Dialogue SMS REST API Toolkit

PHP Quick Start Guide

Date: March 2013

Version: 1.7.2

CONTENT

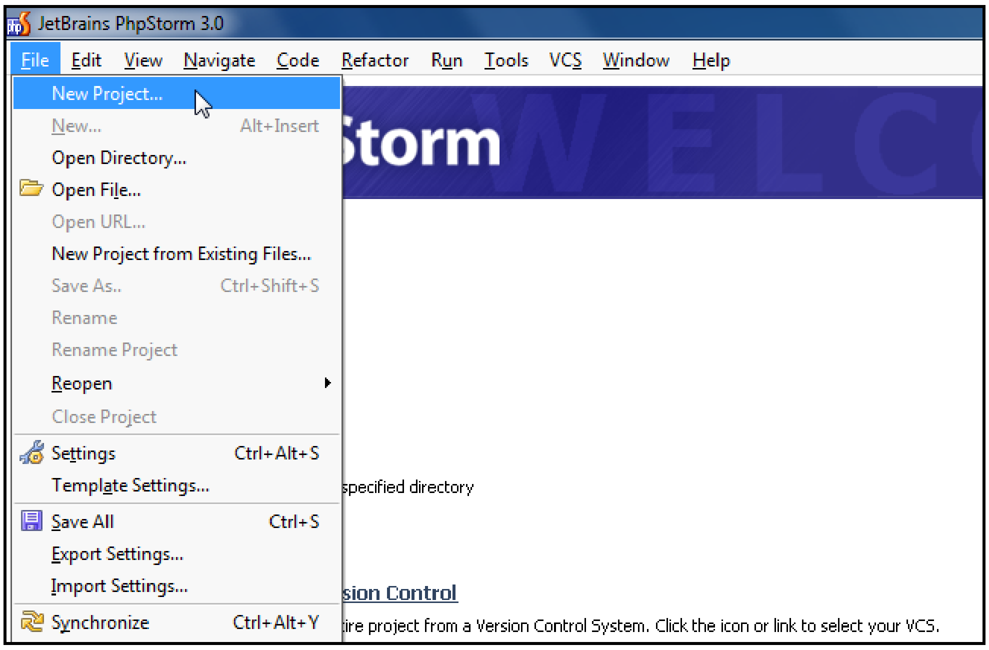
1 Sending Messages

1.1 Creating the Project

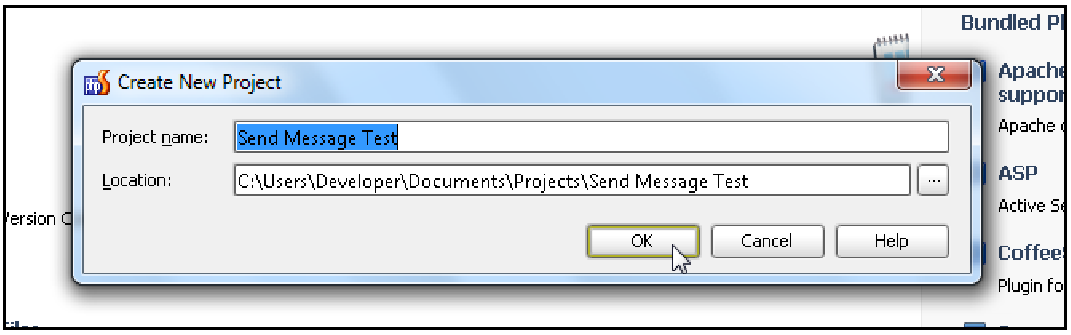
The Dialogue SMS REST API Client Library for PHP requires PHP version 5.3 or greater with the cURL extension enabled. We recommend you use the latest stable PHP editions. Most Linux distributions already ship with PHP. For other systems you can download and install PHP here: http://php.net/download/. Windows users can download and install PHP here: http://windows.php.net/download/. The PHP interpreter is free of charge.

This PHP Quick Start Guide uses a PHP IDE (integrated development environment) called JetBrains PhpStorm. This is a commercial product; a trial version can be downloaded at http://www.jetbrains.com/phpstorm/. Since PHP is based on text files executed directly by the PHP interpreter you don't need a development environment to use this guide. However, an IDE will help you with syntax highlighting and checking. If you're using another development environment you should be able perform the same tasks, although the steps required to achieve these tasks may be different.

Using PhpStorm create a new project using menu **File** > **New Project...**



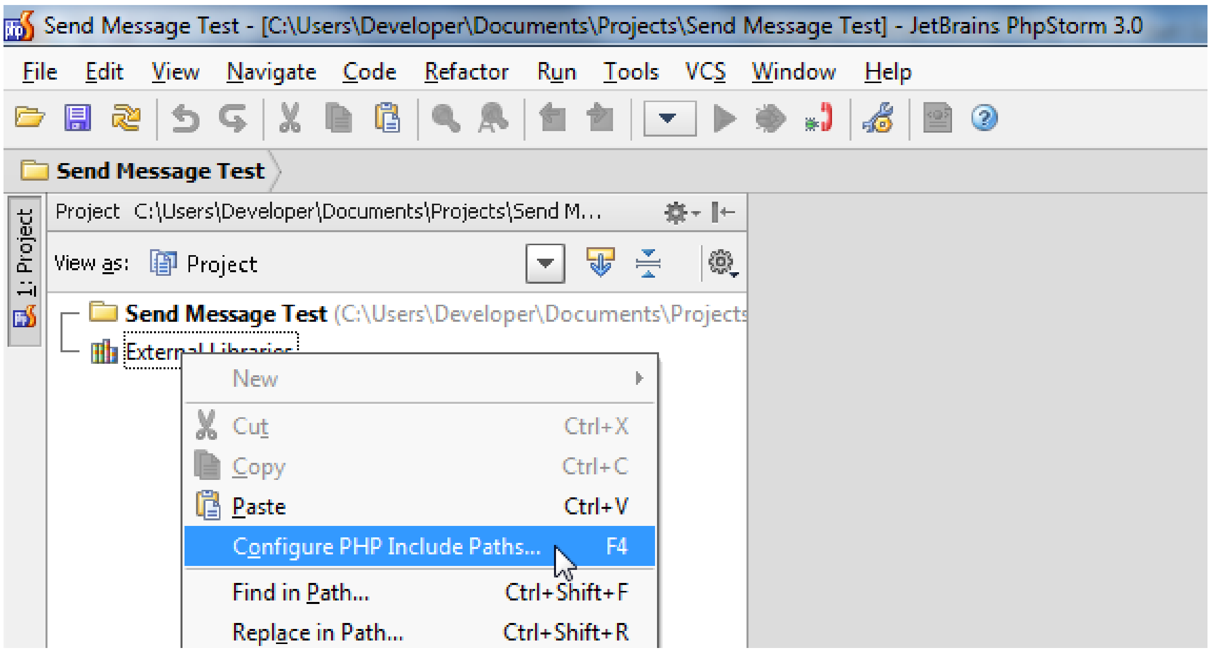
Give your project a name, select a location and press OK.



