myMSP API v2 Appendix Web service

System Interface - Specifications

21st Century Mobile
Marketing & Sales
+46 (0)8 21 21 55
sales@21st.se

www.21st.se
Research & Development
+46 (0)8 21 21 55
tech@21st.se

1 Purpose of Document

The purpose of this document is to define the web service interface made available by myMSP to access the application functionality. Accessing myMSP can be done from the computer systems of external parties/customers.

1.1 Author

21st Century Mobile
Mikael Rosvall
mikael.rosvall@21st.se
+46 (0)8 21 21 55

1.2 Document Status

- Published 2007-01-08

1.2.1 Document Changes

- B [2007-06-15]:
  - Java code example modified.
  - Minor changes and improvements
- C [2007-07-17]
  - New method added: Retrieve send results
    - 3.4: usage scenario
    - 4.5: method description
  - Added *recommended usage patterns* to some method descriptions in chapter 4.
- D [2007-08-20]
  - Added a note that the maximum length of a sender alias is ten characters.
- E [2007-12-11]
  - Updated the headers and footers with new information.
  - Added a note that the maximum length of a sender alias is eleven characters.
- F [2008-03-28]
  - Updated all URL-references from the old domain (www.mymsp.eu) to the new domain (mymsp.21st.se).
- G [2012-01-24]
  - Added a new section: 4.3.2 Additional SMS parameters.
- H [2012-12-14]
  - Removed parameter “Charset” from section 4.3.1.
- I [2019-03-11]
  - Updated graphic profile.
  - Deprecated MMS methods.

1.3 Document Perimeter

Neither economical questions nor questions regarding agreements/contracts will be dealt with in this document.

The information given in this document may change without notice and describes only the matters defined in the general part of this document. Please verify that your company has the most recent version. This information is intended for the use of customers and partners of 21st Century Mobile. The information or statements given in this document concerning the suitability, capacity or performance of the mentioned

service cannot be considered binding but shall be defined in the agreement concluded between 21st Century Mobile and the customer, if applicable. 21st Century Mobile shall not be responsible in any event for errors in this document or for any damages, incidental or consequential (including monetary losses), that might arise from the use of this publication or the information in it. This material and the service described in this document are copyrighted in accordance with the applicable laws.

2 Technical description

This interface is realized through Web Services. Web Services are software systems that are designed to support machine-to-machine communication over a network, e.g. the Internet. The information exchange is carried out using SOAP-formatted XML-envelopes that are sent through the HTTP-protocol. The web service URL for accessing myMSP is:

- https://mymsp.21st.se/external/ws2/myMSP

WSDL (Web Services Description Language) is used to describe the system interface, i.e. what services are available and how they are accessed. WSDL is constructed to enable an automatic information and service exchange between computers. To hide the complexity of invoking the web services a development tool (e.g. Microsoft Visual Studio, Eclipse, Netbeans) is normally used to automatically generate the required parameter classes as specified by the WSDL file. It is important to note that the address to the WSDL-file should only be used to generate the required parameter classes in the development tool. It should not be used access the web service itself. The WSDL file that describes the myMSP web service can be found at:

- https://mymsp.21st.se/external/ws2/myMSP?wsdl

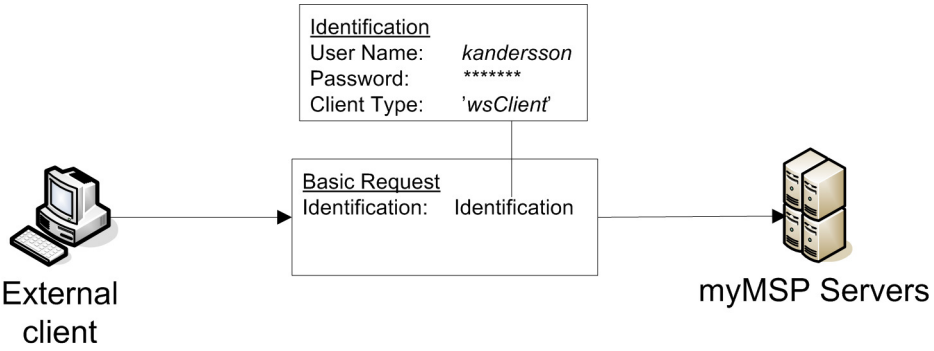
3 Usage scenarios

The following scenarios are intended to give a brief overview of how to use the myMSP web services. In the next chapter the myMSP Web Services are described in more detail.

