

```
<inboxdel xmlns="">
    <result>OK</result>
    <deleted>1</deleted>
    <credits>POSTPAID</credits>
</inboxdel>
```

If the message ID format is incorrect

```
<inbox xmlns="">
    <result>605.6 The action you requested cannot be
performed,
          because one of your request parameters (msgid ) is
                incorrect. Please verify it and try
again.</result>
</inbox>
```

XI. Sending Keep Alive Signals

The ***KeepAlive*** function periodically sends 'keep alive' signals from the application to the server, in order to prevent MESSAGEMANAGER PLATFORM falling into the sleeping mode.

The example provided below is the ***KeepAlive*** function prototype:

```
KeepAlive(SoapUser user)
```

The required parameters are:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- Language—the language in which the client application is going to communicate with the user.
- userPassword—the password of the user specified.

An example of calling the ***KeepAlive*** function is provided below:

```
SoapUser user = new SoapUser();
user.CustomerID = 1;
user.Name = "Bob";
user.Language = "en";
user.Password = "secret";
CommonResult commonResult = messenger.KeepAlive(user);
Console.WriteLine(commonResult.Result);
```

An example of responding to the ***KeepAlive*** function is provided below:

```
<?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>
    <KeepAliveResponse xmlns="http://pmmsoapmessenger.com/">
      <KeepAliveResult>
        <Result>string</Result>
      </KeepAliveResult>
    </KeepAliveResponse>
  </soap:Body>
</soap:Envelope>
```

You can also use the ***HTTP_KeepAlive*** alias to call this function without using SOAP, by the means of pure HTTP GET, as shown in the example below:

```
GET /MessageManager
Platform/soap/Messenger.asmx/HTTP_KeepAlive?customerID=string&userName=string
&userPassword=string

HTTP/1.1
```

XII. Sending Service SMS messages

With the help of the ***SendServiceSms*** function the customer can send service SMS messages using the specified parameters.

The example provided below is the ***SendServiceSms*** function prototype:

```
SendServiceSms(Int32 customerID, String userName, String userPassword, String
originator, String serviceName, string serviceUrl, String recipientPhone,
MessageType messageType, String defDate, Boolean blink, Boolean flash,
Boolean Private)
```

The required parameters are:

- customerID
- userName—the customerID

The unique identifier for every customer in the Message Manager Platform database.

- customerUserName for the user for whom the list of originators is requested.
- userPassword—the password of the user specified.
- originator—the parameter identical to originatorName
- serviceName—the name of the service.
- serviceUrl—the URL of the service.
- recipientPhone—the recipient's phone numbers separated by commas (the parameter can contain just one number).
- messageType—the type of the character set used in the SMS text (Latin, Arabic with Latin numbers, Arabic with Arabic numbers).
- defDate—the parameter specifying the date and time of deferred delivery (must be in the **yyyyMMddhhmmss** format).
- flash—the parameter defines whether the SMS message is to be saved in the recipient phone inbox (True—the message is not saved, False—the message is saved).
- blink—the parameter defines whether the SMS message blinks when onscreen (applicable for some makes of Nokia mobile phones).
- private—the parameter defines whether the SMS message appears in the account statement.

An example of calling the ***SendServiceSms*** function is provided below:

```
int id = 2;
string userName = "Bob";
string password = "secret";
string originator = authResult.Originators[0];
string defDate = DateTime.UtcNow.AddHours(1).ToString("yyyyMMddhhmmss");
string smsData = "test";
string phone = "971505000001";
string serviceName = "bar";
string serviceUrl = "http://url/mm/soap/messenger.asmx";
messenger.SendServiceSms(id, userName, password, originator, serviceName,
       serviceUrl, phone, MessageType.Latin, defDate, false, false, false);
```

An example of responding to the ***SendServiceSms*** function is provided below:

```
<?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>
    <SendServiceSmsResponse xmlns="http://pmmsoapmessenger.com/">
```

```
      <SendServiceSmsResult>
        <RejectedNumbers>
          <string>string</string>
          <string>string</string>
        </RejectedNumbers>
        <TransactionID>string</TransactionID>
        <NetPoints>string</NetPoints>
      </SendServiceSmsResult>
    </SendServiceSmsResponse>
  </soap:Body>
</soap:Envelope>
```

XIII. Error Replies

With the help of the ***SendServiceSms*** function the customer can send service SMS messages using the specified parameters.

