



PHP API
Release 2.1
User Manual

messagemedia

THE SMS SPECIALISTS

Table of Contents

PHP API	1
RELEASE 2.1	1
TABLE OF CONTENTS	1
TABLE OF CONTENTS	2
PRODUCT INFORMATION	3
INTRODUCTION.....	3
SUPPORTED PLATFORMS.....	3
HOW IT WORKS.....	4
INSTALLATION	4
APPLICATION DEVELOPMENT BY EXAMPLE	5
SENDING MESSAGES	5
READING MESSAGES	6
DELIVERY REPORTS	8
CHANGING ACCOUNT PASSWORDS	8
CHECKING REMAINING CREDITS	9
USING A PROXY SERVER	10
REFERENCE	11
CLASS SMSINTERFACE	11
CLASS SMSREPLY	14
CLASS VALIDITYPERIOD	15

Product Information

Product Name	Messagemedia SMS PHP API Release 2.0	
Purpose	This document contains the installation and usage instructions for the Messagemedia SMS PHP API, as well as a detailed reference.	
Prepared by	Matthew Kwan, Messagemedia Pty Ltd	
Release	2.0	4 February 2004
	2.1	13 June 2006
	2.2	29 August 2006

Introduction

The Messagemedia PHP API is designed to provide access to the Messagemedia SMS server via a PHP interface. This access includes the sending and receiving of SMS messages, as well as basic account maintenance.

The API is provided in the form of source code, which should be included in your applications via the PHP command `require("SmsInterface.inc")`

Supported Platforms

Two versions of the API are provided, supporting different releases of PHP. To benefit from the full feature set of the API, it is recommended that you use PHP 4.3 or above.

`SmsInterface-40.inc` works with PHP 4.0 and above, but uses a number of deprecated PHP commands, so it should only be used if you have a version of PHP that is older than 4.2

`SmsInterface.inc` works with PHP 4.2 and above, and is the recommended version. Note, however, that releases of PHP prior to 4.3 will not support encrypted communications with the Messagemedia server via SSL, so it is recommended that you upgrade to the 4.3 release to take full advantage of this API.

Because PHP is an interpreted language, PHP applications will run on any machine that has an interpreter installed. PHP interpreters are available for Linux, Windows NT, and all variants of Unix. For a complete list of the supported operating systems and downloadable copies of their interpreters visit <http://www.php.net>

How It Works

The API provides a wrapper for communicating with the Messagemedia SMS server via port 80 (or port 443 for encrypted SSL communications), either directly or via a proxy server.

The first step is to contact a list of active SMS servers in sequence until there is a positive response from one of them. Having found an active server, the API will then send through a username and password for authentication, then issue a command and extract information from the server's response.

Installation

If you are running PHP 4.2 or above, you should install the file `SmsInterface.inc`. If you are running an older version, install the file `SmsInterface-40.inc` instead, but rename it to `SmsInterface.inc`

Since the file is distributed as source code, it can be installed anywhere that your PHP interpreter has read access. For convenience, we suggest that you install it in the same directory as the PHP application that will be using it.

If your system does not have a PHP interpreter installed, visit <http://www.php.net> to download the latest version and follow the installation instructions provided.

Application Development By Example

The following section will explain how to use the API through the use of examples. A more detailed description of the individual API components can be found in the Reference section.

Sending Messages

```
<HTML>
<HEAD><TITLE> SMS sending results </TITLE></HEAD>
<BODY>
  <?php
    require ("SmsInterface.inc");

    echo "<P> Message sent to <B>" . $_POST["phone"] . "</B> ";

    $si = new SmsInterface (false, false);
    $si->addMessage ($_POST["phone"], $_POST["message"]);

    if (!$si->connect ($_POST["username"], $_POST["password"], true,
                      false))

      echo "failed. Could not contact server.\n";
    elseif (!$si->sendMessages ()) {
      echo "failed. Could not send message to server.\n";
      if ($si->getResponseMessage () != NULL)
        echo "<BR>Reason: " . $si->getResponseMessage () . "\n";
    } else
      echo "OK.\n";
  ?>
</BODY>
</HTML>
```