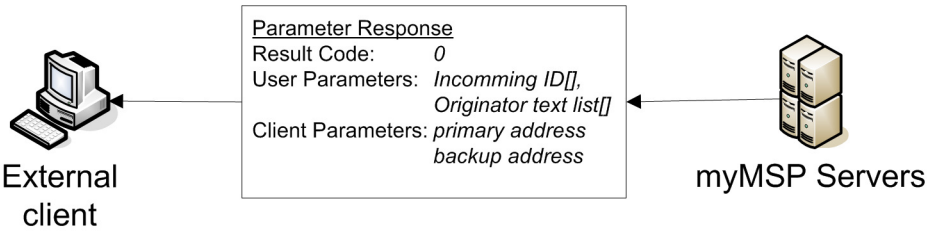
3.1 Setup communication

Before a client that utilizes the myMSP web service can send and receive messages the user identification has to be validated and some configuration information needs to be fetched.

1. In order to initialize communications with myMSP a *basic request*, including user identification and type of client, is used to fetch setup parameters:



2. The *response* includes two sets of parameters: *user-* (e.g. allowed originator texts and incoming ids) and *client-parameters* (e.g. primary address to be used when communicating with the web service and a backup address that can be used if the primary address is inaccessible).

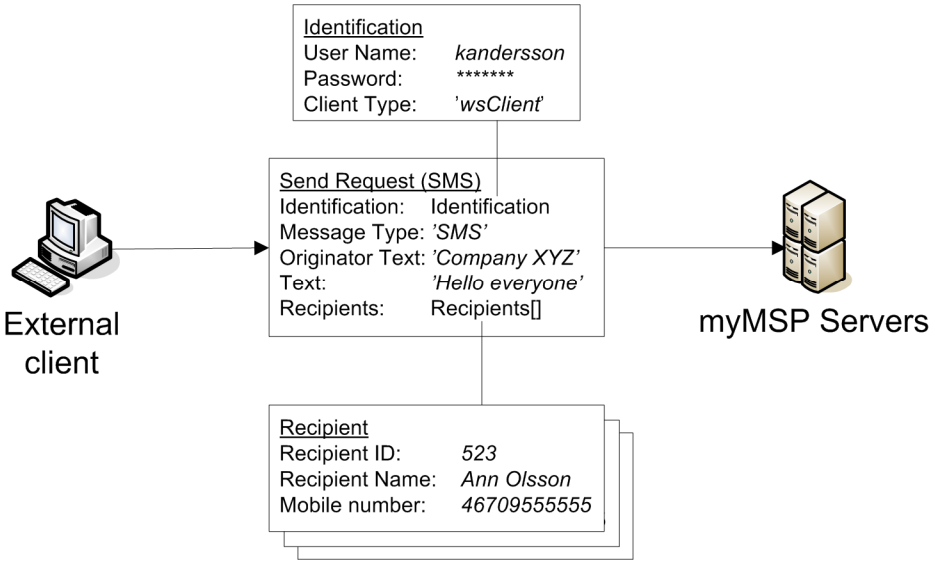


After the client has sent a request to myMSP and has received a response with a *result code* of 0 (indicating that the web service address, user name, and password are all valid) the actual sending and receiving of messages can commence.