Project name: Send Message Test

Location: C:\Users\Developer\Documents\Projects\Send Message Test

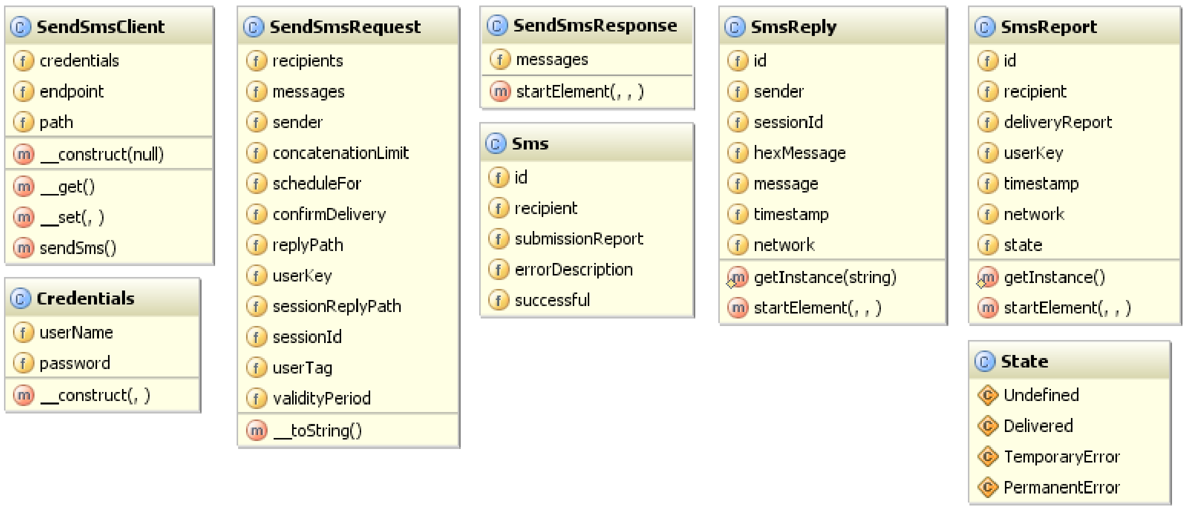
To add the toolkit library to your project right-click on **External Libraries** in the **Project Tool Window** (docked panel on left hand side) and click **Configure PHP Include Paths...** (alternatively press F4).


Send Message Test - [C:\Users\Developer\Documents\Projects\Send Message Test] - JetBrains PhpStorm 3.0
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Send Message Test
1: Project
Project C:\Users\Developer\Documents\Projects\Send M...
View as: Project
Send Message Test (C:\Users\Developer\Documents\Projects
External Libraries
New
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Configure PHP Include Paths... F4
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R

Click the [+] icon and select the folder containing DialoguePartnerToolkitClientLibrary.php. Confirm the **Select Path** dialogue by pressing **OK**. Repeat this for the **Include Paths** dialogue.

The library is now added to your project.

You can see a DialoguePartnerToolkitClientLibrary.php entry under External Libraries > PHP > PHP in the **Project Tool Window.** If you right-click the file and select **Diagram** > **Show Diagram** you can visualise the available objects, their fields and functions.



1.2 Importing the Namespace

The namespace used by the library is `Dialogue\Toolkit\Sms`. Here are the objects of interest for sending messages:

Object	Description
`Credentials`	Container object for user name and password
`SendSmsClient`	Allows sending of messages
`SendSmsRequest`	Input object (message parameters)
`SendSmsResponse`	Output object (submission results)
`Sms`	Contained once or more inside SendSmsResponse

To include the library in your PHP script write:

```
<?php
require_once '<Path>\DialoguePartnerToolkitClientLibrar
y.php';
```

where *<Path>* must be replaced by the full path to the directory containing the library, for example:

```
<?php
require_once 'C:\Users\Developer\Downloads\Dialogue Partner Toolkit
- Client Libraries\PHP\DialoguePartnerToolkitClientLibrary.php';
```

Note that on Windows the path separator is a backslash (\) character. On Mac or Linux systems the path separator is a forward slash (/).

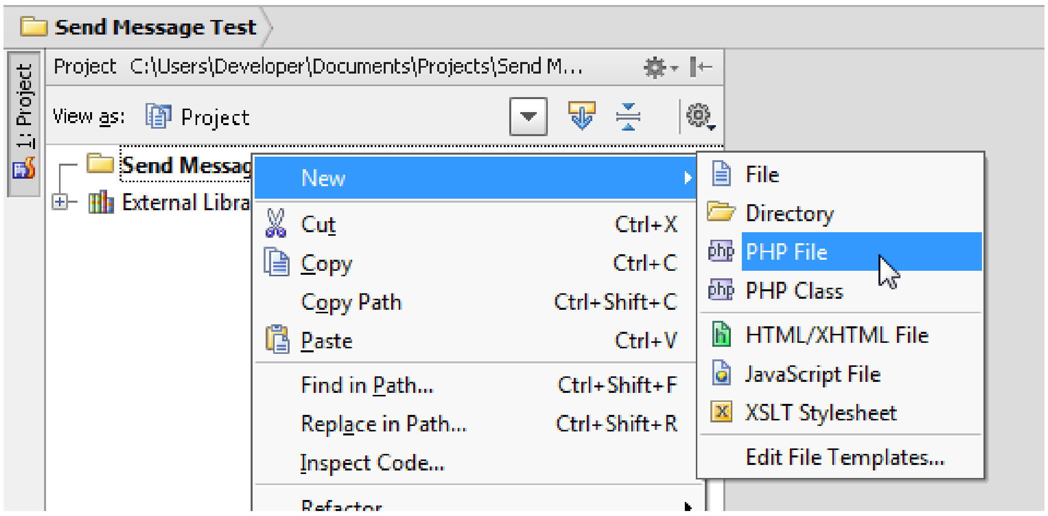
As the library uses a namespace you should to “use” each object to create a short alias by adding the code below to your script:

```
<?php
.
.
.
use Dialogue\Toolkit\Sms\SendSmsClient;
use Dialogue\Toolkit\Sms\Credentials;
use Dialogue\Toolkit\Sms\SendSmsRequest;
use Dialogue\Toolkit\Sms\SendSmsResponse;
use Dialogue\Toolkit\Sms\Sms;
```