This is an example of a PHP script that would be called from a web server. The script will connect to the Messagemedia server and send out a single SMS message to a specified phone number. The source code for this script can be found in the examples directory of this distribution under the name sendsms.php. It could be called from the following HTML script (which is also included in the examples directory under the name sendsmsform.html).

```
<HTML>
<HEAD><TITLE> Send SMS </TITLE></HEAD>
<BODY>
  <FORM ACTION="sendsms.php" METHOD="POST">
    Username: <INPUT TYPE="text" NAME="username" SIZE="10"> <BR>
    Password: <INPUT TYPE="password" NAME="password" SIZE="10"> <BR>
    Phone number: <INPUT TYPE="text" NAME="phone" SIZE="13"> <BR>
    Message:
    <INPUT TYPE="text" NAME="message" SIZE="80" MAXLENGTH="160">
  <BR>
```

```
<INPUT TYPE="submit" VALUE="Send">
</FORM>
</BODY>
</HTML>
```

In the PHP code, the line `require ("SmsInterface.inc");` includes the API and is needed by every application that wishes to use it.

The API uses an object-oriented design, whose main class is `SmsInterface`. The constructor for this class takes two optional parameters - `autoSplit` which specifies whether message splitting will take place (a boolean value, which is optional and defaults to false), and `concatenate` which specifies that message concatenation will take place (also boolean, defaulting to false). These parameters are used to decide how to treat messages greater than 160 characters in length. If both are false, all such messages will be truncated to 160 characters. However, if `autoSplit` is true, long messages will be split into separate messages smaller than 160 characters and delivered at 30 second intervals, while if `concatenate` is true, long messages will be concatenated into a single long message using an extension to the GSM protocol. Note that this extension may not work on older phones.

The `$si->addMessage()` function adds a message to an internal list. You can add as many messages as you like, and they will be sent in a batch when `$si->sendMessages()` is called. However, it is recommended that you send batches of no more than 200 messages, to minimize both the time it takes to complete the task and the chance that a network or server error will cause the batch to fail. Note that if there is an error, no messages from the batch will be sent, so it is safe to re-send the entire batch.

In the example above, the `addMessage` function is taking phone number and message parameters, both strings. You can also specify extra parameters, setting a message ID, scheduled delay, validity period, and whether a delivery report is required. For details, see the Reference section of this document.

The `$si->connect()` call attempts to make a connection to the Messagemedia SMS server. The first two parameters are the username and password for your account. The third parameter determines whether message IDs will be used (even if you don't use them, as in this example, we recommend setting the parameter true anyway). The fourth parameter is used to select a secure SSL connection with the server, but please note that a true value will not work with PHP versions prior to 4.3.

If the connection is successful, we then send the messages with the `$si->sendMessages()` command. Note that this command will close the connection before returning, so you must re-connect if you wish to issue any other commands to the server. If this call fails, the server may provide a reason, which we can access via the `$si->getResponseMessage()` function.

Reading Messages

```
<HTML>
<HEAD><TITLE> Messages retrieved from server </TITLE></HEAD>
```

```

<BODY>
  <?php
    require ("SmsInterface.inc");

    $si = new SmsInterface ();
    if (!$si->connect ($_POST["username"], $_POST["password"], true,
                      false))

      echo "<P> Unable to connect to the SMS server.\n";
    elseif (($srl = $si->checkReplies ()) === NULL) {
      echo "Unable to read messages from the SMS server.\n";
      if ($si->getResponseMessage () !== NULL)
        echo "<BR>Reason: " . $si->getResponseMessage() . "\n";
    } elseif (count ($srl) == 0)
      echo "No messages waiting on the SMS server.\n";
    else {
      echo "<TABLE><TR>"
      echo "<TH> Message ID <TH> Phone number <TH> Message\n";

      foreach ($srl as $sr) {
        echo "<TR><TD> $sr->messageID <TD> $sr->phoneNumber <TD>
";

        if ($sr->status == MessageStatus::none ())
          echo "$sr->message\n";
        elseif ($sr->status == MessageStatus::pending ())
          echo "Message delivery pending\n";
        elseif ($sr->status == MessageStatus::delivered ())
          echo "Message successfully delivered\n";
        elseif ($sr->status == MessageStatus::failed ())
          echo "Message delivery failed\n";
        else
          echo "Unknown message status '$sr->status'\n";
      }

      echo "</TABLE>\n";
    }
  ?>
</BODY>
</HTML>

```

The above script will connect to the Messagemedia server, using a username and password specified in the calling form, and display all the previously-unread replies in a table. The source code for this script can be found in the examples directory under the name readsms.php.

After connecting to the server with the `$si->connect()` call, the `$si->checkReplies()` call will return an array of `SmsReply` objects and close the connection to the server. Note that if there is a problem, such as an incorrect password, this function will return `NULL`, while a successful connection with no replies will return an empty array. You should test for a `NULL` return by comparing with the `===` operator. Do not use the `==` operator, since it will not distinguish between `NULL` and an empty array.

