SimplePay 1.0.x

Development documentation

Payment process, development by using SDK

2018. 08. 29.

TABLE OF CONTENT

1 Brief summary

1.1 The SimplePay online payment

The SimplePay is an online credit card payment solution provided by OTP Mobil Kft.

1.2 Concepts

Concepts frequently used in the documentation
One-step payment: after initiating the payment, the card is debited.
Two-step payment: after the payment, the given amount is reserved on the customer's account but it is not yet debited. Afterwards, the merchant has 21 days to initiate the card debit or to unlock the amount.
Sandbox: a test system which has the same technical functions as the live system but in case of transactions made through it no real payments are made. In this system only transactions necessary for development are made.
Live system: a system which can be used for real transactions. It is only available after the test system has been developed and successfully tested. Technically, it is similar to the sandbox system.
SDK: a sample code which can be used for the development, the implementation in the own system and payment.
Transaction: a payment initiated from the merchant's website, the result of which the SimplePay system reports to the merchant. The content of the transaction (the payable product) can be anything within the framework of the contract. The payment transaction starts with the initiation from the merchant and ends with a confirmation of the payment. The payment transaction is only a part of the merchant-side purchase.
Payment page: an element of SimplePay system. The customer provides his/her card details, or in case of a wire transfer, he/she is notified about the necessary information here
LiveUpdate: starting a payment transaction from the merchant's page to SimplePay's payment page.
BackRef: an information page in the merchant's system. The customer is redirected to here after the payment from the SimplePay's payment page.
IPN: background communication between SimplePay and the merchant. It notifies about successful payments.
IOS: query of a transaction's status.
IDN: in case of a two-step payment, the initiation of the second step (debit) by the merchant
IRN: initiation of a refund by the merchant
Merchant account: a unique merchant administrational platform which is available both in sandbox and the live systems.
Currency: the currency of transaction which may be forint (HUF), euro (EUR) and dollar (USD). The currency depends on the merchant's account.
Payment method: it may be credit card or wire transfer in the SimplePay system
Logo: SimplePay logo on the merchant's page
Data transmission declaration: accepting the delivery of customer data during the registration or before the transaction is made.
Merchant tests: tests made during the development by the merchant whether the operation complies with the documentation.
Testing: payment test before activation. The test is made by the SimplePay. Only those websites or payment systems are activated where the tests were successful.
Activation: after the successful tests the live account will also be activated for the merchant. It will be able to receive online payments through this account.

1.3 The development and activation process of online payment

The following tasks shall be performed by the developer during the development.

- Making a transaction with customer and basket details
- Notifying customers redirected from the payment page about the transaction's result
- Receiving an IPN message
- Placing SimplePay logo on the website

- Placing SimplePay data transmission declaration on the website

After performing the above tasks, the completion of the payment's development can be indicated to itsupport@otpmobil.com. Afterwards, our colleagues will also check the online payment.
If the tests were successful, the payment activation begins.

1.4 Timing of the development

During the implementation of SimplePay payment, the following shall be taken into account:

- Time needed for you or your developer
- Time needed for SimplePay to test the website/payment system
- Technical lead time of the activation

If the development is fully performed and each compulsory point is checked by the developer based on **chapter 7.,** the SimplePay testing of the website may be requested at itsupport@otpmobil.com.
The testing is performed within **1-3** business days from the announcement. Our colleagues will provide information about the tests' results in all cases.
If the SimplePay technical tests are successful, the payment can be activated on the site. In this case, the account is activated **within 1-5 business days after the successful tests**.

NOTE: A criterion for the activation is a signed contract.

Conclusively, you may only be sure that the planned starting date of credit card payments can be respected if you tell us about the completion of the development **5-8 business days** earlier.

1.5 Steps of activation regarding the merchant

The development is performed by using the test system (sandbox). After the successful tests the system will be active. The merchant ID data necessary for making a transaction are the same in the live and the sandbox system. Consequently, after activating the system you may regulate to which system you want to send the transaction in the future by changing one parameter.
There is no relation between the test and live systems, thus the own configurations regarding the payment shall be performed in the live system too.
No further action required if you integrate the sample code attached to the development into your own system.

1.6 Process of payment transactions

1.6.1 Credit card payment

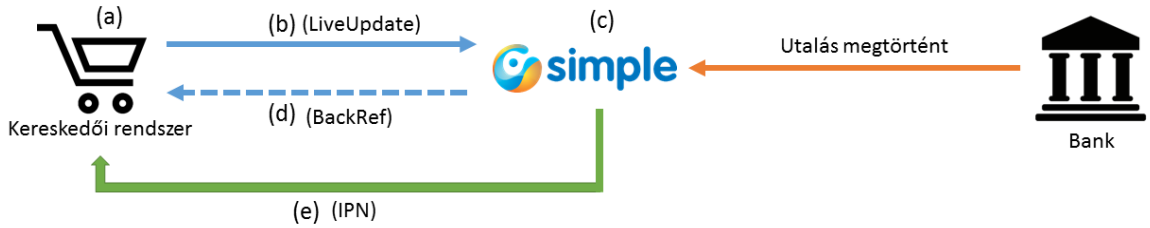
a. The customer selects the products to be purchased on the merchant's website, then initiates the SimplePay online payment.
b. When he/she starts the payment, the order's details are transferred to SimplePay system (**LiveUpdate**). Simultaneously, the customer is redirected to SimplePay's payment page. The customer may provide his/her credit card details on the payment page. When he/she arrives to the payment page, the transaction is made in SimplePay system.
c. After giving the credit card details, the payment is authenticated by the bank.
d. The customer is redirected to the merchant's website (**BackRef URL**). Here, the customer must be notified about the transaction's result based on the provided data.
e. Afterwards, the fraud monitoring and preventive process is executed in the background. If during this the system does not detect any problem, the SimplePay system sends confirmation to the website (**IPN**). This is the end of a payment transaction. After the IPN message arrives, the merchant is allowed to fulfil the order.

During the development those included in points **a, b, d, f, e** and **h** shall be implemented in the merchant's website.

1.6.2 Transfer Payment

a. The customer selects the products to be purchased on the merchant's website.
b. When he/she starts the payment, the order's details are transferred to SimplePay system (**LiveUpdate**). Simultaneously, the customer is redirected to SimplePay' information page. Here you can find data needed for the transfer such as bank account number and the content of notice.
c. Upon arriving to the information page the transaction is processed.
d. Optionally, the customer may return to the merchant's website (**BackRef**) where he/she must be notified about the transaction's status.
e. Once the transferred amount has arrived, the system sends confirmation to the website (**IPN**). This is the end of a payment transaction. After the IPN message arrives, the merchant is allowed to fulfil the order.

During the development those included in points **a, b, d, f, e** and **h** shall be implemented in the merchant's website.



1.7 SANDBOX system

The SimplePay is composed of two completely individual payment systems. One of the systems can be used for developing tests while the other for live transactions. **The two systems have no connection with each other at all.**
Due to the above it is vital to take into consideration that **each configuration** which you perform on one or the other system **is only effective in that system.** If you want to apply it also in the other system, then you will need to configure it as well.
Your account in the live system will be activated once the development and testing by SimplePay on the sandbox have been successfully completed. If you attempt to make a transaction earlier you will receive an error message.
These are the main differences between the two systems:

Sandbox
It is **solely** suitable for **test transactions**.
Transactions can be made only by the two test cards found on the payment page.
The ID numbers of transactions begin with "9" and have eight digits (**9xxxxxxx**).
Merchant admin platform: https://sandbox.simplepay.hu/admin

Live
It is **solely** suitable for **live transactions**.
The ID numbers of transactions begin with "2" and have eight digits (**2xxxxxxx**).
Merchant admin platform: https://secure.simplepay.hu/admin

In the following we present each example by using sandbox system.

2 Operational test system in 15 minutes

The PHP sample code (SDK) necessary to implement the payment system is available together with this documentation for those who use or develop SimplePay.
The sample code may be used under GNU GPL3 license conditions in order to make transactions in SimplePay.
The use of SimplePay system is regulated by the SimplePay merchant contract.
The sample code is appropriate for making payment transactions, thus t**he most important thing is to know the operation of the respective, host system** in order to see through its logic and be able to **include the on-line payment in** the suitable place.
By means of SDK it is possible to create an operational test system within 15 minutes. Once you extract this code onto your server and configure it as indicated below, the credit card payment can promptly be tried in SimplePay's test system (sandbox).
If you perform this, you will exactly see in your operational sample code what you should accomplish in your own system.
During the development it greatly facilitates troubleshooting if you use the SDK without changes as an own separate test system. In this case, if you encounter a problem in the embedded code you will immediately be able to compare and check it whether the given function operates well.

