# SMS Gateway - Interface XML-RPC

## Table of contents



*Please note that the latest version of this document is always available on https://sms.netoxygen.ch/api2/documentation/ ; check the version number in the footer to know if you are already using the latest version available.*

# Available methods

The URL for the server is: https://sms.netoxygen.ch/api2/ (or http://sms.netoxygen.ch/api2/).

## *Send an SMS*

**method:** send_message
**parameters:** user, pass, dest, message[,sender,date,type]
**returns:** an array of arrays

→ the 3rd parameter can be either a string or a table containing several strings, one per recipient.

→ a 5th optional parameter makes it possible to define the sender. If the field is empty or the current user is not allowed to change the sender the default sender for this user is used. Otherwise, if no default sender is set, “textobox.ch” is used.

→ a 6th optional parameter makes it possible to define the date and time when the message should be sent (reference timezone is: Europe/Zurich). If this parameter is empty, contains an invalid date or a date in the past, the message is sent immediately. The date format is ISO format, eg: “2006-12-06 20:05:00”. To specify a delivery date while using the default sender, simply give an empty string as the 5th parameter.

→ a 7th optional parameter defines de type of message. It can currently take two values: “text“ or “flash“. By default, if nothing is defined, the type is set to “text”. A “flash” SMS is an SMS that appears directly on the terminal of the recipient without requiring the user to “open” the SMS.

→ the returned array contains for each recipient a unique identifier and the message status. The identifier can be used later to query the status of each message.

## *Check a user/password pair*

**method:** auth
**parameters:** user, pass
**returns:** TRUE/FALSE

## *Request the amount of available credits*

**method:** get_credits
**parameters:** user, pass
**returns:** the amount of available credits or FALSE

## *Query the account status*

**method:** account_status
**parameters:** user, pass
**returns:** 'regular' or 'free'

### *Get the user's « group_id »*

**method:** get_guid
**parameters:** user, pass
**returns:** a unique group (account) identifier to which belongs the user or FALSE

→ This identifier is a unique string of 36 characters in the UUID format aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee.

→ This method is primarily used by applications using the API to change user's account preferences and not for this user in particular.

→ The result of this method should not be cached for a long time as an user might be moved from an account to another account.

### *Number normalization*

**method: clean_number**
**parameters:** user, pass, number
**returns:** a number

→ ex : clean_number('022/364 8000') returns '41223648000'

### *Request a message status and error code*

**method:** get_status
**parameters:** user, pass, identifier
**returns:** an array of arrays

→ The 3rd parameter can be either a string or an array containing several strings, one for each queried message.

→ The returned array contains informations about the state of each message identified by its unique identifier (error code and status code).

### *Cancel the routing of a scheduled message*

**method:** cancel_message
**parameters:** user, pass, identifier
**returns:** an array of arrays

→ The third parameter may be either a string or an array containing several strings, one by message that we want to cancel.

→ The returned array includes a boolean value indicating if the cancellation of the scheduled message has been successfully performed for each identity passed in parameter.

# Status and error codes list

## *Error codes*

The status codes (see next section) give more details on the current error code.

| code | description | commentaire |
|---|---|---|
| 1 | OK | The message has been received or is being sent. |
| 2 | Gateway error | Problem with the SMS gateway. |
| 3 | Message routing error | Problem routing the SMS. |
| 4 | Message format error | A problem has occured with the format of the message. |
| 5 | Invalid destination number | The message won't be received. |
| 6 | Invalid source number | The message won't be sent. |
| 7 | Empty message | The message was empty. |
| 8 | Max message length exceeded | The maximum message size has been exceeded. |
| 9 | Pending | The message is scheduled. |
| 10 | Message canceled | The message has been canceled before it could be sent. |

## *Status codes*

| code | description | commentaire |
|---|---|---|
| 1 | Delivered to the recipient device | The message has been received by the recipient. |
| 2 | Delivered to the recipient network | Message is on the recipient's network but not delivered yet. |
| 3 | Message submitted | The message has been submitted to the recipient's network. |
| 4 | Message error, delivery failed | An error with the message itself has occured. |
| 5 | Routing error, delivery failed | An error with the message routing has occured. |
| 6 | Message expired, delivery cancelled | The recipient didn't get the message before its expiry. |
| 7 | Gateway error | Error from the gateway. |
| 8 | Message in queue | The message is being handled by our gateway. |
| 9 | Message scheduled | A planned delivery date has been set. |

# Technical support

For any question or request we kindly ask you to:

- Check our Frequently Asked Question (FAQs) on our website at : https://netoxygen.ch/support/faqs/
- write us an email at support@netoxygen.ch
- contact us by phone at +41 (0)22 364 8000 (or on our 24/7 support number outside business hours for clients with a support contract)

# Usage examples

## *Sending an SMS from PHP*

```php
<?php
require_once("XML/RPC.php");  // belongs to PEAR

$cli = new XML_RPC_Client("/api2/", "sms.netoxygen.ch");
//$cli->debug = true;   // uncomment if needed

$msg = new XML_RPC_Message("send_message",                // method
    array(new XML_RPC_Value("myuser", "string"),          // username
          new XML_RPC_Value("mypass", "string"),          // password
          new XML_RPC_Value("0792223344", "string"),      // recipient
          new XML_RPC_Value("test message", "string")     // message
    )
  );

$answer = $cli->send($msg);
?>
```

For a more complete example, see https://github.com/NetOxygen/smsapiv2clients/tree/master/PHP

## *Simple Perl script*

```
#!/usr/bin/perl
use RPC::XML::Client;
my $cli = RPC::XML::Client->new('http://sms.netoxygen.ch/api2/');
my $resp = $cli->send_request('send_message', ('myuser', 'mypass', @ARGV[0], @ARGV[1]));
```

## *More complete Perl script (handles accentuation and errors)*

```
#!/usr/bin/perl

use strict;

unless (scalar(@ARGV) == 2) {
  die("Usage: sendsms.pl destination message\n");
}

use RPC::XML qw (smart_encode);
use RPC::XML::Client;
use Encode;

my $user = smart_encode('myuser');
my $pass = smart_encode('mypass');

my $dst = new RPC::XML::base64(Encode::encode('utf8', @ARGV[0]));
my $msg = new RPC::XML::base64(Encode::encode('utf8', @ARGV[1]));

my $cli = RPC::XML::Client->new('http://sms.netoxygen.ch/api2/');
my $resp = $cli->send_request('send_message', ($user, $pass, $dst, $msg));

unless (ref $resp) {
  die("Fatal error: XML::RPC communication failed. ($resp)\n");
}

if ($resp->is_fault) {
  die("Fatal error: XML::RPC response returned with fault status. (code: ".$resp->code.",
".$resp->string.")\n");
}

my ($d, $s, $i) = @{@{$resp->value}[0]};
unless ($s) {
  die("Error: unable to send message to $d. (Error code: $i)\n");
}

print "$i\n";
```

### *.Net*

There are several XML/RPC implementation for .Net. This one can be used for example : http://xml-rpc.net/ (C# or VB.NET).