The `checkReplies()` function takes an optional boolean parameter, defaulting to `true`, which causes the function to automatically confirm with the server that the replies have been received successfully. Once this confirmation is received, the server will mark the replies as read, and they will not be downloaded the next time this function is called. However, if the parameter is set to `false`, the replies will not be marked as read on the server until there is an explicit call to `confirmRepliesReceived()`. This can be useful if you want to make certain that the replies have been successfully saved or processed before flagging them as read. Some code to do this might look like this (plus some error checks).

```
$si->connect ("username", "password", true, false);
$srl = $si->checkReplies (false);
if (processReplies ($srl)) {
    $si->connect ("username", "password", true, false);
    $si->confirmRepliesReceived ();
}
```

The `SmsReply` class contains a `messageID` field, which corresponds to the message ID of the message that it is replying to. The `phoneNumber` and `message` fields are self-evident. The `status` field is used to indicate delivery reports, which may be received if you previously sent a message that requested one.. The `SmsReply` class also has another field not shown, `$sr->when`, which indicates how many seconds ago the reply was received. See the Reference section for further details

Delivery Reports

When sending an SMS, it is possible to request a delivery report from the network. This is done by setting the 6th parameter of the `$si->addMessage()` to `true`. However, it should be noted that messages requesting delivery reports cost more to send than regular messages. Please contact `MessageMedia Pty Ltd` for detailed pricing information.

When the SMS is successfully sent (but not yet received), a delivery report will be sent to your account by the telecommunications carrier. This report can be retrieved via the `checkReplies` function, where it will appear in the form of an `SmsReply` where the field `$sr->status` is equal to `MessageStatus::pending()`.

When the SMS is successfully delivered – or if it cannot be delivered for some reason, a second delivery report will be sent to your account. Again, this will take the form of an `SmsReply`, with a status of either `MessageStatus::delivered()` or `MessageStatus::failed()`.

Changing Account Passwords

```
<HTML>
<HEAD><TITLE> Change password result </TITLE></HEAD>
<BODY>
  <?php
    require ("SmsInterface.inc");
```

```

$si = new SmsInterface ();
if (!$si->connect ($_POST["username"], $_POST["password"], true,
                false))

    echo "<P> Unable to connect to the SMS server.\n";
elseif (!$si->changePassword ($_POST["newpassword"])) {
    echo "Unable to change the password for ";
    echo $_POST["username"] . "\n";
    if ($si->getResponseMessage () != NULL)
        echo "<BR>Reason: " . $si->getResponseMessage() . "\n";
} else {
    echo "Successfully changed the password for ";
    echo $_POST["username"] . "\n";
}
?>
</BODY>
</HTML>

```

The above script will connect to the Messagemedia server and change the password for the user's account. The source code for this script can be found in the examples directory under the name password.php.

The `$si->changePassword()` function takes the new password as its argument, which it passes to the server before closing the connection.

Checking Remaining Credits

```

<HTML>
<HEAD><TITLE> Credits remaining </TITLE></HEAD>
<BODY>
<?php
    require ("SmsInterface.inc");

    $si = new SmsInterface ();
    if (!$si->connect ($_POST["username"], $_POST["password"], true,
                    false))

        echo "<P> Unable to connect to the SMS server.\n";
    else {
        $cr = $si->getCreditsRemaining ();

        if ($cr == -2) {
            echo "Unable to read credits for " . $_POST["username"];
            echo " from the SMS server.\n";
            if ($si->getResponseMessage () != NULL)
                echo "<BR>Reason: " . $si->getResponseMessage() . "\n";
        } elseif ($cr == -1) {
            echo "The account for " . $_POST["username"];
            echo " is not a pre-paid account.\n";
        } else {
            echo "The account for " . $_POST["username"];

```

```

        echo " has $scr credits remaining.\n";
    }
}
?>
</BODY>
</HTML>

```

The above script will connect to the Messagemedia server and return the number of credits remaining for the user's account. The source code for this script can be found in the examples directory under the name credits.php.

The `$si->getCreditsRemaining()` function retrieves the number of credits remaining, then closes the connection. If there is a problem retrieving the information, the function returns -2, and may set the response message to specify the reason. The function returns -1 if the user does not have a pre-paid account, since post-paid accounts do not have credits remaining.