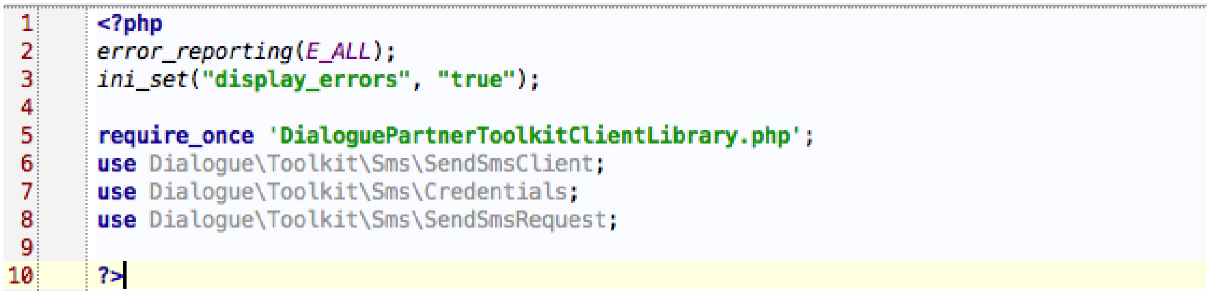
Let’s put it all together. Create a PHP script first: right-click the **Send Message Test** project root and select **New** > **PHP File**. Give the file a name, e.g. `test`, leave the extension as `php` and click **OK** to create the file. You can remove the comments.

Add the `require_once` and `use` clauses shown above. It’s also recommended to add error logging to the console by writing:

```
<?php
error_reporting(E_ALL);
ini_set("display_errors", "true");
.
.
.
```



Your full script should now look as follows:



```
<?php
error_reporting(E_ALL);
ini_set("display_errors", "true");

require_once 'DialoguePartnerToolkitClientLibrary.php';
use Dialogue\Toolkit\Sms\SendSmsClient;
use Dialogue\Toolkit\Sms\Credentials;
use Dialogue\Toolkit\Sms\SendSmsRequest;

?>
```

1.3 Instantiating SendSmsClient

To send a message you must first instantiate the `SendSmsClient` object and pass it your API endpoint and API credentials (user name and password) using the constructor:

```
$client = new SendSmsClient(
      "endpoint",
      new Credentials("user", "pass")
);
```

Replace the yellow marked endpoint, user and pass by your API user name and password, respectively. You will have received your API credentials as part of the Dialogue Partner Toolkit sign-up process. You will also have been given a document or link to a document, which contains the endpoint to use. Do not prefix the `Endpoint` property with http://.

By default secure communication via SSL is enabled; to disable SSL you can set the `secure` property to false; however we strongly recommend leaving secure communication enabled.

Bundled with the library is a recent SSL certificate authority bundle in **cacert.pem** this is required to use secure communication. In the event you need to update this file, you can download the latest version from http://curl.haxx.se/ca/cacert.pem

1.4 Sending a Message

To send a message, first instantiate the `SendSmsRequest` object, which defines various message parameters. For example:

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
```

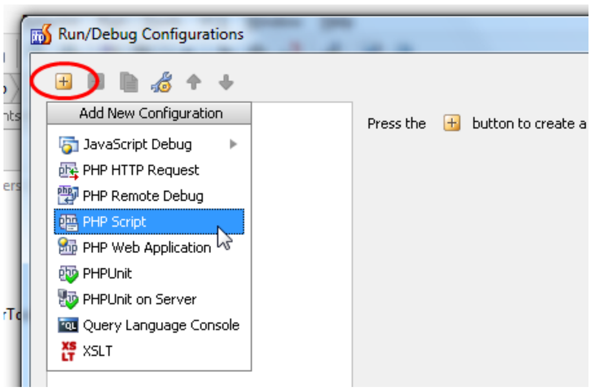
The constructor's first argument is a message (or array of messages), the second a recipient (or array of recipients). Replace the yellow marked 44xxxxxxxxxx by your recipient number in international, normalized format, e.g. starting with 1 for US numbers or 44 for UK numbers.

To submit the message write:

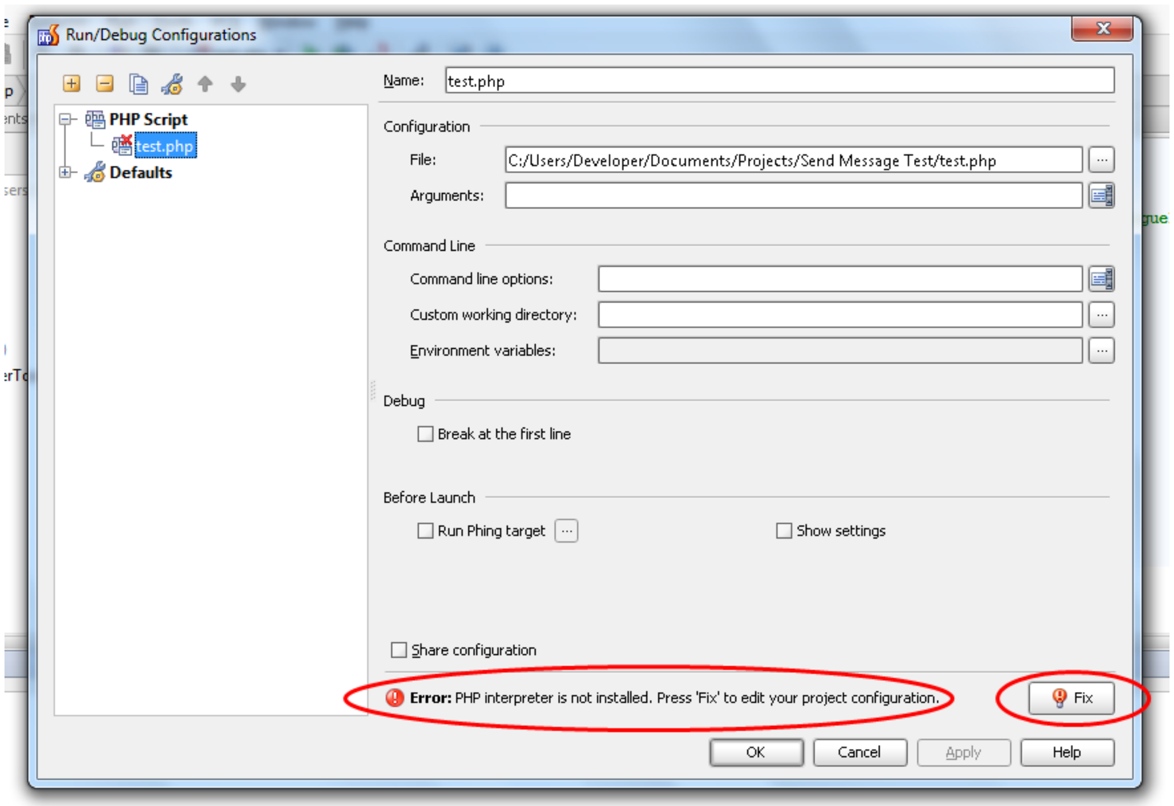
```
try {
    $response = $client->sendSms($request);
    print_r($response);
} catch (Exception $e) {
    echo "Request failed with code " . $e->getCode() .
        " and message " . $e->getMessage();
}
```

(The *`print_r`* line is not required; it merely dumps the response object for inspection, see below.)

Before you can run your script you need to set up the Run Configuration: select menu **Run** > **Edit Configurations** and click the [+] icon (Add New Configuration), then select **PHP Script**.



Change the name from *Unnamed* to *test.php* and under Configuration > File, browse to and select the test.php file you've created (you'll find it under your project root directory).


Run/Debug Configurations

Name: test.php

Configuration

File: C:/Users/Developer/Documents/Projects/Send Message Test/test.php

Arguments:

Command Line

Command line options:

Custom working directory:

Environment variables:

Debug

Break at the first line

Before Launch

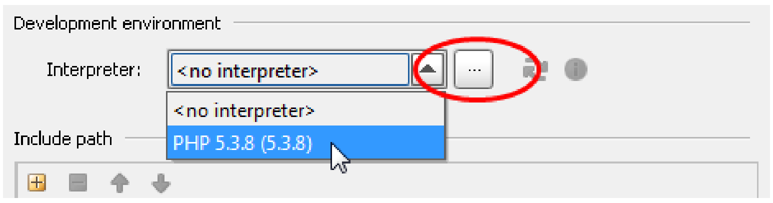
Run Phing target

Show settings

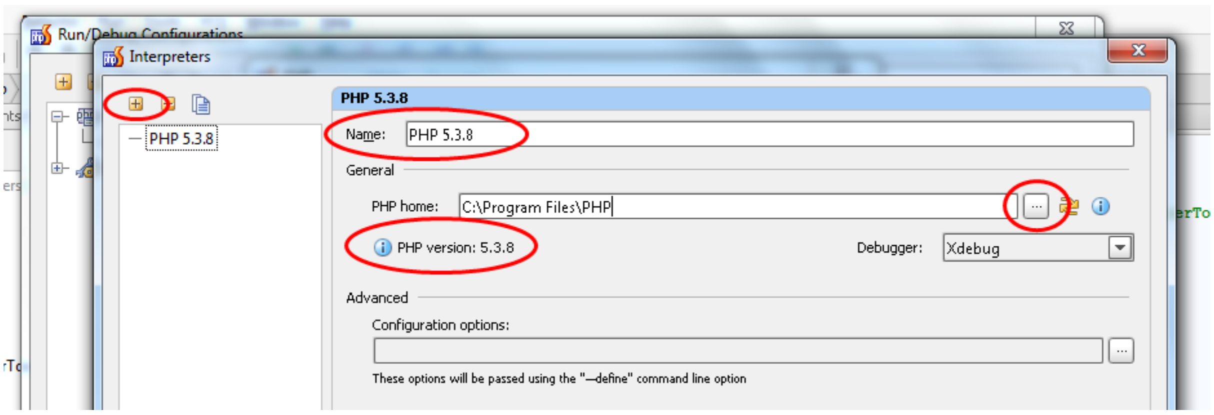
Share configuration

Error: PHP interpreter is not installed. Press 'Fix' to edit your project configuration.

If you see a warning about a PHP interpreter not being installed click the **Fix** button and select your PHP interpreter using the **Interpreter** drop down.



If no interpreter is set up click the ellipsis button followed by the [+] icon (Add). Give the interpreter setup a name under **Name**, e.g. *PHP* (or *PHP 5.3.8*, if you know the exact version you have installed) and under **PHP home** browse to the PHP installation folder; on Windows, for instance, this might be C:\Program Files\PHP. After selecting and confirming the installation folder the IDE shows the detected version.


Run/Debug Configurations
Interpreters
PHP 5.3.8
Name: PHP 5.3.8
General
PHP home: C:\Program Files\PHP
PHP version: 5.3.8
Debugger: Xdebug
Advanced
Configuration options:
These options will be passed using the "—define" command line option

Confirm all dialogs by pressing **OK**.

Run your application using menu Run > Run and that's it! You've just sent an SMS message. The **Run Tool Window** should show something similar to this (a dump of the response object):

```
net\dialogue\toolkit\sms\SendSmsResponse Object
(
    [messages] => Array
        (
            [0] => net\dialogue\toolkit\sms\Sms Object
                (
                    [id] => 5E0F7200B0574BB0A851C7C56B03431F
                    [recipient] => 447xxxxxxxxx
                    [submissionReport] => 00
                    [errorDescription] =>
                    [successful] => 1
                )

        )

)
```

1.4.1 Complete Code Sample

Here is the full code again:

```
<?php

error_reporting(E_ALL);
ini_set("display_errors", "true");

require_once 'DialoguePartnerToolkitClientLibrary.php';
```

```php
use Dialogue\Toolkit\Sms\SendSmsClient;
use Dialogue\Toolkit\Sms\Credentials;
use Dialogue\Toolkit\Sms\SendSmsRequest;
use Dialogue\Toolkit\Sms\SendSmsResponse;
use Dialogue\Toolkit\Sms\Sms;

$client = new SendSmsClient(
    "endpoint",
    new Credentials("user", "pass")
);

$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);

try {
    $response = $client->sendSms($request);
    print_r($response);
} catch (Exception $e) {
    echo "Request failed with code " . $e->getCode() .
        " and message " . $e->getMessage();
}
```

Important: If you copy and paste this code change the `require_once '...';` path to point to the correct location on your system.

1.5 Processing the Response

You should inspect the `SendSmsResponse` object returned by the `sendSms` call to see if the submission was successful or a failure. The response contains an array of `Sms` objects, each one corresponding to a submitted message. The `recipient` property can be used to associate a submission result with the original recipient (in case multiple recipients were specified).

The `successful` property of each `Sms` object is `true` if the submission succeeded (in which case the `id` property contains the unique submission identifier); otherwise use properties `submissionReport` and `errorDescription` to get an error code and description respectively.

Here is some code that inspects the `SendSmsResponse` object:

```
foreach ($response->messages as $message) {
    if ($message->successful) {
        echo "Submission to '$message->recipient' successful; messageId: 
$message->id\n";
    } else {
        echo "Submission to '$message->recipient' failed; errorCode: 
$message->submissionReport, errorDescription: 
$message->errorDescription\n";
    }
}
```

1.5.1 Complete Code Sample

Here is the full code again (the `print_r` statement has been commented out since we now process the response object):

```
<?php

error_reporting(E_ALL);
ini_set("display_errors", "true");

require_once 'DialoguePartnerToolkitClientLibrary.php';

use Dialogue\Toolkit\Sms\SendSmsClient;
use Dialogue\Toolkit\Sms\Credentials;
use Dialogue\Toolkit\Sms\SendSmsRequest;
use Dialogue\Toolkit\Sms\SendSmsResponse;
use Dialogue\Toolkit\Sms\Sms;

$client = new SendSmsClient(
    "endpoint",
    new Credentials("user", "pass")
);

$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
```

```
try {
    $response = $client->sendSms($request);
    //print_r($response);
    foreach ($response->messages as $message) {
        if ($message->successful) {
            echo "Submission to '$message->recipient' successful; messageId:
$message->id\n";
        } else {
            echo "Submission to '$message->recipient' failed; errorCode: $message-
>submissionReport, errorDescription: $message->errorDescription\n";
        }
    }
} catch (Exception $e) {
    echo "Request failed with code " . $e->getCode() .
        " and message " . $e->getMessage();
}
```

Important: If you copy and paste this code change the `require_once '...';` path to point to the correct location on your system.

1.6 Multiple Messages

You can submit multiple messages at once by setting the `messages` property to an array of strings.

```
$request = new SendSmsRequest(
    array("This is a test message", "This is another message"),
    "447xxxxxxxxx"
);
```

In the response you will find one `Sms` object for each message. The foreach loop used for processing the response in section 0 iterates over all returned `Sms` objects and prints the submission outcome for each message.

1.7 Multiple Recipients

You can target multiple recipients at once by setting the `recipients` property to an array of strings.

```
$request = new SendSmsRequest(
    "This is a test message",
    array("447xxxxxxxxx", "447xxxxxxxxx")
);
```

In the response you will find one `Sms` object for each recipient. The foreach loop used for processing the response in section 0 iterates over all returned `Sms` objects and prints the submission outcome for each recipient.

1.8 Additional Parameters

There are other parameters that can be provided via properties of the `SendSmsRequest` object. You can call `echo $request;` to dump all parameters in XML representation to the console for inspection.

1.8.1 ConcatenationLimit

By setting the `concatenationLimit` property, you enable long message concatenation. If the length of any message exceeds the default character limit the messaging gateway will send multiple concatenated messages up to the concatenation limit, after which the text is truncated. Concatenation works by splitting and wrapping the message in packets or fragments, each fragment being prefixed by the current fragment number and the total number of fragments. This allows the phone to know when it received all fragments and how to reassemble the message even if fragments arrive out of order.

The concatenation limit refers to the maximum number of message fragments, not the number of characters, e.g. a concatenation limit of “3” means no more than 3 SMS messages will be sent, which in total can contain up to 459 GSM-compatible characters. The concatenation overhead is 7 characters, so 160 - 7 = 153 available characters per fragment x 3 = 459 total characters.

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->concatenationLimit = 10;
```

In the response you will find one `Sms` object for each message segment.

1.8.2 UserTag

By setting the `userTag` property you can tag messages. You can use this for billing purposes when sending messages on behalf of several customers.

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->userTag = "Customer1";
```

The UserTag property must not exceed 50 characters; longer values will make the submission fail.

1.8.3 ScheduleFor

Use the `scheduleFor` property to delay sending messages until the specified date and time. The property is of type `DateTime` and should be constructed with a time zone defined so that time zone conversion is possible.

This example sends the message on 1st December 2011 at 6:30 PM UTC:

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->scheduleFor = new DateTime("1-Dec-11 18:30:00 UTC");
```

If the schedule date/time is in the past the message is sent immediately.

1.8.4 ValidityPeriod

Use the `validityPeriod` property to specify the maximum message delivery validity after which the message is discarded unless received. The property is of type `DateInterval` (years and months are not allowed, seconds and milliseconds are ignored, which leaves weeks, days, hours and minutes as units – note that `DateInterval` does not allow the use of weeks and days simultaneously, e.g. "P1W2D" will not work as expected). If not set the default validity period is applied.

This example sets the validity period to 1 week:

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->validityPeriod = new DateInterval("P1W");
```

This example sets the validity period to 6 hours and 30 minutes:

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->validityPeriod = new DateInterval("PT6H30M");
```

Note that when specifying a `DateInterval` in PHP you need to prefix the "time" part (hours and minutes) with the letter "T"; also valid would be `"P1DT12H"`, meaning 1 day and 12 hours.

The maximum validity period is 14 days, if you specify a longer validity period 14 days will be used. You cannot use a `DateInterval` with units Year (P*n*Y) and Month (P*n*M).

1.8.5 ConfirmDelivery, ReplyPath, UserKey

The property `confirmDelivery` can be set to `true` to enable tracking of message delivery. If you enable `confirmDelivery` you must also set the `replyPath` property pointing to an HTTP event handler that you implement (see section 2). Optionally, set `userKey` to an arbitrary, custom identifier, which will be posted back to you. You can use this to associate a message submission with a delivery report.

```
$request = new SendSmsRequest(
    "This is a test message",
    "447xxxxxxxxx"
);
$request->confirmDelivery = true;
$request->replyPath = "http://www.myserver.com/mypath";
$request->userKey = "myKey1234";
```

Note that depending on that path setup some web servers may send redirects. For example, if `mypath` is a directory the web server may respond with a 302 redirect response indicating the post should go to `mypath/`. The messaging platform does **not** follow redirects, so you need to make sure that your reply paths do not use redirection.