2.1 The structure of SDK

The extracted SDK has a structure indicated below.
/sdk
The sample code for credit card payment is in this folder in **SimplePayment.class.php** file. The file includes all the necessary codes for all the realizable functions, thus also for LiveUpdate, BackRef, IPN, IDN, IRN, IOS communications.
In the very same folder you will find **config.php** file. In this file you have to configure the merchant's account details without which no transaction can be made.
/demo
In this folder you will find the elements of demo platform. This folder is only necessary for the use and testing of sample code, but not for the implementation, that is, it can be deleted.
/log
Default log folder of SDK If you want to forward logging to anywhere else, the own logging route can be supplied in **config.php** file.
/logo
Place of logos used by demo platform. If you do not use demo platform, you can delete it here, or you can copy it anywhere else.
nogui.zip
If you want to work from simplified files without demo platform you have to do the following:

1. Unzip **nogui.zip** file into the main folder which overwrites the php files in the main folder.
2. Delete the "demo" folder from the sdk.

php files in the main folder
If you want to deduct the code to be embedded from the existing demo platform, you have to pay attention to much more things. Due to its easy testability, the sample code is provided with a SimplePay web-based user interface.
As much as possible this interface is separated from the code executing online payment. As the sample code was made for using demo interface, the code therein includes also such elements that may help to understand but not necessary to develop them in the own website.
Each file is the sample of a possible function.
The php files in the main folder use constants defined in demo_functions.php file in the demo folder in many places. If you do not use the demo system you have to delete these constants, or you have to overwrite them with the values of your own variables, otherwise you will receive an error message because of the undefined constants. These constants are deleted from php files in **nogui.zip** file.

If you only want to work with the source code executing payment you should use the content of **nogui.zip.** The nogui.zip package contains the sample codes without demo platform. The php files in the main folder can be overwritten with the content of the package.

2.2 Settings of SDK config

Unzip the sample code onto your server. Open sdk/config.php. The file includes an array which elements can be configured for the first tests by configuring the merchant's details and the URL in the following way. You should not deal with the other parameters at this time during the first test.

2.2.1 Merchant's account details

The sample code is suitable for the concurrent use of all the three possible currencies. If you sell goods in more currencies on your website and you have HUF, EUR and USD accounts, then the sample code can automatically alternate between the accounts with different currencies based on the currency of the current transaction.

Data of accounts can be supplied in the following fields

```
'HUF_MERCHANT' => '',        //merchant account ID (HUF)
'HUF_SECRET_KEY' => '',      //secret key for account ID (HUF)
'EUR_MERCHANT' => '',        //merchant's account ID (EUR)
'EUR_SECRET_KEY' => '',      //secret key for account ID (EUR)
'USD_MERCHANT' => '',        //merchant's account ID (USD)
'USD_SECRET_KEY' => '',      //secret key for account ID (USD)
```

You will find data for these fields on SimplePay's merchant admin interface under the following URL: https://sandbox.simplepay.hu/admin
The header section of "**Account/Technical**" page contains the general data of the account. You can find here the Merchant ID (**MERCHANT**).
The "**Base data**" tab shows the key for hash computing (**SECRET_KEY**).
These properties of the accounts are unique per currencies.

You can alternate between the unique currency accounts in the menu on the right (**Select another account**).

2.2.2 Protocol

Protocol used on your website. Its value may be either http or https.

```
'PROTOCOL' => 'http',
```

2.2.3 URLs

```
'BACK_REF' => $_SERVER['HTTP_HOST'].'/backref.php',      //url of payment backref pa
ge
'TIMEOUT_URL' => $_SERVER['HTTP_HOST'].'/timeout.php',   //url of payment timeout pa
ge
```

You have to provide the source of **backref.php** in SDK to the BACK_REF variable, the customer is redirected to here after payment from the SimplePay's payment page.
Also here, you need to provide the source of **timeout.php** to the TIMEOUT_URL variable. The customer is redirected to here in case of a timed-out or cancelled payment.
If you set the above and you call the **index.php** file in the sample code in browser, you can immediately make test transactions, during which you will reach SimplePay's payment page, then after selecting credit card details and starting the payment you will be redirected to the sample code's backref.php page.

NOTE: test credit card details for testing can be selected on the payment page.

2.2.4 Alternating between test and live transactions

By default, the sandbox is active in SDK.

```
'SANDBOX' => true,
```

If you change to live system after successful tests, then you have to change this variable to 'false' value.
Your sandbox account remains also after the activation; thus you can do tests in future if you would make any further developments.

2.2.5 Means of communication

By default, cURL is set for communications in the background.

```
'CURL' => true,
```

If this function is not available on your server, you have to set this to 'false'. In this case, the SDK can read the SimplePay-side output through file_get_contents().

2.2.6 Logging

Logging is an important help in troubleshooting during development. By default, it is activated in the SDK, only the sdk/log folder must be writeable.

```
'LOGGER' => true,
'LOG_PATH' => 'log',
```

2.2.7 Debug mode

Logging records the transaction's basic details but not the statuses of program's run. By activating debug mode, you can get detailed running data.
Debug options are disabled by default. If during the development you activate it for troubleshooting purposes, it will write all possible data and the statuses of SDK's running in the log during a SimplePay payment.
In order to activate debug mode in case of the desired process, you have to set the below config data to 'true'

```
'DEBUG_LIVEUPDATE_PAGE' => false,
'DEBUG_LIVEUPDATE' => false,
'DEBUG_BACKREF' => false,
'DEBUG_IPN' => false,
'DEBUG_IRN' => false,
'DEBUG_IDN' => false,
'DEBUG_IOS' => false,
'DEBUG_ONECLICK' => false,
```

If the DEBUG_LIVEUPDATE_PAGE is activated, it stops the transaction in the demo interface and displays the debug data of LiveUpdate. In case all the other elements are activated, debug data are included in the log file.

NOTE: debug notes may be great help in troubleshooting but it may increase the size of logs in an unnecessary extent in case of ordinary transactions. Please apply it in view of the above!

2.3 The IPN setting of sample code

The Instant Payment Notification (IPN) is the end of a payment transaction. At this point the SimplePay system notifies the merchant's online store about the transaction's success. IPN is a background process between SimplePay and the merchant. The IPN is sent from the SimplePay system to the URL provided by the merchant with POST method.

IPN messages are sent **only in case of successful and feasible transactions**. This is a clear indication of the transaction's success. In case of success, the payment can be set as paid in the merchant's system - in accordance with the store - and the order can be fulfilled.
The IPN's URL setting can be performed in the merchant's control panel (**https://sandbox.simplepay.hu/admin/**). Data can be configured in "**Technical data**" menu.
In case of the sample code, the path of ipn.php is "http://domainname.hu/ipn.php"
IPN must be configured per account. If you use more accounts in one domain (e.g. both HUF and EUR currencies are available), URL must be provided in both cases (accounts). In case of an SDK, the two URLs can be the same because the sample code will perform the data processing in the appropriate merchant account by deducting the incoming IPN call's currency.
In case of IPN URL, you must pay attention to the following

- it must be publicly available
- it must not be .htpassword protected
- redirection rules, SSL settings cannot prevent access

The test system achieves its full potential by using IPN. You will find further important information on the usage of IPN in the chapter of IPN testing.

3 Transaction and its statuses

Payment transactions must be differentiated from the entire merchant-side **purchasing process** as it only represents a part of that. Purchasing process includes the selection of products, basket details, then, optionally, the request for customer details necessary for the performance.
Once all information needed for the purchase are available, SimplePay payment may be started. Depending on the outcome of this can the order be fulfilled and the purchasing process be finished.
The status of payment is basically important in a relation of successful/unsuccessful from the aspect of the merchant's purchasing process but it has more complex and more statuses.
The following statuses can any time be queried with an IOS call. The details of IOS call will be discussed in detail subsequently with the other steps of the process.
If the request is sent regarding a non-existing transaction, the IOS call provides a NOT_FOUND status.

The SimplePay's payment process has the following statuses.