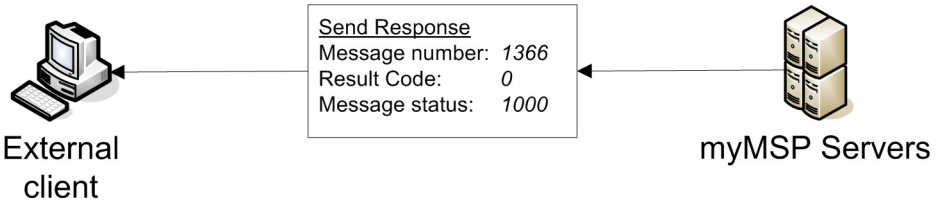
3.2 Send a message

As previously mentioned in the main API-document (3.1 The send message sequence) sending a message through myMSP is done in steps.

1. First, a message *send request* is sent to myMSP. The request contains user identification, a list of *recipients*, and the message content.



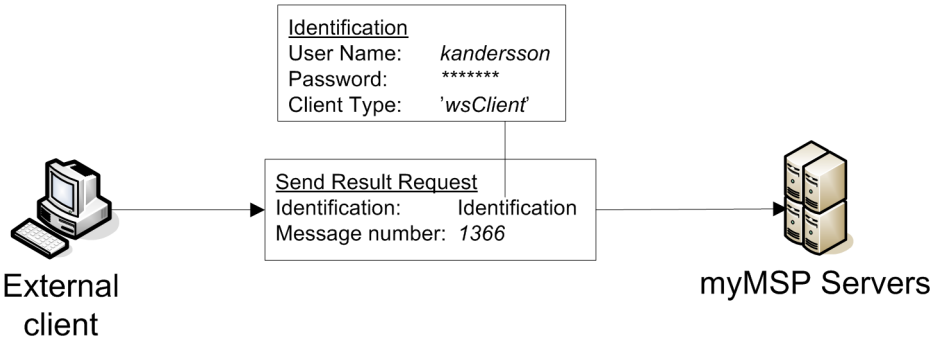
2. After the send request has been sent, a *send response* is returned containing, among other things, the *status* of the message, i.e. how far it has come in the transmission sequence.



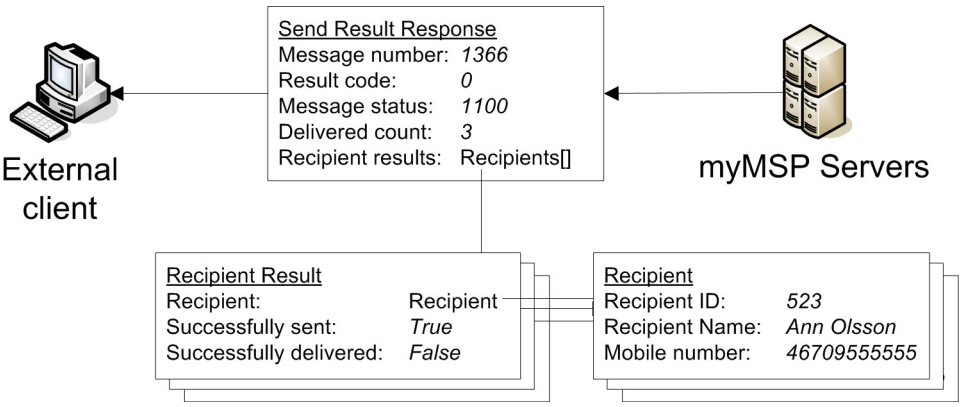
3.3 Check the status of a sent message

It can take some time before the message is delivered to all recipients. The status of a message can be checked while it is being delivered.

1. To check the current status of a message a *send result request* has to be sent to myMSP. The request is composed of user identification, and the message number of the message whose status should be checked.



2. myMSP returns a response containing the status of the message (i.e. how far it has come in the transmission sequence). The result response also contains information regarding the send status for each recipient, e.g. if the message has been successfully sent and delivered to a specific recipient.