Using A Proxy Server

If your PHP application operates in an environment where all HTTP communications must go via a proxy server, you will need to call the `$si->setHttpProxy()` function. This should be done after creating an `SmsInterface` object, but before doing the `connect()` call. Assuming your proxy's name is `proxy.yourcompany.com`, and you connect via port 80, and the proxy doesn't require authentication, the code would look something like this.

```

$si = new SmsInterface ();
$si->setHttpProxy ("proxy.yourcompany.com", 80);
if (!$si->connect ($_POST["username"], $_POST["password"], true,
                    false))

```

If you are using the secure SSL option and HTTPS communications go via a proxy server, you will need to call the `setHttpsProxy()` function. Assuming this is done via port 443, and the proxy server authenticates with username "john" and password "xx", the code might look like this.

```

$si = new SmsInterface (2);
$si->setHttpsProxy ("proxy.yourcompany.com", 443, "john", "xx");

```

Reference

class SmsInterface

```
SmsInterface (  
    $autoSplit,          // Boolean, optional, default = false  
    $concatenate // Boolean, optional, default = false  
);
```

The SmsInterface constructor takes two parameters, which specify how long messages will be sent.

If both the autoSplit and concatenate parameters are false, all long messages will be truncated to 160 characters. However, if autoSplit is true, long messages will be split into separate messages smaller than 160 characters and delivered at 30 second intervals, while if concatenate is true, long messages will be concatenated into a single long message using an extension to the GSM protocol. Note that this extension may not work on older phones.

```
$si->connect (  
    $username,          // String  
    $password,          // String  
    $useMessageID,     // Boolean  
    $secure             // Boolean  
);
```

The \$si->connect() call attempts to make a connection to the Messagemedia SMS server. The first two parameters are the username and password for the account. The third parameter determines whether message IDs will be used (even if you don't use them, it is recommend that this parameter be set true anyway). The fourth parameter is used to select a secure SSL connection with the server, but note that a true value will not work with PHP versions prior to 4.3.

This function returns a boolean value, true on success, false on failure.

```
$si->getResponseCode ();
```

This function returns the integer response code received from calls to changePassword, getCreditsRemaining, sendMessages, and checkReplies. A value of -1 indicates that no response code has been received from the server.

Values in the 100-199 range indicate success. 400-499 indicate temporary errors (e.g. network down), while 500-599 indicate permanent errors (e.g. wrong password). Some common values are:

- 400 Server error
- 500 Syntax error (due to an error in the API communications)
- 510 Incorrect password

- 520 Insufficient daily credit
- 540 Insufficient credit

```
$si->getResponseMessage ();
```

This function returns the response message received from calls to `changePassword`, `getCreditsRemaining`, `sendMessages`, and `checkReplies`. A value of `NULL` indicates that no response code has been received from the server. The messages correspond to the codes returned by `$si->getResponseCode()`.

```
$si->addMessage (
    $phone,           // String
    $messageText,    // String
    $messageID,      // Integer, optional, default = 0
    $delay,           // Integer, optional, default = 0
    $validityPeriod, // Integer, optional, default = 169
    $deliveryReport  // Boolean, optional, default = false
);
```

The `$si->addMessage()` function adds a message to an internal list. You can add as many messages as you like, and they will be sent in a batch when `$si->sendMessages()` is called. However, it is recommended that you send batches of no more than 200 messages, to minimize both the time it takes to complete the task and the chance that a network or server error will cause the batch to fail.

The `$phone` parameter is a string containing the phone number. This can be a local-format Australian mobile number (e.g. 04xxxxxxx), an international-format Australian mobile number (e.g. 614xxxxxxx), or a full international number (e.g. +447746xxxxx). Any spaces and non-numeric characters will be stripped out before processing, so it is safe to separate the digits by spaces.

The `$messageText` contains the message to be sent. If it is over 160 characters, it will either be truncated (default), split into separate messages if the constructor sets the `$allowSplitting` parameter true, or concatenated into a long message if `$concatenate` is true.

The `$messageID` is an integer that will be passed back if there are any replies to this message. Careful selection of this value can allow messages to be correlated with replies.

The `$delay` parameter is an integer that specifies how many seconds in the future the message should be delivered.

The `$validityPeriod` parameter specifies how long the SMS network should keep trying to deliver the message if the recipient's phone is not responding. Suitable values are specified in the `ValidityPeriod` class. The default value is 3 days.