Event	Status	Process
Initiated transaction on payment page	CARD_NOTAUTHORIZED	LiveUpdate
Timeout on payment page	TIMEOUT	LiveUpdate, Timeout
Cancelled payment on payment page	CANCELLED	LiveUpdate, Timeout
Waiting for payment in case of credit card payment or wire transfer	WAITING_PAYMENT	LiveUpdate, BackRef
Unsuccessful authorization	CARD_NOTAUTHORIZED	BackRef
Successful authorization	PAYMENT_AUTHORIZED	BackRef
Suspected fraud	FRAUD	BackRef
Reversed reserved amount (two-step payment)	REVERSED	IRN
Successful, completed transaction	COMPLETE	IPN, IDN
Refunded (partial or total)	REFUND	IRN

4 Making transactions with SDK

Before starting the development, you should do tests regarding payment methods specified in the contract. Check the payment method and comprehend what you should accomplish in your own system.

Before making the transaction you may provide a valid e-mail address in the index.php to which SimplePay will send you the transaction notification letter.
It may be provided in the index.php where you should give a value to the BILL_EMAIL variable.

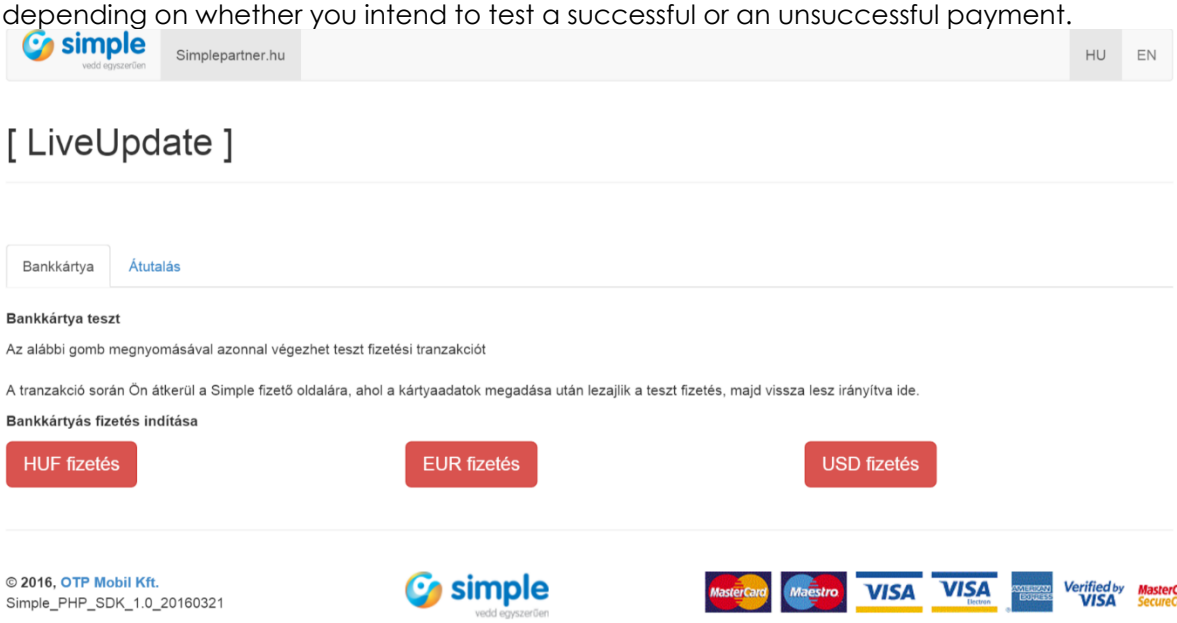
```
$lu->setField("BILL_EMAIL", "sdk_test@otpmobil.com");
```

4.1 LiveUpdate

If you call the installed SDK to the browser, you will get to the following page. Here you can select which payment method, "**Credit card**" or "**Transfer**", you want to test under the tabs on the left.
After selecting the payment method by clicking on the red button according to the currency, you get to the payment page of the given payment method.

NOTE: payment methods which can be used depend on the contract between OTP Mobile Ltd. and the merchant, and you can make transactions with the one which has been activated for the merchant.

The SDK is able to deal with SimplePay merchant accounts with **HUF, EUR** and **USD** currencies. Earlier you could set your account's ID in config.php. Only those currencies will appear on the SDK test interface regarding which you set the **MERCHANT** and **SECRET_KEY** values in the config.php.
If you operate an online store where the customer can choose which currency he/she will use for his/her payments, then the SDK will start the transaction with the data of the appropriate SimplePay's account.
Only the credit card payment may be started in any currencies. Wire transfer may only be made in HUF.
It is possible to select a **test bank account number** on the sandbox payment page depending on whether you intend to test a successful or an unsuccessful payment.



If you set the **DEBUG_LIVEUPDATE_PAGE** variable to the value of 'true' in config.php, the SDK starts the transaction but before redirecting you to the SimplePay's payment page it cancels the process and displays the transaction's all relevant technical data. In addition to the details, it displays the complete form which you should also create during LiveUpdate.

NOTE: you may use this debug setting which appears also on the website only during the development and troubleshooting. You must always disable it during real purchase transactions. Of course, the other debug possibilities in logs may any time be used, where necessary.

4.1.1 Payment page

In case of a **credit card payment,** the payment page is the form which requests card details. After providing the data, the transaction can be started by clicking on "Confirm" button.
In case of a **transfer,** this page is only informative. In this case the transaction is initiated but the payment is not yet made. The customer can see to where and with which notice he/she should transfer. Once the money arrived after the transfer, the system assigns it to the transaction made. The IPN message about the end of a successful payment is the same as the one of credit card payments.

4.1.2 LiveUpdate errors

If there is an error during the transaction due to which the system cannot accept it, it immediately redirects you to the BackRef page. In this case, the **"err"** variable is appended to the BackRef URL whose value refers to the type of error.
Invalid account: the value of **MERCHANT** is not appropriate. In this case you have to check if the copied MERCHANT data are equivalent to the above and if there is not an accidental space in the beginning/end. It is worth checking it if the character coding is UTF-8.
Invalid signature: the value of **SECRET_KEY** is not appropriate. In this case you have to check if there is not an accidental space at the beginning or end of the copied SECRET_KEY data. It is worth checking if the character coding is UTF-8. This error occurs also if some of the compulsory data fail and thus the value of the validating hash is not appropriate. We will discuss this in detail during the development of LiveUpdate.

4.2 BackRef in case of successful and unsuccessful card authorization

If you make a transaction with a credit card number belonging to a successful or unsuccessful payment which can be selected on the payment page and you set the BACK_REF variable in config.php appropriately, you will be redirected to the SDK **backref.php** after the payment page.
This page notifies the customer about the transaction's result.
The demo interface of SDK's sample code has been created to simplify basic merchant tests. You should not use this on live page in a way that you make the transaction from a page similar in view to your own website, then you redirect the customer to the SDK's BackRef with an unchanged appearance because different pages may create mistrust in the customer.
In each case you must redirect the customer to a finishing page clearly identifiable with the page on which the transaction started.

4.2.1 Unsuccessful payment

The transaction's status in this case will be **CARD_NOTAUTHORIZED**
In case of an unsuccessful payment, this is the end of the transaction.
Reason for failure may include that the credit card does not exist or has expired. It is possible that there is not enough money on the card, or there is money but the customer has exceeded his/her daily limit.

The actual reason for the failure may be provided by the bank which issued the card for the customer.

After an unsuccessful payment, the following **compulsory pieces of information** must be displayed
1. The result, that is, "Unsuccessful payment"
2. The following information: Please check the validity of data provided during the transaction. If all data are correct, please contact your bank which issued the card to request the reason for the refusal.
3. SimplePay reference number which is the SimplePay's transaction ID. The SimplePay may provide an answer on the basis of this regarding any questions relating to the transaction
4. Merchant's order ID
5. Date of transaction

Sample information:
Unsuccessful transaction.
Please check the validity of data provided during the transaction. If all data are correct, please contact your bank which issued the card to request the reason for the refusal.

SimplePay reference number: **99013817**
Order ID: **101010514582296094262**
Date: **2016-03-17 16:46:52**

4.2.2 Successful credit card payment

The transaction's status in this case will be **PAYMENT_AUTHORIZED**
In case of a successful card authorization, the BackRef page **is not the end of the transaction but it is only informative** for the customer.
The order can be fulfilled only after the reception of the IPN message described in chapter 5.