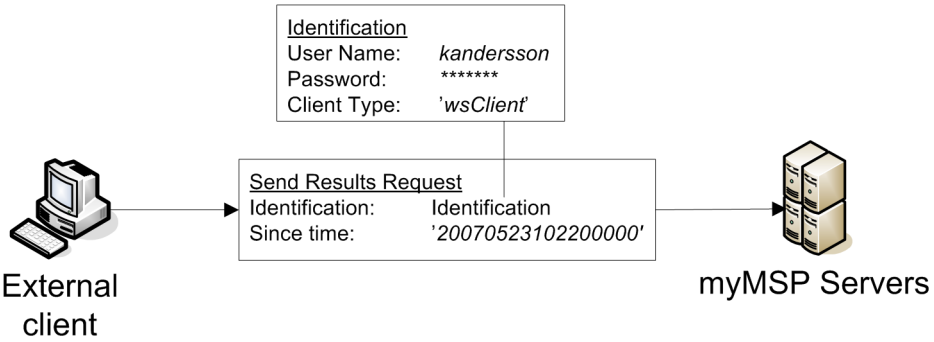


The process checking the send status of a message can be repeated as many times as is desirable, e.g. to check and make sure that all recipients have received the message.

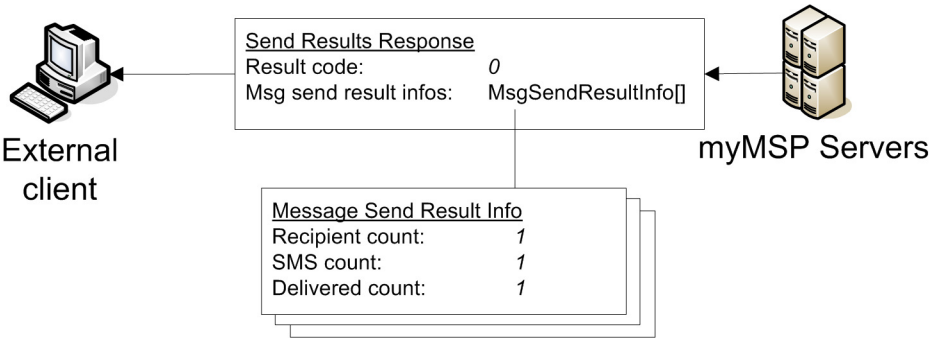
3.4 Check the status of several sent messages

Sometimes it is preferable to check the status of several sent messages at once. Use this method to reduce the number of requests sent to myMSP, e.g. if you regularly send many messages with few recipients per message.

1. To check the current status of several messages a *send results request* has to be sent to myMSP. The request is composed of user identification and a time stamp.



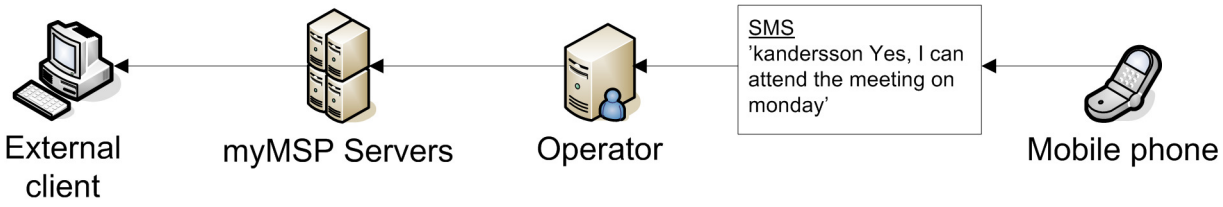
2. myMSP returns a response containing the aggregated send status (e.g. recipient count, sent count, delivered count) of those messages whose status has changed since the time that was specified in the request. However, the response does not contain information regarding the send status of the each recipient of each message. Submit a *send result request* (see 3.3) in order to check the send status of each recipient of a message.



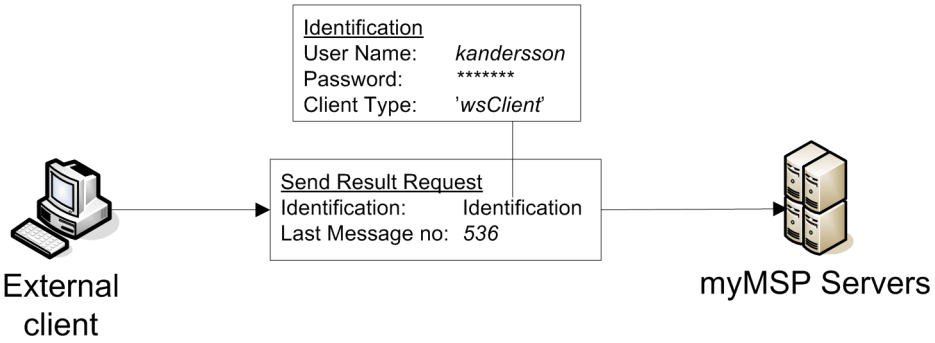
3.5 Receive a message

myMSP can also receive messages from mobile phones.