1.8.6 SessionId, SessionReplyPath

The property `sessionReplyPath` points to an HTTP event handler that you have implemented (see section 2). The handler is invoked if the recipient replies to the message they have received. Optionally you can specify the `sessionId` property, which will be posted back to you. You can use this to associate a message submission with a reply.

```
    $request = new SendSmsRequest(
        "This is a test message",
        "447xxxxxxxxx"
    );
    $request->sessionReplyPath = "http://www.myserver.com/mypath";
    $request->sessionId = "1234567890";
```

Note that depending on that path setup some web servers may send redirects. For example, if `mypath` is a directory the web server may respond with a 302 redirect response indicating the post should go to `mypath/`. The messaging platform does **not** follow redirects, so you need to make sure that your reply paths do not use redirection.

1.9 Custom Parameters

The Dialogue Messaging Gateway supports further parameters. To be able to use these parameters the `SendSmsRequest` object extends `ArrayObject`, which in PHP is an ordered map of keys and values. For example, to supply your own unique submission identifier you can write this code:

```
    $request = new SendSmsRequest(
        "This is a test message",
        "447xxxxxxxxx"
    );
    $request["X-E3-Submission-ID"] = uniqid();
```

(If you use the same `X-E3-Submission-ID` parameter in a future `SendSmsRequest` object it will not perform a new message submission but return the previous `SendSmsResponse` object. Do not confuse this parameter with the returned `Sms.Id` property – they are different.)

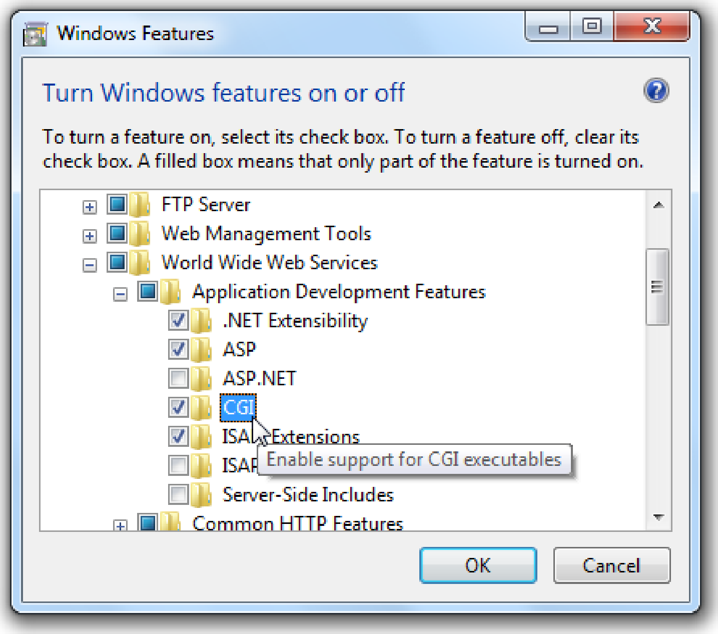
2 Receiving Reports and Replies

The toolkit library does not rely on a particular web server, i.e. you're free to use other web or application server technologies as long as they support PHP.

On most Linux distributions PHP is installed by default and available through the Apache web server.

Windows users (of Home Premium, Professional, Enterprise and Ultimate editions) can use the preinstalled Internet Information Server (IIS), provided the FastCGI module is enabled. By default IIS ships with the module disabled; to enable it follow these steps:

1. In the Windows Start Menu's **Search programs and files** field type **optionalfeatures.exe** and hit ENTER.
2. In the **Windows Features** dialog expand *Internet Information Services* > *World Wide Web Services* > *Application Development Features* and then enable the **CGI** checkbox.
3. Click **OK** and wait until the installation is complete.

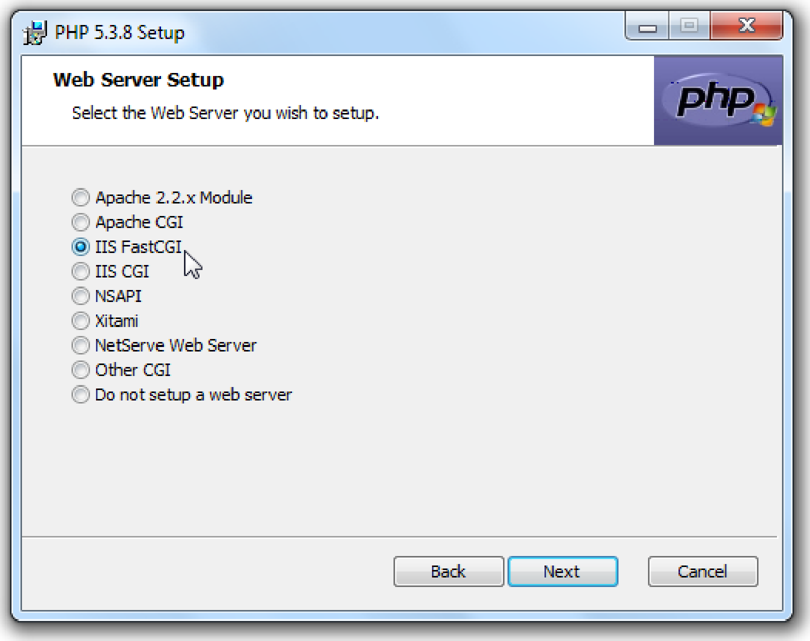


To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- FTP Server
- Web Management Tools
- World Wide Web Services
  - Application Development Features
    - [✓] .NET Extensibility
    - [✓] ASP
    - [ ] ASP.NET
    - [✓] CGI
    - [✓] ISAPI Extensions
    - [ ] ISAPI Filters
    - [ ] Server-Side Includes
  - Common HTTP Features

Enable support for CGI executables

4. Install or re-install (selecting the Change option) PHP for Windows (downloaded from http://windows.php.net/download/) and select the **IIS FastCGI** option under **Web Server Setup**. Complete the installation wizard, when done IIS will be able to execute *.php files.


PHP 5.3.8 Setup

Web Server Setup

Select the Web Server you wish to setup.