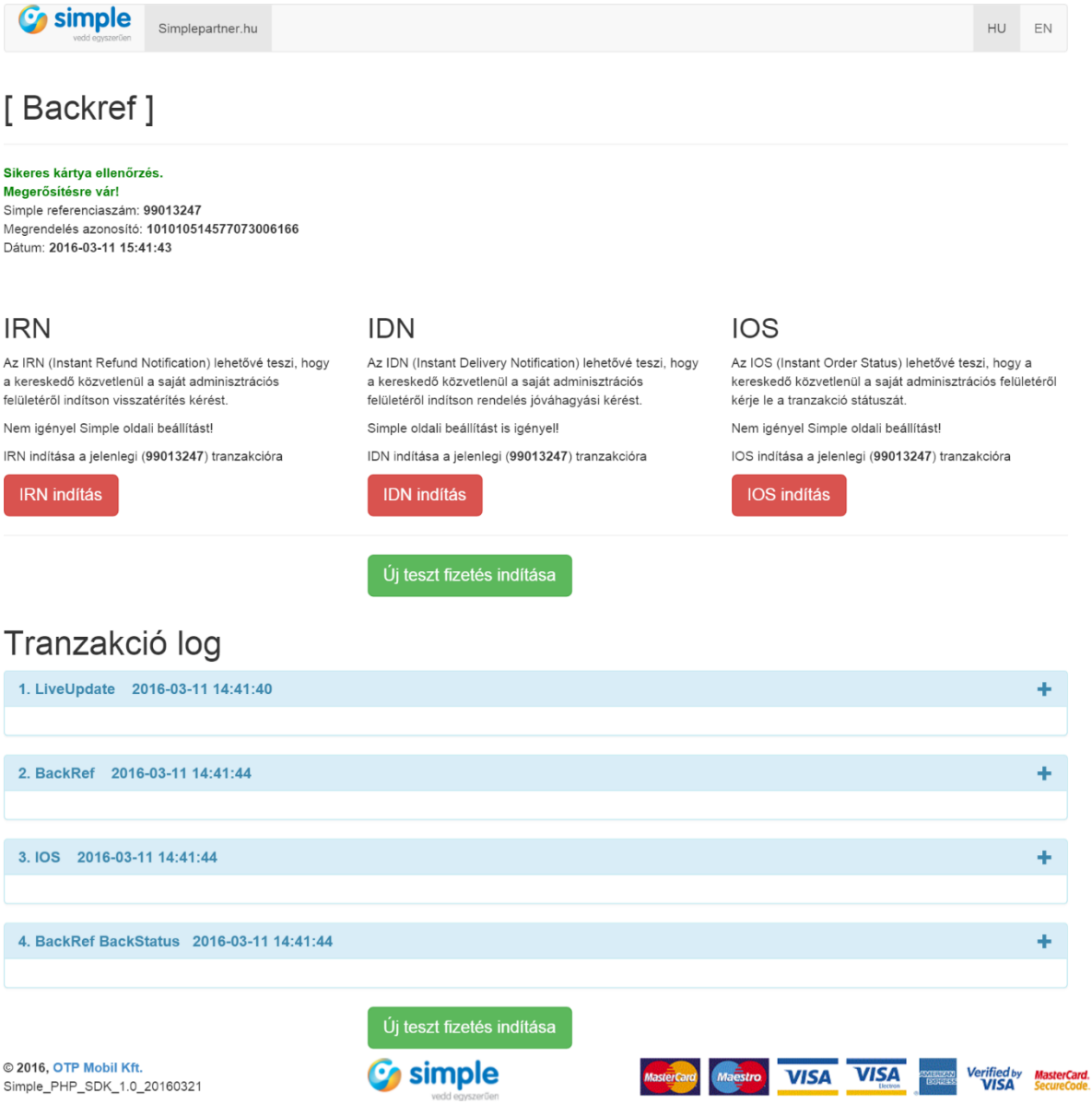
After the successful payment, the following **compulsory pieces of information** must be displayed
6. The result, that is, "Successful payment", or "Successful transaction", or "Successful card authorization", or optionally "Waiting for confirmation"
7. SimplePay reference number which is the SimplePay's transaction ID. The SimplePay may provide an answer on the basis of this regarding any questions relating to the transaction
8. Merchant's order ID
9. Date of transaction

Sample information:
Successful card authorization.
SimplePay reference number: **99013247**
Order ID: **101010514577073006166**
Date: **2016-03-11 15:41:43**

Fraud monitoring runs in the background at this point. Only if it is successful does the system send out the IPN message. The transaction ends with the IPN message. The merchant can fulfil the order at this time.
It is very important to know that the IPN message may arrive in the background at the same time (or even earlier) as the customer is redirected to the BackRef page.

NOTE: You always have to fulfil the order based on the IPN.

This also means that if separate statuses belong to these event in your system, then the IPN must in each case be higher than BackRef, namely, the **status of BackRef must never overwrite the status of IPN.**
In case of a successful payment, both the IRN and the IDN functions may be tested from the BackRef page. These two functions do not form a part of the basic payment process. Their development or abandonment may freely be decided depending on the merchant's business logic. Both functions are available from admin interface also. Additionally, IOS function may also be tested from here.
IRN, IDN and IOS functions will be detailed subsequently.
If logging is set, the BackRef page reads the entries of the transaction. It opens the log in openable blocks separately based on its belonging to the unique function.
The log's reading in browser may solely be used for development or troubleshooting, **do not display it in case of live payments!**



4.2.3 Successful payment by a transfer

The process in case of a wire transfer differs only in payment method until the payment. In this case the payment page is informative showing data necessary for the transfer. On the information page the customer does not have to do anything (the transfer can be carried out from his/her net bank), thus it is not sure whether he/she will be redirected to the merchant's website.
If he/she clicks on the "Back" button and returns to the merchant's website, then he/she has to be notified. The notification in this case differs from the credit card payment.

Upon returning to the BackRef page, the transaction's status in this case will be **WAITING_PAYMENT**
The BackRef page is **not the end of the transaction but only informative** for the customer.
The order can be fulfilled only after the reception of the IPN message described in chapter 5.

After the wire transfer, the following **compulsory pieces of information** must be displayed
1. At this point only an order has been placed, there cannot be information about payment, that is, "Successful order".
2. It has to be indicated that the fulfilment happens on a later date, that is, "The order will be fulfilled after the arrival of the transfer"
3. SimplePay reference number which is the SimplePay's transaction ID. The SimplePay may provide an answer on the basis of this regarding any questions relating to the transaction
4. Merchant's order ID
5. Date of transaction

Sample for information:
Successful order.
The order will be fulfilled after the arrival of the transfer.
SimplePay reference number: **99014232**
Order ID: **101010514599378142316**
Date: **2016-03-29 12:16:55**

4.3 Cancelled or timed-out transaction

In case of a **cancelled** transaction after starting LiveUpdate, the customer gets to SimplePay's payment page but due to some reasons he/she wants to return to the merchant's website. In this case he/she may click on "Cancel" button with which he/she may start the return.
In case of **timeout** after starting LiveUpdate, the customer gets to the SimplePay's payment page but until the time provided in ORDER_TIMEOUT variable expires, he/she cannot start the payment. In this case, at the timeout, the payment page automatically redirects the customer to the merchant's website.
In both cases he/she will return to the URL provided in the TIMEOUT_URL variable in the config.php.
It is very important to take into consideration during the information that **payment was not made in either cases** as the payment has been cancelled before supplying credit card details and making the payment. Therefore, here it is not possible to notify the customer about unsuccessful payment.

After a cancelled or timed-out transaction, the following **compulsory pieces of information** must be displayed
1. The result, that is, "Cancelled transaction", or "Timed-out transaction"
2. The following information: "You cancelled the payment, or the maximum time of transaction has expired!"
3. Optionally: merchant's order ID and date

Sample for information
Cancelled transaction
You cancelled the payment, or the maximum time of transaction has expired!
Order ID: **10101051458143998**
Date: **2016-03-16 16:00:07**

4.4 Receiving IPN

IPN communication is the finishing part of a successful payment process. The IPN call is sent from the SimplePay system to the URL provided by the merchant.
The URL can be provided under the panel "Basic data" of the menu "Account manager/Technical settings" in the merchant's control panel (https://sandbox.simplepay.hu/admin/).
At this point the credit card transaction has gone through the card authorization of bank, and the fraud monitoring. It is very important that the IPN be sent out only in case

of successful payments. If the authorization was unsuccessful it already returned to BackRef as an unsuccessful payment and the transaction ended there.
If the card authorization was successful but it blocked during the fraud monitoring, the SimplePay customer service gets in touch with the merchant and it does not approve the transaction (does not send IPN message) until there is a potential risk of abuse.
In case of a wire transfer, the IPN is sent if the money has arrived from the customer to the account of OTP Mobile Ltd. and based on the transfer's notice it can be assigned to the transaction made on the informative page.
It follows from the above that the IPN always indicates successful payments.
IPN can be sent out also manually, especially for testing and debugging purposes. This is possible only in case of a successful transaction.
If you click on the order reference number in the transaction list on the merchant's control panel, you will open the page showing transaction details. Here, you can send it by clicking on the blue "Send IPN" button.
The result of sending can be seen on the same page under "Communication" tab among "IPNs". As the request will be included in a task list when the request is started it is not sure that the result of sending will promptly be displayed in the list.
By opening the IPN message you may check if the website which processes the IPN message in the given URL is available in your system. If the value of the answer's HTTP code is not 200, for some reason it is not available.
Here you can check data sent to your website and the response which your website sent to the IPN message.