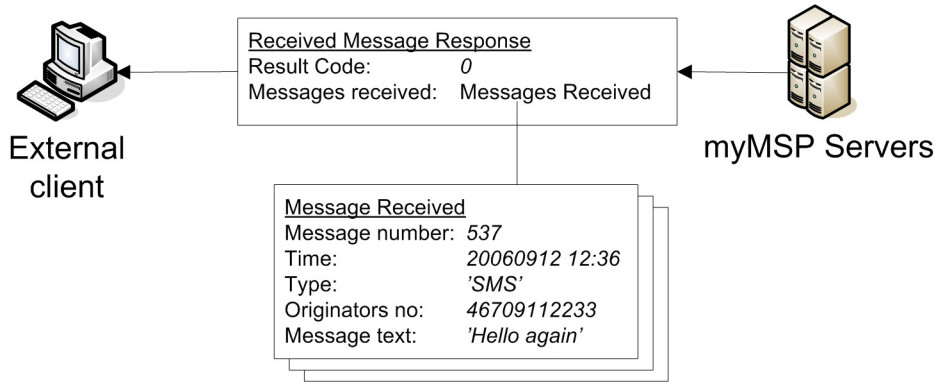
1. An initial ID is needed in order to send a message to a myMSP account from a mobile phone. The initial ID is placed in the beginning of the text in the message and should be followed by a blank space.



2. A *received message request* is needed in order to get new messages that have been sent to a myMSP account.



3. A *received message response* is returned by myMSP containing a set of received messages.



4 Web service methods

This chapter describes the various methods of the myMSP Web service in more detail.

4.1 Control Connection (ping)

Checks if there is a connection to the myMSP-server. The input and output parameters are pretty much dummies.

Recommended usage pattern: ping is useful when developing your application and later when initializing your application in order to check that you connection to myMSP is working properly.

4.1.1 Input Parameters (String)

Who	A text string

4.1.2 Output Parameters (String)

pingReturn	A text string with the value "Alive"

4.2 Retrieve parameters (getParams)

The method makes it possible to customize the remote application based on the current user and client type. The input parameter is a basic request object that contains an instance of the identification class (4.2.2). The identification class is included in all requests to myMSP, accept for the ping.

Recommended usage pattern: Since parameters rarely change they need only be fetched about once a day and when initializing your application.

4.2.1 Input Parameters (BasicRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).

4.2.2 Input class Identification

Username	User Account
Pwd	Password
clientType	Type of client (use 'wsClient')
clientVersion	Not applicable

4.2.3 Output Parameters (ParamsResponse)

resultCode	See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
userParams	An array of parameters (Param) for the user. A parameter contains a key and a value.

clientParams	An array of parameters (Param) for the client. A parameter contains a key and a value.
originatorTexts	An array of allowed optional sender aliases (se 'Originator text' in main API-document glossary) to be displayed instead of the originating number.
initialIds	An array of initial ids. An initial id has to be included when a recipient replies to a message sent through myMSP. This enables myMSP to direct the incoming message to the correct user account.

Example of user parameters:

- locale (defines the users country and language 'sv_SE', 'us_EN')
- charset (the users preferred character encoding, 'ISO-8859-1, 'UTF-8')

Example of client parameters:

- primaryAddress (primary address to web service)
- backupAddress (secondary address to web service)
- homepage (Web address to myMSP home page with production status)

4.3 Send Message (sendMsg)

Send a message to recipients. Recipients are sent in the form of an array of recipient objects (4.3.2).

