**PHP Library**

Simple client library for PHP providing classes to send and receive JSON-RPC requests.

**Requirements**

* PHP 5 >= 5.2.0
* PECL json >= 1.2.0
* PHP cURL extension

**Installation**

require\_once(‘path/to/lib/SkrillPSP.php’);

**How to use library?**

Each request type is represented in a separate object. Request object loads Skrill’s predefined json structure and sets id member on creation.

1. Prepare request object for methods supported by Skrill.

**$debit = new SkrillPsp\_Debit();**

For Credit/Debit request you can provide payment token as constructor parameter.

**$debit = new SkrillPsp\_Debit("58409c4a39a94cf4bd3fe07647fca9fd");**

If you don't provide payment **token** you must provide parameters from account group (cardholder, number, expiry date, cvv).

For Refund/Capture/Reversal/Cancel request you can provide **referenceId** as constructor parameter.

**$capture = new SkrillPsp\_Capture("c21a0ef567a14d7bad56582099f9f7c8");**

For some requests referenceId is required so if you don’t pass it like argument an exception will be thrown.

1. After object creation you have to set Skrill API endpoint. You can use one of the following methods:

**$debit->setUri("https://psp.dev.skrillws.net/v1/json/3e40a821/channelid\_3d/creditcard");**

or

**$debit->setMerchantUrl("https://psp.dev.skrillws.net/v1/json", "3e40a821", "channelid\_3d", "creditcard");**

The last one allows you to dynamically provide and change Merchant ID, and Channel ID and Payment method instead of using hard coded url.

1. Next step is to set parameters that you want to send to the Skrill Payment Platform. Each type of request needs different parameters. Some of them are required while others are optional. You can set parameters through the following methods:

$param = array(‘identification’ => array(‘transactionid’ => ‘CR\_DF45’, ‘customerid’ => ‘11134’),

‘payment’ => array(‘amount’ => ‘128’, ‘currency’ => ‘GBP’, ‘descripor’ => ‘test descr’),

‘account’ => array(‘cardholder’ => ‘Test Test’, ‘number’ => ‘352637478483’, ‘expiry’ => ‘10/2015’, ‘cvv’ => ‘122’));

**$debit->setParameters($params);**

or use setter methods like:

**$debit->setAmount(“456”)-> setCurrencty(“EUR”);** etc.

Setter methods support method chaining in order to save you typing. With setters you set parameters one by one.

With **setParameters** method user can set all parameters at once. You save typing for the sake of readability. If you don’t provide required parameters for respective request type or invalid data an exception is thrown. With this method you can’t modify or overwrite predefined json-rpc structure for the request which saves you from sending incorrect data to Payment Platform. For Credit/Debit request if you provide a payment token within constructor and then set account group parameters (cardholder, number, exp.date, cvv) with **setParameters** method or setters methods they will not be sent to Payment Platform.

If required parameters are changed or json request structure is modified you can handle such a situation with **addParameters** method. It will not allow you overwrite the already predefined structure of json request only to add additional groups and parameters.

**$debit->addParameters(array('frontend' => array('responseurl' => 'http://resp.php', 'successurl' => 'http://success.php', 'errorurl' => 'http://error.php')));**

1. After that you can view json request parameters and its values that will be sent to Skrill Payment Platform with the help of **showJson** method.

**echo $debit->showJson();**

1. Final step is to send a request with **makeCall** method

**$response = $debit->makeCall();**

1. Working with response

If processing of the request was successful you have to manage the response. $response will be an object which contain *id, jsonrpc* and *result* fields in case of success, otherwise the result object will contain *error* instead of *result*. The error object has three members, *message, code* and *data* which are optional. You can understand if the response is success or error with two methods:

**$response->isSuccess() or $response isError();**

In case of success you can get the *id* field with **getId()** method

**$id = $response->getId();**

Jsonrpc version with **getVersion()** method.

**$version = $response->getVersion();**

You can get the values *of level, code, method, type, message* fields in two ways:

* they are available either like public properties of $response object

**$response->type;**

**$resposne->message;**

**$response->method;**

**$response->code;**

**$resposne->level;**

* or through **getLevel, getCode, getMethod, getType, getMessage** methods

**$response->getLevel();**

**$response->getCode();**

**$response->getMethod();**

**$response->getType();**

**$response->getMessage();**

Identification, payment and account groups can be accessed with the help of **getIndentification, getPayment and getTokenFromAccount** methods and iterate through their returned values.

**$identity = $result->getIdentification();**

**$payment = $result->getPayment();**

**$token = $result->getTokenFromAccount();**

In case of error you can access *id* and *version* in the same way as with success response. *Code and message* fields are accessible in two ways

* through methods **getErrorCode, getErrorMessage**

**echo $result->getErrorMessage();**

**echo $result->getErrorCode();**

* through the **getError** method which returns CODE and MESSAGE members of JSON-RPC Error object

**$errorData = $result->getError();**

**echo $errorData->code;**

**echo $errorData->message;**

*Level, errormessage and advice* can be accessed in two ways too:

* Through the **getErrorData** method which returns DATA member of JSON-RPC Error object

**$data = $result->getErrorData(); and iterate through $data**

* Shorthand way – directly through the **getErrorLevel, getAdvice, getErrorDataMessage** methods

**echo $result->getErrorLevel();**

**echo $result->getErrorDataMessage();**

**echo $result->getAdvice();**

**getErrorDataMessage** provides a more detailed description of error in contrast to **getErrorMessage** that provides a short description of the error.

Some code samples for different request types can be viewed in examples folder.

Our API can raise exceptions for many reasons, invalid parameters, authentication errors, and network unavailability. We recommend always trying to gracefully handle exceptions from our API.