- ( ) Apache 2.2.x Module
- ( ) Apache CGI
- [✓ IIS FastCGI]
- ( ) IIS CGI
- ( ) NSAPI
- ( ) Xitami
- ( ) NetServe Web Server
- ( ) Other CGI
- ( ) Do not setup a web server

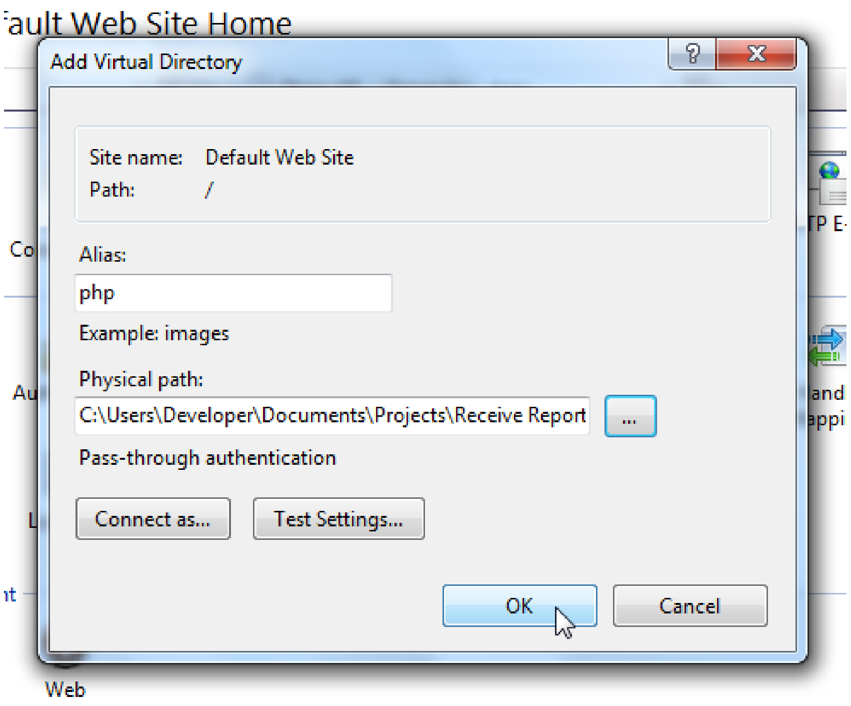
2.1 Creating the Project

Please consult section 1.1 on how to create a basic PHP project using the PhpStorm IDE. Give the project a different name, such as “Receive Report, Reply Test” and store it in a separate location.

Add the Dialogue Partner Toolkit Client Library to the project as per section 1.1.

If you are using Windows and IIS you need to create a Virtual Directory pointing to your project root directory:

1. In the Windows Start Menu’s **Search programs and files** field type **inetmgr.exe** and hit ENTER.
2. In the Internet Information Services (IIS) Manager window, in the **Connections** panel, expand the server node (typically the name of the machine) and *Sites* node.
3. Right-click *Default Web Site* (or your preferred custom web site node) and select **Add Virtual Directory...**
4. Under **Alias** enter a directory name, e.g. “php” (without the quotes).
5. Under **Physical Path** browse to your project root directory.
6. Click **OK**.
7. Make sure that IIS has access to the root directory; check that the user accounts “IUSR” or “IUSR_<*MACHINENAME*>” (where <*MACHINENAME*> is the name of your PC or server) are granted read access in the ACL (Access Control List). You can do this by right-clicking the directory in Window Explorer, selecting **Properties** and opening the **Security** tab.


Add Virtual Directory

Site name: Default Web Site
Path: /

Alias: php

Example: images

Physical path: C:\Users\Developer\Documents\Projects\Receive Report

Pass-through authentication

2.2 Importing the Namespace

The namespace used by the library is `Dialogue\Toolkit\Sms`. Here are the objects of interest for receiving reports and replies:

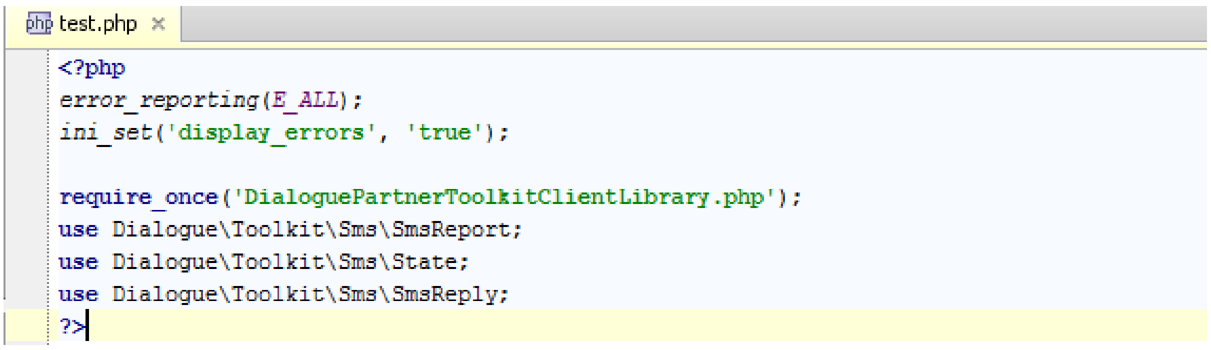
Object	Description
`SmsReport`	Utility object for parsing incoming reports
`SmsReply`	Utility object for parsing incoming replies

Start by creating a new PHP script: right-click the **Receive Report, Reply Test** project root and select **New** > **PHP File**. Give the file a name, e.g. `test`, leave the extension as `php` and click **OK** to create the file. You can remove the comments.

Add the recommended error logging and required include lines as previously done in section 1.2. Leave out the code that instantiates the `SendSmsClient`, `SendSmsRequest`, etc. Do not add the use line from section 1.2, instead write:

```
use Dialogue\Toolkit\Sms\SmsReport;
use Dialogue\Toolkit\Sms\State;
use Dialogue\Toolkit\Sms\SmsReply;
```

Your full script should now look as follows:


test.php

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', 'true');

require_once('DialoguePartnerToolkitClientLibrary.php');
use Dialogue\Toolkit\Sms\SmsReport;
use Dialogue\Toolkit\Sms\State;
use Dialogue\Toolkit\Sms\SmsReply;
?>
```

2.3 Processing Message Reports

To process message report POST requests on your page write:

```
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $report = SmsReport::getInstance();
    switch ($report->state) {
        case State::DELIVERED:
            // Message delivered
            break;
        case State::TEMPORARY_ERROR:
            // Temporary error but could still be delivered
            break;
        case State::PERMANENT_ERROR:
            // Permanent error, message was not delivered
            break;
         case State::RETRY_ERROR:
            // Retry error, message was not delivered
            break;
    }
}
```

You could add some page output using (this is ignored by the Messaging Gateway):

```
echo "Message delivered: $report->recipient, $report->userKey";
```

Open your web browser and point it to the PHP page:

http://localhost/php/test.php

(IIS usually runs on the default port, which is 80. If your IIS is set up to use a different port then use http://localhost:<*port*>/..., replacing <*port*> by the port number. If you assigned an alias other than "php" to your Virtual Directory, you will need to change the path.)

This displays a blank page. There's no parsing of an incoming report since this is a GET request.

To test the report processing download and install the `curl` command line utility from http://curl.haxx.se. Once installed you can fake a successful delivery report by running:

```
curl -H "Content-Type: application/xml" -d "<callback X-E3-Delivery-
Report=\"00\" X-E3-ID=\"90A9893BC2B645918034F4C358A062CE\" X-E3-
```

```
Loop=\"1322229741.93646\" X-E3-Network=\"Orange\" X-E3-
Recipients=\"447xxxxxxxxx\" X-E3-Timestamp=\"2011-12-01 18:02:21\" X-E3-
User-Key=\"myKey1234\"/>" -X POST http://localhost/php/test.php
```

To fake a temporary error, change the X-E3-Delivery-Report value in the XML from 00 to something between 20 and 3F. To fake a permanent error, change the X-E3-Delivery-Report value to something between 40 and 7F.

That's it! You've successfully implemented a delivery report event handler.

Of course, you should use the `recipient` and/or `userKey` properties to associate the report with a previous submission. For example, prior to submission you might store a recipient and user key record in your own database. The record would also have a DELIVERED column which is left undefined or NULL. Upon receiving a report you may look up the record using the `recipient` and `userKey` properties and update the DELIVERED column to 'Yes' or 'No' to reflect the delivery state.

All properties exposed by `SmsReport` are:

Property	Description
`id`	Unique report identifier (not that of the original submission).
`recipient`	The original recipient of the submission.
`deliveryReport`	Delivery report value, 00 to 1F indicating a successful delivery, 20 to 3F indicating a temporary error and 40 to 7F indicating a permanent error.
`state`	The delivery report value classified into `State::DELIVERED`, `State::TEMPORARY_ERROR`, `State::PERMANENT_ERROR` or `State::RETRY_ERROR`
`successful`	True if `state` is `State::DELIVERED`, false otherwise.
`userKey`	Optional `userKey` parameter from the message submission. Can be used to associate unique submissions with reports, even if the recipient is the same.
`timestamp`	Date and time the report was received (DateTime type).
`network`	Mobile operator network name.

Here is the full PHP page source code:

```
<?php

error_reporting(E_ALL);
ini_set("display_errors", "true");

require_once 'DialoguePartnerToolkitClientLibrary.php';
```

```
use Dialogue\Toolkit\Sms\SmsReport;
use Dialogue\Toolkit\Sms\State;
use Dialogue\Toolkit\Sms\SmsReply;

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $report = SmsReport::getInstance();
    switch ($report->state) {
        case State::DELIVERED:
            // Message delivered
            echo "Message delivered: $report->recipient, $report->userKey";
            break;
        case State::TEMPORARY_ERROR:
            // Temporary error but could still be delivered
            echo "Temporary error: $report->recipient, $report->userKey";
            break;
        case State::PERMANENT_ERROR:
            // Permanent error, message was not delivered
            echo "Permanent error: $report->recipient, $report->userKey";
            break;
         case State::RETRY_ERROR:
            // Retry error, message was not delivered
            echo "Retry error: $report->recipient, $report->userKey";
            break;
    }
}
?>
```

Important: If you copy and paste this code change the `require_once '...';` path to point to the correct location on your system.

Do not run this code using the PHP interpreter, for example, via the IDE's Run button; it must be executed in the context of an HTTP request by pointing your browser at your web server.

2.4 Processing Message Replies

To process message reply POST requests on your page write:

```
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $reply = SmsReply::getInstance();
    $sender = $reply->sender;
    $message = $reply->message;
    $sessionId = $reply->sessionId;
}
```

You could add some output using:

```
echo "$sender, $message, $sessionId";
```

By default the `$message` string is UTF-8 encoded. If you plan to use this string with other character encodings you can pass the desired character set name as the second argument to the `getInstance` function. For example:

```
$reply = SmsReply::getInstance(null, "CP1252");
```

The above encodes the message string using the Windows-1252 or CP-1252 character set.

Open your web browser and point it to the PHP page:

http://localhost/php/test.php

(IIS usually runs on the default port, which is 80. If your IIS is set up to use a different port then use http://localhost:<*port*>/..., replacing <*port*> by the port number. If you assigned an alias other than "php" to your Virtual Directory, you will need to change the path.)

This displays a blank page. There's no parsing of an incoming report since this is a GET request.

To test the reply processing download and install the `curl` command line utility from http://curl.haxx.se.

Once installed, you can fake a successful delivery report by running:

```
curl -H "Content-Type: application/xml" -d "<callback X-E3-Account-
Name=\"test\" X-E3-Data-Coding-Scheme=\"00\" X-E3-Hex-
Message=\"54657374204D657373616765\" X-E3-
ID=\"809EF683F022441DB9C4895AED6382CF\" X-E3-Loop=\"1322223264.20603\" X-
E3-MO-Campaign=\"\" X-E3-MO-Keyword=\"\" X-E3-Network=\"Orange\" X-E3-
Originating-Address=\"447xxxxxxxxx\" X-E3-Protocol-Identifier=\"00\" X-E3-
Recipients=\"1234567890\" X-E3-Session-ID=\"1234567890\" X-E3-
Timestamp=\"2011-11-25 12:14:23.000000\" X-E3-User-Data-Header-
Indicator=\"0\"/>" -X POST http://localhost/php/test.php
```

That's everything required to implement a handler which can receive incoming replies. Here are all the properties exposed by `SmsReply`:

Property	Description
`id`	Unique message identifier (not that of the original submission).
`sender`	The original recipient of the submission (now the sender as it's a reply).
`sessionId`	Optional `SessionId` parameters from submission. Can be used to associate unique submissions with replies, even if the recipient is the same.
`message`	The message text.
`timestamp`	Date and time the message was received (DateTime type).
`network`	Mobile operator network name.

Here is the full PHP page source code:

```php
<?php

error_reporting(E_ALL);
ini_set("display_errors", "true");

require_once 'DialoguePartnerToolkitClientLibrary.php';

use Dialogue\Toolkit\Sms\SmsReport;
use Dialogue\Toolkit\Sms\State;
use Dialogue\Toolkit\Sms\SmsReply;

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $reply = SmsReply::getInstance();
    $sender = $reply->sender;
    $message = $reply->message;
    $sessionId = $reply->sessionId;
    echo "$sender, $message, $sessionId";
}
?>
```

Important: If you copy and paste this code change the `require_once '...';` path to point to the correct location on your system.

Do not run this code using the PHP interpreter, for example, via the IDE’s Run button; it must be executed in the context of an HTTP request by pointing your browser at your web server.