NOTE: IPN sending may be activated also in case of the authorization if it is necessary for the merchant's business logic. The IPN sending options may be configured on the merchant's admin interface under "Account/Technical" on the panel "Merchant notification via IPN".

4.5 Query of IOS transactional data

By using IOS (Instant Order Status) a transaction's current status may be queried from the SimplePay system. The IOS call may any time be initiated from the merchant's page. In case of BackRef, the IOS runs in the SDK in case of each transaction.

It is possible to use additional IOS in case of time-sensitive transactions when you need to get immediate answer.
The sign of the successful transaction is the COMPLETE status. When the transaction has reached this status the system initiates the sending of the mentioned IPN.
The sending process works from a queue, so the sending may delay. On the other hand, the fast IOS query provides data directly from the data of specified transaction. Since this IOS is faster then waiting for IPN. The time difference of the two methods may reach minutes.

You can use additional IOS for IPN in the following way. After the authorization the customer has been redirected to the merchant's website (BackRef page). If the payment is unsuccessful (by RC variable) at this point the transaction will be closed.
It only makes sense to run IOS query in every 10 secs if the authorization is successful (RC=000 and the status is PAYMENT_AUTHORIZED).

If once the status was turned to COMPLETE, you can fulfil the order. After this status there is no need to start further IOS queries.
If the IPN was faster than first IOS, there is no need to start it, because IPN message contains the status of the transaction.

4.6 Initiating IDN

IDN (Instant Delivery Notification) is a function used in case of a two-step payment method when the merchant finalizes all transactions with its own approval.
In case of a two-step payment, the total payment is not made during the LiveUpdate but only the given amount's is reservation. This means that the given amount is reserved

on the customer's account but it is not debited. By sending IDN the debit may be initiated and the payment can be finalized.
In this case only those transactions will be accounted that have been confirmed with IDN message.
The payment is made in two steps.

1. In the 1st step the transaction's total amount is reserved on the customer's card.
2. In the 2nd step the real debit is performed after sending the IDN.

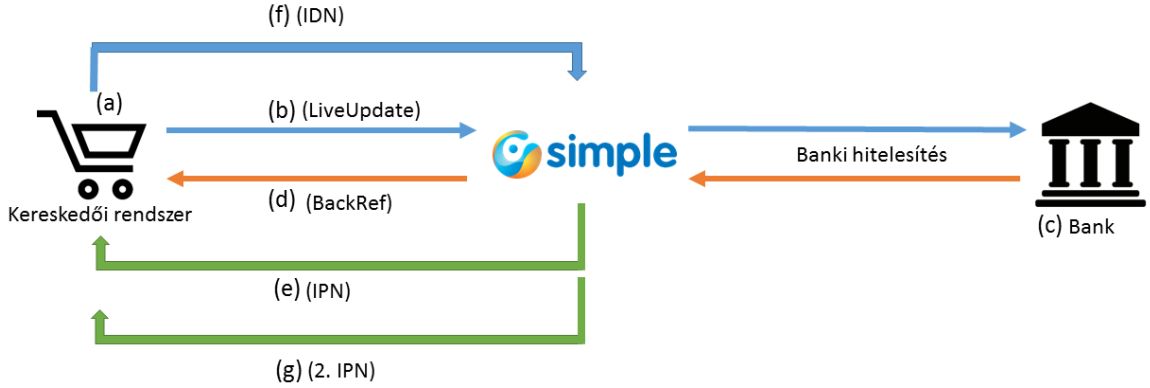
NOTE: the merchant has 21 calendar days to send IDN in case of a two-step payment. If it does not do it until then, the reserved amount will automatically be unlocked.

If the merchant decides before the 21st day that it does not want to debit the amount (e.g. because it cannot fulfil it), then instead of the IDN it may unlock the reserved amount on the customer's account by sending an IRN (Instant Refund Notification).
In order to use it, IDN function has to be configured on SimplePay's side. For more information on use, please contact our colleagues.

4.6.1 The operation of IDN

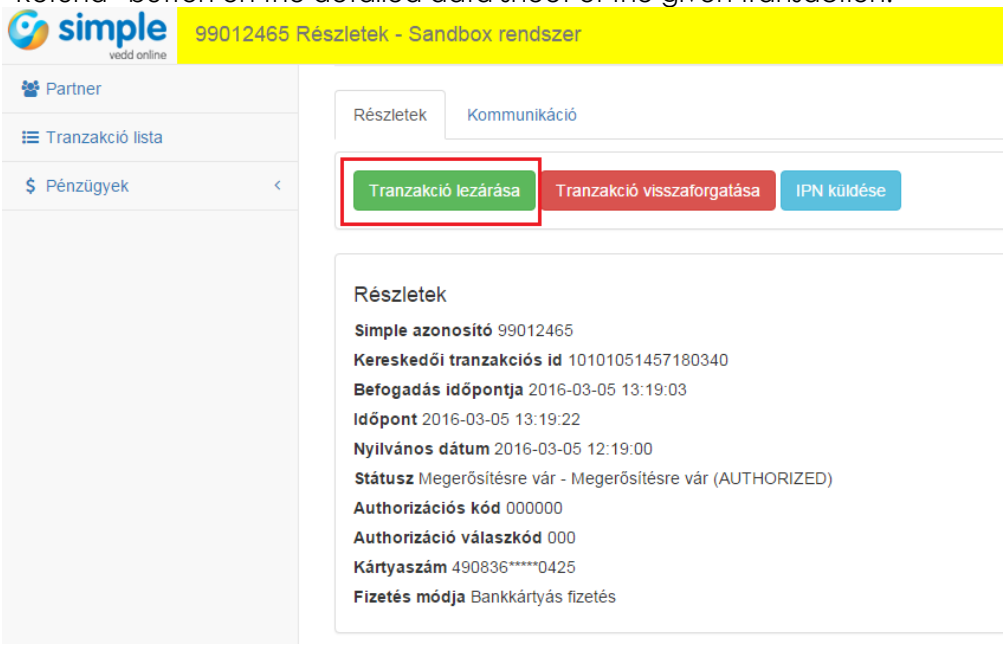
a. The customer selects the products to be purchased on the merchant's website, then initiates the SimplePay online payment.
b. When he/she starts the payment, the order's details are transferred to SimplePay system (**LiveUpdate**). Simultaneously, the customer is redirected to the SimplePay's payment page. The customer may provide his/her credit card details on the payment page. Upon the arrival at the payment page, the transaction is made in SimplePay system.
c. After providing credit card details, the payment is authorized by the bank.
d. The customer is redirected to the merchant's website (**BackRef URL**). Here, the customer shall be notified about the transaction's result based on the provided data.
e. Afterwards, the fraud monitoring and preventive process is executed in the background. If during this the system does not detect any problem, the SimplePay system sends confirmation to the website (**IPN**). At this time, the transaction is not finished, the IPN's status will be **PAYMENT_AUTHORIZED.** The amount has been reserved on the customer's account.
f. After receiving IPN according to point (e), the merchant sends an IDN message to the SimplePay system with which it finishes the transaction. **The first IPN message must be finished before the IDN call.**
g. After the merchant closed it, the SimplePay system sends another IPN message about the successful debit. At this time, the IPN message's status will be **COMPLETE**. Therefore, the amount reserved as per point (e) will be debited on the customer's account.

During the development those included in points **a, d, f** and **e, g** shall be implemented in the merchant's website.



4.6.2 Initiating IDN from the merchant's admin interface

The second step of a two-step payment may be initiated from the merchant's admin interface. By selecting from the "Transaction list" it may be initiated by clicking on "Refund" button on the detailed data sheet of the given transaction.