4.3.1 Input Parameters (SendRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).
msgType	Type of message ('SMS')
originatorText	Originator text of the message (see 'Originator text' in main API-document glossary). Note that only pre-registered originator texts acquired through the 'getParams' request (userParams in 4.2.3) can be used. The maximum length of an alias is eleven (11) characters.
Recipients	An array of recipients (see 4.3.2 class Recipient below).
smsText	Text Message. Maximum of 765 characters for SMS. See also 4.3.2 on how to add additional SMS parameters to the message using smsText.
subject	[deprecated]
Contents	[deprecated]
externalRef	An external reference for the user of myMSP. Statistics may be available based on this reference.
requestReadReceipt	[deprecated]

4.3.2 Additional SMS parameters

It is possible to set some additional parameters when sending an SMS. This can be done by adding parameter tags at the beginning of the message text ("smsText" in 4.3.1) and placing the actual parameters and their values in between: <param>param1=value1&param2=value2</param>. All parameters and the start and stop tags are removed from the message text once they have been processed and do not add to the total number of characters in the message.

timetolive	How long the SMS should "live", i.e. how long the operator should attempt to send

	the message to a mobile phone before giving up and deleting it. The default value is 72 hours. A timetolive value consists of a number and one of three time units: minutes (m), hours (h) and days (d). The max value is 3 days, and the min value is 5 minutes. Time units cannot be mixed, e.g. a value of one and a half hours should be set in minutes (90m), NOT hours and minutes (1h30m).
deliverytime	The date and time that the message should be delivered. Should be formatted as follows: yyMMddHHmmsszzzz, e.g. 120125183525004+. The last field (zzzz) is the time zone difference in quarters of an hour from Western European Time (UTC+0) i.e. GMT. A value of 000+ or 000- equals GMT, 004+ equals Central European Time (UTC+1), etc.
isflash	Set to "true" if the message should be sent as a flash SMS. A flash SMS is displayed in the recipient's mobile phone as soon as it is delivered, i.e. does not have to be opened by the user. It is also deleted immediately after being read, although some mobile phones allow the user to save the message to their SMS inbox.

Example: to send a flash SMS with the text "This is a test message" that should live for 15 minutes the "smsText" parameter should be set to: "<param>timetolive=15m&isflash=true</param>This is a test message".

4.3.3 Input class Recipient

recipientId	The myMSP internal ID of the recipient
recipientName	The name of the recipient
mobileNumber	The mobile number of the recipient
externalRef	An external reference for the recipient in myMSP. If the user of myMSP services uses another way of identifying a recipient, other from recipientName or mobileNumber, it can be specified here.

4.3.4 [deprecated] Input class MessageContent

slideNo	[deprecated]
duration	[deprecated]
contentType	[deprecated]
contentCharset	[deprecated]
contentBytes	[deprecated]

4.3.5 Output Parameters (SendResponse)

resultCode	All results other then zero means that there has been an error. See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
msgNo	Message identification number in myMSP.
msgStatus	Message Status (state) in myMSP. See message status in main API-document (3.2)!
createTime	The time (yyyyMMddHHmmssSSS) the message was created

4.4 Retrieve Result (getSendResult)

Check the send result of a message. The individual delivery status for each recipient is included in an array of recipient result-objects (4.4.3).

Recommended usage pattern: Typically, the send result of a message changes frequently at first (1-2 min) and later less frequently. It is therefore recommended that the send status of a message be check less frequently the longer it has been since the message was sent.

4.4.1 Input Parameters (SendResultRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).
msgNo	Message identification number in myMSP.

4.4.2 Output Parameters (SendResultResponse)

resultCode	See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
msgStatus	See message status in main API-document (3.2)!
sendRequestTime	The time (yyyyMMddHHmmssSSS) the message was requested to be sent.
recipientCount	The number of recipients of the message.
sentOkCount	The number of messages that where successfully sent to the operator.
deliveredOkCount	The number of successful deliveries to recipients.
readOkCount	[deprecated]
recipientResults	A result per recipient represented by an array of the class RecipientResult described below.