Error Message	Description
401.1 FAILURE (You have exceeded the allowed number of unsuccessful login attempts. Please contact your Customer Service to continue.)	Server will return this message in case of authentication when user exceed maximum of failed logins.
403.2 FAILURE (User account is blocked or deactivated).	This is the message that the server will return if the user account is blocked.
403.3 FAILURE (Customer is blocked or deactivated)	This is the message that the server will return if the customer is blocked.
403.4 FAILURE (Customer's prepaid credit is insufficient).	This is the message that the server will return in case of customer's prepaid credit is insufficient.
403.5 FAILURE (Customer has no Originator values configured)	Server will return this message if no originator is configured for the current customer.
404.2 FAILURE (User is unknown)	This is the message that the server will return if the user is unknown.
500.1 FAILURE (Server error: user/customer link broken)	This is the message that the server will return in case of a broken relation between the user and the customer.
500.2 FAILURE (Dealer is not found)	Server will return this message if the dealer is not found.
600.1 Originator for CustomerID is not found	This is the message sent to the MM user if the originator of the submitted message was not found in the list of the given Customer ID originators.
600.4 Phone not specified	This is the message sent to the user when the phone number is not specified.
600.9 Customer cannot send via web service	Error will be replied if customer cannot send via web service.
601.1 Length of the originator must be up 11 characters	This is the message displayed to the user when the originator’s name exceeds its maximum length.
601.2 Originator has bad chars	This message is sent to the user when the originator's name contains the symbols that

	cannot be used. Used by MM Platform handler.
601.6 MMS data is not posted	This message is replied to the user in case there is no MMS data to send via MMS 2 Group.
601.7 VAS-ID must be numeric	Error will be sent to user when he sends MM7 with incorrect VAS-ID
601.8 ChargedParty must be Sender	Error will be sent to user when he sends MM7 with incorrect ChargedParty
603.1 Sorry, your request cannot be processed. Server is busy. Please try again later.	Client will receive that error if server is busy and cannot process request.
603.3 FAILURE (You cannot access this Interface. Contact your Admin for details.)	Server will return this message if authentication failed because of customer has no access to interface.
605.6 The action you requested cannot be performed, because one of your request parameters incorrect. Please verify it and try again.	This message is returned as response from MessageManager handlers if request parameter format is incorrect.
605.7 The action you requested cannot be performed, because one of your the required request parameters was not supplied.	This message is returned as response from MessageManager if a required request parameter is not supplied.
606.5 Sorry, you cannot send SMS or MMS messages, because the corresponding feature is not activated for you. Please contact your Administrator if you want to have this feature available for you.	This message is returned as error during message sending processing if corresponding feature is not activated for customer. .
606.8 Your message is rejected, because at least one of the recipient numbers you specified exceeds 20-digit limit. Please check it and try again. If necessary, put the delimiter sign for the numbers.	This message is returned as the error from MessageManager during message sending processing if one of recipient phones length is more than maximum allowed.
1000 An error has occurred in Message Manager Platform. The administrator has been already notified. We are sorry for any inconvenience this might have caused you.	This is the message content that the user receives in case an error occurs in the MM Platform server.
2108 You cannot send messages using this Originator, because the Originator has not yet been approved. Please contact your Admin for details.	This is the message sent as a response if the originator has the 'Pending' status and the 'Trusted Originator' feature is not activated. Used by the request handler.
MM7 file contains corrupted base64 content.	This message will be sent to the user when MessageManager receives corrupted base64 content of the mm7 file via soap.
Your distributor was blocked or deactivated.	This is content of error page in case of distributor was blocked or deactivated and try to login in Web UI. Used by MM platform in distributor Web UI.
Your current message cannot be sent now, because another message of yours is being processed. Please try again in a few minutes.	This message is returned from MessageManager if a similar message is now being processed.

FAILURE: All recipient numbers in your message are either Rejected or Blacklisted	This message is returned from MessageManager if all recipients of the message were either rejected or blacklisted. .
Message ID not specified	This is the message sent to the user when the server failed to recognize the ID of the submitted message.
SMS ID for status request is incorrect or not specified	This is the message sent to the user when the server fails to receive or recognize the SMS ID due status request.
Version not specified	This is the message sent to the user when the version of the Message Manager is not specified. .
ClientID not specified	This is the message sent to the user when the CustomerID is not specified.
It's illegal to use MM Client software for sending messages.	This is content of error which will return to MM client software in case of sending via MM operations is disabled.
UserID not specified	This is the message sent to the user when the username ID is not specified. .
Pass not specified	This is the message sent to the user when the password is not specified.
The message cannot be sent by SMS because the encoding is incorrect. Please change the message language.	This is the error notification that will be displayed if the message encoding is incorrect.
The Messaging Platform cannot send your message, because the text in your message is unacceptable. Please make it more appropriate and try again.	This is the error notification that will be if SMS text or mms subject contains unacceptable words.
The Server cannot process your message now, because the delivery time you specified comes to the blockout period. Please set your delivery time within the nearest sending period starting at …	This message will be sent to client if he tries to send SMS/MMS during block out period.
Not Enough Money For Prepaid Customer	This is the message sent to the MM User when the user's message cannot be delivered, as there is no money on the Customer's account.
You cannot send SMS messages because your sms limit exceeded. Contact your Admin for details.	This is the message sent as a response if a customer SMS limit exceeded.
You cannot send MMS messages because your mms limit exceeded. Contact your Admin for details.	This is the message sent as a response if a customer mms limit exceeded.