4.7 Initiating IRN

An IRN (Instant Refund Notification) may be sent against any transaction when the merchant intends to reimburse the transaction's total amount, or if it wants to unlock the amount reserved in the first step in the case of a two-step transaction. In order to use this function there is no need for a configuration on the SimplePay's side.
An IRN may be sent regarding the transaction's total amount after each successful authorization (e.g. in case of a two-step payment) or a total successful payment.
If the payment's status becomes COMPLETE, it can be sent regarding a part amount also.
The amount provided in the IRN has to be bigger than 0, and the aggregated amount of the one or more IRNs cannot be more than the transaction's total amount. The reimbursement is paid in the same currency as the original payment transaction.

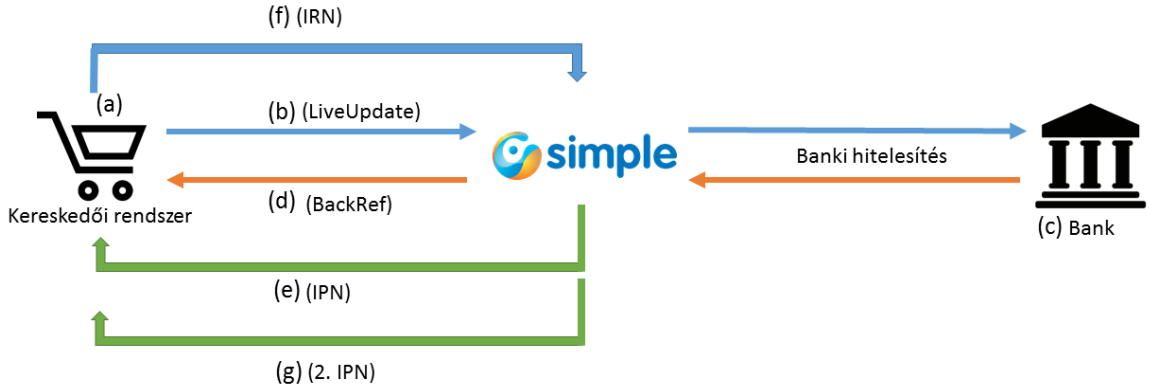
NOTE: The SimplePay system initiates the reimbursement at the same time as the receipt of the IRN request. However, the amount is credited for the cardholder at the time in accordance with the unique accounting system of the bank which issued the card.

4.7.1 The operation of IRN

a. The customer selects the products to be purchased on the merchant's website, then initiates the SimplePay online payment.
b. When he/she starts the payment, the order's details are transferred to SimplePay system (LiveUpdate). Simultaneously, the customer is redirected to SimplePay's payment page. The customer may provide his/her credit card details on the payment page. Upon the arrival at the payment page, the transaction is made in SimplePay system.
c. After providing credit card details, the payment is authorized by the bank.
d. The customer is redirected to the merchant's website (BackRef URL). Here, the customer shall be notified about the transaction's result based on the provided data.
e. Afterwards, the fraud monitoring and preventive process is executed in the background. If during this the system does not detect any problem, the SimplePay system sends confirmation to the website (IPN). At this time, the transaction is not finished, the IPN's status will be PAYMENT_AUTHORIZED.

f. After receiving IPN according to point (e), the merchant sends an IRN message to the SimplePay system with which it reimburses the transaction. The first IPN message must be finished before the IRN call.
g. After the merchant closed it, the SimplePay system sends another IPN message about the successful debit. At this time, the status of the second IPN message will be REFUND.

During the development those included in points **a, d, f** and **e, g** shall be implemented in the merchant's website.



The transaction may be refunded by an IRN message sent from the merchant's page or manually from the merchant's admin system.

4.7.2 Initiating IRN from the merchant's admin interface

By selecting from the "Transaction list", the amount of transaction may be refunded by clicking on "Refund" button on the detailed data sheet of the given transaction.

5 Implementation by means of SDK's sample code

You have to implement the sample code's necessary parts in your own system in the following way.
Since the sample code is functional in itself and can be used for all currently available SimplePay payment methods and currencies, thus from this point **the most important thing is the appropriate knowledge of your own system.**

NOTE: Before you start development, back up your online store and data!

During the development pay special attention to prevent credit card payment being under development or testing from being used in live purchase or payment transaction. Particular attention should be paid to this by the developer especially if he/she implements the SimplePay system in the functional online store as a new payment method.
If a separate developing system is not available and the operator performs the development in the functional online store, he/she must **notify the customer not to** choose credit card payment because **it is still under development/testing.**

5.1 LiveUpdate

Sample: **index.php**
LiveUpdate is the beginning of the payment transaction. At this point, the transaction data are collected in the merchant system, then they are forwarded to SimplePay.
Data include the order's global data, basket content, billing information, delivery information, SimplePay system-specific data (backref URL, timeout URL, etc.)
The necessary data are sent in a form with POST method during which the customer is redirected to the payment page of SimplePay. Here, the sent data are processed and the customer's credit card details are requested, then finally the transaction starts by clicking on "Confirm button.
As a first step, find the point in your own system **where** the order cannot be changed anymore and **all the necessary data are available for the payment**. The whole sample code is in the index.php. If you want to work from a code without graphic interface,

then you should overwrite index.php in the main folder with the index.php in **nogui.zip** in the main folder.
It is necessary to perform file includes at this point at the latest. Here we suppose that the data of your merchant account have been already configured in a way specified earlier in **config.php. SimplePayment.class.php** includes all necessary functions for the payment.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

The content of the above two files should not obviously be in separate files. The application of these **depends entirely on your system's logic of functioning**.
Afterwards, the transaction's currency should be determined which may be HUF, EUR or USD. If more than one currency is available for payment in your online store, you should give value to this based on your own system's variable storing currency. If you use only one currency continuously, then you can provide the value according to this. The merchant account to be used for the transaction will be configured in accordance with the currency.

```
$orderCurrency = 'HUF';
```

Then you should provide the merchant-side unique ID of the transaction.

```
$testOrderId = '42';
```

The next step is to instantiate the class executing LiveUpdate process with config data. The merchant ID must be unique per transaction. After an unsuccessful or cancelled payment, the same ID can be used for another payment.

```
$lu = new SimpleLiveUpdate($config, $orderCurrency);
$lu->setField("ORDER_REF", $testOrderId);
```

setField()
By using setField function you can add fields to the transaction. The first parameter (BILL_EMAIL) is the field's name, the second parameter (sajatemail@example.com) is the field's value.

```
$lu->setField("BILL_EMAIL", "sajatemail@example.com");
```

The possible variables relate to the delivery and billing addresses and can be found in the sample code. Non-compulsory fields are marked.
SDK initiates the transaction with default values but these can be modified uniquely. The new value should be defined by using setField() function for the modification.
The value of the following variables can be modified:

- **LANGUAGE:** the language of payment page can be configured, its value may be CZ, DE, EN, IT, HR, HU, PL, RO, SK
- **ORDER_TIMEOUT:** time in seconds which can be spent on the payment page. If the customer exceeds it, he/she will automatically be redirected to the merchant's page
- **PAY_METHOD:** configures the payment method. Its value is CCVISAMC in case of credit card payment, and WIRE in case of a bank transfer
- **DISCOUNT:** amount of discount. This amount will be deducted from the aggregated amount of products. Zero, or a bigger number, which is smaller than the total price.