4.4.3 Output class RecipientResult

recipient	A class containing the recipient information (see 4.3.2 class Recipient).
operatorResultCode	The result code (error code) from the operator. Provided as guidance for an eventual correction of the message.
operatorResultDescription	The result description (error text) from the operator. Provided as guidance for an eventual correction of the message.
sentOk	Indicates if the operator has accepted the message and the recipient.
sentTime	The time (yyyyMMddHHmmssSSS) the message was accepted by the operator.
deliveredOk	Indicates if the operator has successfully delivered the message to the recipient.
deliveredTime	The time (yyyy-MM-dd HH:mm) the message was delivered to the recipient (mobile phone).
readOk	[deprecated]
readTime	[deprecated]

4.5 Retrieve Results (getSendResults)

Check the send result of several messages.

Recommended usage pattern: If your application normally sends many messages with a few recipients each (e.g. 100 messages a day with one recipient per message) then it is usually better to send a *sendResultsRequest* (that fetches a batch of send results) every few minutes instead of sending a *sendResultRequest* for each message.

4.5.1 Input Parameters (SendResultsRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).
sinceTime	A time stamp, format: yyyyMMddHHmmssSSS. All send status changes that have occurred after this time will be returned.

4.5.2 Output Parameters (SendResultsResponse)

resultCode	See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
msgSendResultInfos	An array of the class MsgSendResultInfo described below.

4.5.3 Output class MsgSendResultInfo

msgNo	Message identification number in myMSP.
sendRequestTime	The time (yyyyMMddHHmmssSSS) the message was requested to be sent.
recipientCount	The number of recipients of the message.
sentOkCount	The number of messages that where successfully sent to the operator.
smsCount	myMSP splits long SMS (>160 characters) into several SMS which are sent separately and later reassembled in the recipient's mobile phone. smsCount specifies the total number of SMS that are sent. For example: an SMS with a 170 characters long text and two recipients will generate a smsCount of four (2 recipients x 2 text segments). The smsCount for normal SMS (=<160 characters) is always the same as the recipient count.
deliveredOkCount	The number of successful deliveries to recipients.
readOkCount	[deprecated]

4.6 Retrieve Received Messages (getReceivedMessages)

The method fetches received messages.

4.6.1 Input Parameters (ReceivedMsgRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).
lastMsgNo	The message identification number of the message last received, input 0 to retrieve all.

4.6.2 Output Parameters (ReceivedMsgResponse)

resultCode	See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
messagesReceived	An array (MessageReceived []) that contains received messages. The class "MessageReceived" is described below.

4.6.3 Output class MessageReceived

msgNo	Message identification number in myMSP.
createTime	Time and date when the message arrived in myMSP.
creatorName	Name of the myMSP user that received the message.
initialId	The initial Id used to identify the recipient of the incoming message.
msgType	Type of message ('SMS')
originator	The originators mobile number
originatorText	Originator text of the message. See glossary above.
smsText	The SMS text
subject	[deprecated]

4.7 [deprecated] Retrieve Message Content (getMsgContent)

Deprecated method for getting the content of a MMS message.

4.7.1 Input Parameters (MsgContentRequest)

Identification	A class containing information that identifies the user and the type of client that is used (see 4.2.2 Identification).
msgNo	Message identification number in myMSP

4.7.2 Output Parameters (MsgContentResponse)

resultCode	See result codes in main API-document (3.3)!
resultDescription	Text describing the results.
contents	[deprecated]

5 Examples

The following code examples illustrate how to use the parameter classes in a development environment. More complete code samples are also available for Visual Studio 2005 and for Java development environments. The samples are meant to illustrate the how to use the functionality of the Web Service, i.e. creating requests, sending requests, receiving responses, reading the data in the responses, and displaying the results. You are free to view, modify, copy and use the code in any way that you like. The code samples can be downloaded from our file library. Go to https://mymsp.21st.se/app and log in to access the library.

Note that the myMSP support classes that are used in the code examples have been automatically generated by a development environment using the myMSP WSDL-file (see chapter 2). These classes look a bit different depending on which development environment is used.