If the `$deliveryReport` parameter is set true, the SMS network will send a delivery report when the message is received by the network, and again when the message is either successfully delivered, or discarded after the validity period expires.

This function does not return any value.

```
$si->sendMessages ();
```

The `$si->sendMessages()` function sends to the Messagemedia server all the messages that have been previously added with `$si->addMessage()`. Note that this command will close the connection before returning.

This function returns a boolean value, true on success, false on failure. In the case of failure, the reason may be provided by the `getResponseCode` and `getResponseMessage` functions.

```
$si->changePassword (
    $newPassword      // String
);
```

This command changes the account's password to the specified value, and closes the connection to the server before returning.

The function returns a boolean value, true on success, false on failure. In the case of failure, the reason may be provided by the `getResponseCode` and `getResponseMessage` functions.

```
$si->getCreditsRemaining ();
```

The `$si->getCreditsRemaining()` function retrieves the number of credits remaining, then closes the connection. If there is a problem accessing the information, the function returns -2, and may set the response message to specify the reason. The function returns -1 if the user does not have a pre-paid account, since post-paid accounts do not have credits remaining.

```
$si->checkReplies (
    $autoConfirm // Boolean, optional, default = true
);
```

The `$si->checkReplies()` function returns an array of `SmsReply` objects and closes the connection to the server. If there is a problem, such as an incorrect password, this function will return NULL, whereas a successful connection with no replies will return an empty array. You should test for a NULL return by comparing with the `===` operator. Do not use the `==` operator, since it will not distinguish between an empty array and NULL.

If the `$autoConfirm` parameter is true, after downloading the array of replies the function will automatically re-connect to the server and confirm that they been successfully received. If the parameter is false, however, the replies will not be marked as read on the server until there is an explicit call to `$si->confirmRepliesReceived()` (see below).

```
$si->confirmRepliesReceived ();
```

The `$si->confirmRepliesReceived()` function notifies the server that the replies downloaded by the most recent call to `$si->checkReplies` (false) have been successfully received and can be marked as read, then closes the connection to the server.

This function returns a boolean value, true on success, false on failure.

```
$si->setHttpProxy (  
    $proxyName,          // String, may be full name or IP address  
    $proxyPort,         // Integer  
    $proxyUsername,     // String, optional, default = NULL  
    $proxyPassword      // String, optional, default = NULL  
);
```

```
$si->setHttpsProxy (  
    $proxyName,         // String, may be full name or IP address  
    $proxyPort,        // Integer  
    $proxyUsername,    // String, optional, default = NULL  
    $proxyPassword     // String, optional, default = NULL  
);
```

The `$si->setHttpProxy()` and `$si->setHttpsProxy()` functions specify the proxy servers to be used for HTTP and HTTPS communications respectively, and a username and password for authorization if required. If your application has direct access to the internet these functions should not be needed. Otherwise you should use the first function to specify the name (or IP address) of the HTTP proxy server and the port it uses, or the second function to specify the HTTPS proxy server and its port if you are using the secure SSL option.

These functions do not return any values.

```
class SmsReply
```

The `SmsReply` class contains a number of fields whose values contain reply information. Given an `SmsReply` object `$sr`, the fields are as follows.

```
$sr->phoneNumber
```

This is a string value, containing the phone number of the sender of the message. Note that the number will be in international format including the + symbol, e.g. +614xxxxxxxx.

```
$sr->message
```

This is a string value containing the reply message, or an empty string if the reply is a delivery receipt.

`$sr->messageID`

This is an integer, whose value matches the message ID of the message that is being replied to.

`$sr->when`

This is an integer value, specifying how many seconds ago the reply was received by the Messagemedia server.

`$sr->status`

This integer value indicates whether the reply is a message or some sort of delivery receipt. It can take the following values.

- `MessageStatus::none ()` // It's a message.
- `MessageStatus::pending ()` // Message awaiting delivery.
- `MessageStatus::delivered ()` // Message successfully delivered.
- `MessageStatus::failed ()` // Message delivery failed.

`class ValidityPeriod`

This class returns the following integer values, which can be used as the fifth parameter in the `SmsInterface addMessage()` function.

- `ValidityPeriod::minimum ()` // 5 minutes
- `ValidityPeriod::oneHour ()`
- `ValidityPeriod::sixHours ()`
- `ValidityPeriod::oneDay ()`
- `ValidityPeriod::twoDays ()`
- `ValidityPeriod::threeDays ()`
- `ValidityPeriod::oneWeek ()`
- `ValidityPeriod::maximum ()` // 63 weeks