- **ORDER_SHIPPING:** shipping cost. This amount will be added to the aggregated amount of products. Zero, or a bigger number.
- **BACK_REF:** overwrites the BackRef URL provided in config.php if you intend to configure it per transaction.
- **TIMEOUT_URL:** overwrites the Timeout URL provided in config.php if you intend to configure it per transaction.

addProduct()
By using addProduct function you can add products to the transaction. The function has one parameter which is an associative array. The product's detail may be provided within the array.

```
$lu->addProduct(array(
        'name' => 'Name of product',
        'code' => 'sku0002',
        'info' => 'Description of product',
        'price' => 159,
        'vat' => 0,
        'qty' => 2
));
```

The following fields are included in the array:
- name: name of product
- code: unique ID of the product in your system
- info: detailed description of the product
- price: price of product
- vat: rate of tax (vat)
- qty: quantity of product

The code field has to include the unique ID of the product. The content of this field has to be unique in your system.
If there is more than one piece in the basket from a product, it is worth to adding it once to the basket and set the number in the quantity (qty) field.
If you provide a gross price, then you should enter 0 value to the ORDER_VAT field. If you provide net price, then you should provide the VAT rate in the vat field with a whole number without a percent sign, e.g. 27.
NOTE: When giving the price, it is recommended to provide a gross amount without decimal digits in HUF currency (in the case of EUR and USD, two decimal digits where the decimal is the dot character) with a VAT value of 0.
If you provide fraction with more decimal digits than needed (e.g. you obtain the net amount and the rate of VAT by dividing from the gross amount), then the system will round because of which the amount in the online store may differ from that of the payment page.

createHtmlForm()
SDK generates one form from the provided data. By sending this, the transaction starts.
The createHtmlForm function has three parameters:
The first parameter is the field's name which can freely be overwritten with a value suitable for you.
The second parameter determines the form's starting method where the possible values
- **link**: generates a link with which the payment may be started
- **link**: generates a button with which the payment may be started
- **auto**: automatically sends the form during generation

The third parameter is the text of element starting the form which can freely be overwritten with a value suitable for you.

```
$display = $lu->createHtmlForm('SimplePayForm', 'button', 'Start Payment');
```

errorLogger()
After generating the form, the possible errors or the debug data among config values in case of activating DEBUG_LIVEUPDATE are written in the log.

```
$lu->errorLogger();
```

Display of form
We display the generated form. If it has been generated with "auto" value, the customer will promptly be redirected to the payment page. If it has been generated with "button" or "link", then the customer will be redirected by clicking on one of them.

```
echo $display;
```

NOTE: If, during LiveUpdate you provide an error with such data that cannot be interpreted by SimplePay, it will immediately redirect you to the website from where the transaction arrives. In this case, the payment has not been performed.
The error message in the URL will be available in the "**err**" variable.

5.1.1 Selecting payment method

The following payment methods are currently available in the SimplePay system

- credit card payment: CCVISAMC
- transfer: WIRE

During LiveUpdate the payment method with which we intend to start the transaction should be determined. The default method in the SDK is the credit card payment. If you would like to transfer, the payment method should be modified in the following way:

```
$lu->setField("PAY_METHOD", 'WIRE');
```

The payment method must be provided before running createHtmlForm() function.

5.2 BackRef

Sample: **backref.php**
The customer provides the credit card details on the payment page, then after starting the payment the customer is redirected to this information page in the browser.
First, you should perform the includes of the necessary files just like in the case of LiveUpdate.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

Afterwards, it is important to provide the order ID of the transaction to the online store, that is, the data provided in ORDER_REF field during LiveUpdate. In addition, it is important to provide the transaction's currency because as during LiveUpdate the appropriate merchant account is applied based on the currency.
It is practical to provide both above variables in the BackRef URL. During LiveUpdate, the SDK automatically assigns both in the following way.
So if you provided the following as BackRef URL:

```
http://domainnev.hu/simplepay/backref.php
```

the SDK started the payment by expanding it to the following:

```
http://domainnev.hu/simplepay/backref.php?order_ref=42&order_currency=HUF
```

In this case, the order ID and its currency can be taken over in the following way, and authentication can be performed with these data.

```
$orderCurrency = (isset($_REQUEST['order_currency'])) ? $_REQUEST['order_currency'] :
'N/A';
$orderRef = (isset($_REQUEST['order_ref'])) ? $_REQUEST['order_ref'] : 'N/A';
```

Order_ref is important because the Instant Order Status (IOS) call running in the background identifies the transaction based on this and can return the transaction's details including its status and payment method.
The next step is to instantiate the class executing BackRef process with config data.

```
$backref = new SimpleBackRef($config, $orderCurrency );
$backref->order_ref = $orderRef;
```

It is the BackRef page's task to interpret the transaction's details and authenticate its sender. If the SimpleBackRef class has been instantiated in the way specified above, then both tasks are performed by the SDK'S single function which indicates the authentication with true/false value.

```
if($backref->checkResponse()){
        $backStatus = $backref->backStatusArray;
        //information about success
        //running own codes
} else {
        $backStatus = $backref->backStatusArray;
        //information about failure
        //running own codes
}
```

The checkResponse() function authenticates the call based on the value of the received ctrl variable. In addition to authentication, it queries the transaction's details in the background by using IOS. On the basis of the IOS, the above sample code includes the transaction's details in the $backStatus array in the following form:

```
Array
(
        [BACKREF_DATE] => 2013-11-22 09:56:18 //time of transaction
        [REFNOEXT] => 82196101521385110558 //merchant's order ID
        [PAYREFNO] => 99400321 //transaction's SimplePay ID
        [ORDER_STATUS] => PAYMENT_AUTHORIZED //transaction's SimplePay status
        [PAYMETHOD] => Visa/MasterCard/Eurocard //payment method
)
```

There is an example code in the sample code both in the successful and the unsuccessful branches which display information messages on the basis of the $backStatus content by using payment method and status.
Different pieces of information must be displayed in case of a credit card payment and a wire transfer. The reason for the difference is that in case of a credit card payment, the customer is notified about the payment made, while in case of a transfer, only the transaction is made but the payment is only performed when the customer actually transfers the necessary amount. The sample code handles both cases.

5.3 IPN

Sample: **ipn.php**
After the successful payment the SimplePay's server sends message to the merchant with POST method. This is the end of the successful transaction; the order can be performed in accordance with this.
First you should perform the includes of the necessary files just like in the case of LiveUpdate and the BackRef.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

Then, it is important to provide the transaction's currency for the online store,

```
$orderCurrency = (isset($_REQUEST['CURRENCY'])) ? $_REQUEST['CURRENCY'] : 'N/A';
```

The next step is to instantiate the class executing IPN process with config data.

```
$ipn = new SimpleIpn($config, $orderCurrency);
```

The function of IPN is duplex; on the one hand it validates the message, on the other hand it reports back to the SimplePay in case of a successful validation.

```
if($ipn->validateReceived()){
        echo $ipn->confirmReceived();
        /**
         * Running of own codes after a successful payment
         */
}
```

validateReceived()
It checks the authenticity of IPN call in order to be sure where the message comes from.
confirmReceived()
If the validation is successful, this function reports back to SimplePay.
The confirmation is created and displayed by the function in the following format:

```
<EPAYMENT>20160324095755|acb62735628945200b111f23e381b6f3</EPAYMENT>
```

If you want to configure this, the automatic display may be disabled before running confirmReceived() in the following way:

```
$this->echo = false;
```

If you disabled it, you will find the necessary response you should display in the function's output.

```
$epayment = $ipn->confirmReceived();
echo $epayment;
```

On the basis of this confirmation, SimplePay system obtains information that the merchant has received and processed the IPN message about the success of transaction. The confirmation is vital because if it does not take place or it is not appropriate, the IPN is attempted to be sent again.
After the first IPN, the logic of resend after a non-received IPN is the following:

- after 5, 10, 15, 30, 45 minutes
- after 1, 2, 3, 6, 9, 12, 18 hours
- after 1, 1.5, 2, 2.5, 3 days

After these, the server no longer sends out IPN message regarding the given transaction.
Of course, IPN messages may be sent out manually from the merchant's admin interface.
The transaction's status at the time when the first IPN is sent out is **COMPLETE**
It may be different in the case of a two-step payment, and it requests IPN message about card verification.
In this case, the status is the same as the **PAYMENT_AUTHORIZED** value on the BackRef page.

5.4 IOS

Sample: **ios.php**
The merchant can query the status of any transaction started from the SimplePay system. The request is provided in each case with the current status.
First you should perform the includes of the necessary files just like earlier.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

Then, you should provide the transaction's currency and merchant ID for the online store.

```
$orderCurrency = 'HUF';
$orderexternalId = '42';
```

The next step is to instantiate the class executing IOS request with config data. Here you can log the potential errors by means of errorLogger() function.

```
$ios = new SimpleIos($config, $orderCurrency, $orderexternalId);
$ios->errorLogger();
```

The $ios->status array includes the result of query.