5.1 C#

```
private void sendSMS()
{
	// The web service connection-object
	eu.mymsp.www.MyMSP2 _webService = new eu.mymsp.www.MyMSP2();

	// Create and fill a identification object
	eu.mymsp.www.Identification id = new eu.mymsp.www.Identification();
	id.username = "johnDoe";
	id.pwd = "xyz123";
	id.clientType = "wsClient";
	id.clientVersion = "";

	// Create a recipient array
	eu.mymsp.www.Recipient[] recipients = new eu.mymsp.www.Recipient[1];

	// Create and fill a recipient object
	eu.mymsp.www.Recipient recipient = new eu.mymsp.www.Recipient();
	recipient.mobileNumber = "46709888888";
	recipient.recipientName = "Jane Doe";
	recipient.externalRef = "";

	// Add the recipient to the recipient array
	recipients[0] = recipient;

	// Create a send request object
	eu.mymsp.www.SendRequest sendRequest = new eu.mymsp.www.SendRequest();

	// Fill the send request
	sendRequest.identification = id;
	sendRequest.recipients = recipients;
	sendRequest.msgType = "SMS";
	sendRequest.smsText = "Hello, this is a test message.";
	sendRequest.requestReadReceipt = true;
	sendRequest.subject = "";
	sendRequest.originatorText = "";
	sendRequest.externalRef = "";
```

```
        sendRequest.contents = null;

        // Create a sendMsg object that will contain all the data that should
        // be sent to the web service
        eu.mymsp.www.sendMsg sendSms = new eu.mymsp.www.sendMsg();

        // Add the send request to the sendMsg-object
        sendSms.SendRequest_1 = sendRequest;

        // Create a response object that stores the result of the send-attempt
        eu.mymsp.www.sendMsgResponse response = null;

        try
        {
            // Send the message
            response = _webService.sendMsg(sendSms);
        }
        catch
        {
            response = null;
        }
}
```

5.2 Java

The following sample code was created using NetBeans 5.5 and depends on support classes generated by the Web Service framework JAX-WS (a NetBeans wizard). The support classes might look different if you use another development environment (e.g. Eclipse) and/or Web Service framework (e.g. JAX-RPC, Axis) to generate them. For more information we refer you to the support documents of your development environment that details how to create a Web Service client.

```
private void sendSMS() throws Exception {
        // The web service connection-object
        eu.mymsp.www.MyMSPInterface1 webServicePort =
            new eu.mymsp.www.MyMSP2().getMyMSPInterface1Port();

        // Create and fill a identification object
        eu.mymsp.www.Identification id = new eu.mymsp.www.Identification();
        id.setUsername("johnDoe");
        id.setPwd("xyz123");
        id.setClientType("wsClient");
        id.setClientVersion("");

        // Create and fill a recipient object
        eu.mymsp.www.Recipient recipient = new eu.mymsp.www.Recipient();
        recipient.setMobileNumber("46709888888");
        recipient.setRecipientName("Jane Doe");
        recipient.setExternalRef("");

        // Create a send request object
        eu.mymsp.www.SendRequest sendRequest = new eu.mymsp.www.SendRequest();

        // Fill the send request
        sendRequest.setIdentification(id);
```

```
        sendRequest.getRecipients().add(recipient);
        sendRequest.setMsgType("SMS");
        sendRequest.setSmsText("Hello, this is a test message.");
        sendRequest.setRequestReadReceipt(true);
        sendRequest.setSubject("");
        sendRequest.setOriginatorText("");
        sendRequest.setExternalRef("");

        // Create a response object that stores the result of the send-attempt
        eu.mymsp.www.SendResponse response = null;

        try {
            // Send the message
            response = webServicePort.sendMsg(sendRequest);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw ex;
        }

        System.out.println(response.getMsgNo());
        System.out.println(response.getMsgStatus());
        System.out.println(response.getResultCode());
        System.out.println(response.getResultDescription());
        System.out.println(response.getCreateTime());
}
```