5.5 IDN

Sample: **idn.php**
First you should perform the includes of the necessary files.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

Then, you should provide the transaction's currency.

```
$orderCurrency = 'HUF';
```

The next step is to instantiate the class executing IDN request with config data.

```
$idn = new SimpleIdn($config, $orderCurrency);
```

Then you should create the array containing transaction's details. The IDN message is started for the reserved amount of this transaction, as a result of which the debit occurs.
NOTE: IDN may only be applied for the transaction's total amount.

```
$data['REFNOEXT'] = '42';
$data['ORDER_REF'] = '99014589';
$data['ORDER_AMOUNT'] = '150';
$data['ORDER_CURRENCY'] = $orderCurrency;
$data['IDN_DATE'] = @date("Y-m-d H:i:s");
```

We start the IDN request with the necessary data

```
$response = $idn->requestIdn($data);
```

The **$response** array includes the IDN's result. The response can be authenticated plus the error logging can be started with the code below.

```
if (isset($response['RESPONSE_CODE'])) {
        if($idn->checkResponseHash($response)){
        /**
         * Running own codes after IDN sending
         */
        }
}
$idn->errorLogger();
```

5.6 IRN

Sample: **irn.php**
First you should perform the includes of the necessary files.

```
require_once 'sdk/config.php';
require_once 'sdk/SimplePayment.class.php';
```

Then, it is important to provide the transaction's currency for the online store.

```
$orderCurrency = 'HUF';
```

The next step is to instantiate the class executing IRN request with config data.

```
$irn = new SimpleIrn($config, $orderCurrency);
```

Then you should create the array containing transaction's details. The IRN request is started for the reserved/debited amount of this transaction, as a result of which the refund occurs.
ORDER_AMOUNT includes the amount of the original transaction.
AMOUNT includes the amount to be refunded.

```
$data['REFNOEXT']  = '42';
$data['ORDER_REF'] = '99014589';
$data['ORDER_AMOUNT'] =  '150';
$data['ORDER_CURRENCY'] = $orderCurrency;
$data['IRN_DATE'] = @date("Y-m-d H:i:s");
$data['AMOUNT'] =  '100';
```

We start the IRN request with the necessary data.

```
$response = $irn->requestIrn($data);
```

The **$response** array includes the IRN's result. The response can be authenticated plus the error logging can be started with the code below.

```
if (isset($response['RESPONSE_CODE'])) {
            if($irn->checkResponseHash($response)){
            /**
             * Running own codes after IRN sending
             */

            }
}
$idn->errorLogger();
```

6 Logos and information pages

SimplePay logo must be displayed in a permanently visible place of the payment acceptance pages (e.g. in the footer), or during the selection of the payment.
The SimplePay logo is protected by trade-mark and copyright, thus the merchant shall only use SimplePay logo in a way and for the purpose determined in SimplePay's Terms and Conditions.
The logo shall not be transparent and its size may only be modified to be clearly visible.
The file including SimplePay logos can be downloaded from:
http://simplepartner.hu/PaymentServices/simplepay_logo.zip
The logo shall be a link to the payment information at the same time. The Payment Information to be linked on logos is available under the following URL:
In Hungarian: http://simplepartner.hu/PaymentService/Fizetesi_tajekoztato.pdf
In English: http://simplepartner.hu/PaymentService/Payment_information.pdf

The logo and the linked payment information may be displayed with the following sample code.
The content of src (marked with red) is the path of logo on your server.

```
<a href="http://simplepartner.hu/PaymentService/Fizetesi_tajekoztato.pdf" target="_blank">
        <img width="197" height="27" src="images/SimplePay_logo.png" title="SimplePay - Online credit credit card payment" alt="SimplePay customer information">
</a>
```

7 Data transmission declaration

Since the merchant shall not transfer the order/customer's details, thus **the customer shall** expressly **accept the data transmission declaration.**
There are several options for placing the declaration
- in the respective Terms and Conditions of the page
- in the respective Data Transmission Declaration of the page
- directly displayed during payment

NOTE: it is not sufficient to place the declaration in the website if the customer does not find and accept it.

It may be accepted with a checkbox, or clearly indicating when the transaction is started that with initiating the transaction the customer accepts the declaration.
The declaration must be completed with real merchant data in the highlighted parts.

Company: company name in the SimplePay contract
Company Address: company address in the SimplePay contract
Website: domain name in the SimplePay contract. In case of mobile application add the application name as well.
Transmitted data: all of transmitted customer data, e.g. customer's name, customer's email, etc.

Hungarian declaration
Tudomásul veszem, hogy a(z) **[Kereskedő cégneve]** (**[székhelye]**) adatkezelő által a(z) **[Fizetési Elfogadóhely webcíme]** felhasználói adatbázisában tárolt alábbi személyes adataim átadásra kerülnek az OTP Mobil Kft. (1093 Budapest, Közraktár u. 30-32.), mint adatfeldolgozó részére. Az adatkezelő által továbbított adatok köre az alábbi: **[Kereskedő által továbbított adatok megnevezése]**

Az adatfeldolgozóáltal végzett adatfeldolgozási tevékenység jellege és célja a SimplePay Adatkezelési tájékoztatóban, az alábbi linken tekinthető meg: http://simplepay.hu/vasarlo-aff

English declaration
I acknowledge the following personal data stored in the user account of **[Company Name]** (**[Company address]**) in the user database of **[Paying Acceptance Web site]** will be handed over to OTP Mobil Ltd. (1093 Budapest, Közraktár u. -32.) and is trusted as data processor. The data transferred by the data controller are the following:
[data transmitted by the trader]

The nature and purpose of the data processing activity performed by the data processor in the SimplePay Privacy Policy can be found at the following link: http://simplepay.hu/vasarlo-aff

8 Testing

Every online store is tested by SimplePay before activation.
After the development is finished, in order to do SimplePay-side tests provide us with the path of the developed payment system, and all the necessary information to reach the payment. It is not necessary to perform the development in the domain name found in the contract, thus we can test it anywhere where it is available to us.
During tests our colleagues check the following:

- SimplePay payment is inserted in the purchase process on the merchant's page
- Technically, the transactions run appropriately
- The logo and information notes as well as the data transmission declaration are displayed appropriately

Compulsory test points

1. **Successful transaction**
    a. The transaction runs appropriately until the end
    b. The information notes specified in chapters 4.2.2 and 4.2.3 are displayed on the BackRef page
2. **Unsuccessful transaction**
    a. The transaction runs appropriately until the end
    b. The information notes specified in chapters 4.2.1 are displayed on the BackRef page

3. **Cancelled transaction**
    a. The transaction runs appropriately until the end
    b. The information notes specified in chapters 4.3 are displayed on the Timeout page
4. **Logos and information pages**
    a. SimplePay logo is displayed as per those included in chapter 6
5. **Data transmission declaration**
    a. The necessary declaration is displayed as per those included in chapter 7

The SimplePay tests are in each case performed in browsers and are exclusively related to the SimplePay requirements and payment transactions. Conclusively, **the tests do not include the following**:

- entry to the server hosting the website (ssh, or any other channels)
- web-based entry to the admin interface of the server hosting the website
- configuration of the server/database hosting the website
- entry to the website's database
- entry to the website's admin interface
- checking and editing the website's source code
- testing the website's technical operation (beyond SimplePay's requirements)
- testing the website's business logic (beyond SimplePay's requirements)
- testing the website's appearance (beyond SimplePay's requirements)

9 Support

For further information or technical support, please contact us at itsupport@otpmobil.com.
For faster processing, please provide us with the data based on which we can identify the problem or your query.

Transaction
If you have questions regarding transactions, provide us with the payment's **SimplePay** ID. The **ID is composed of eight digits** and its format is **9xxxxxxx** in the case of sandbox, and **2xxxxxxx** in live transactions.

Merchant account
You should provide us with the **merchant account** ID within the SimplePay system in connection with technical configurations. The ID is the account's **MERCHANT value**.

Payment system
Which system your question relates to. In case of tests: **sandbox system**, in case of real payment transactions: **live system**

Activation
In case of activation tests, please write us to itsupport@otpmobil.com providing the following information

- under which contracted domain name the testable payment was made
- which account you use (MERCHANT)
- where we find the testable system

10 Appendix

10.1 Social login in case of webview

On the SimplePay payment page, the customer has an option to use a previously registered card saved in Simple mobile application. In this case, there is no need to fill card data fields on payment page. In order to use the registered card, the customer has to authenticate into Simple's system on payment page, choose the registered card, then he/she can initiate the payment. The authentication can be carried out with Google of Facebook social login as well.

This card registration feature does not require any developing on merchant side.

However, when a payment page appears enclosed into a webview in an application, there is a potential difficulty. In case of some older webview component, Google and Facebook denied the social login.

SimplePay payment page is not able to solve this issue, because the problem needs to be handled on the client (mobile) side. We have a few suggestions for handling this issue.

10.1.1 Android

Using Chrome Custom Tab (preferred solution)

Instead of WebView, opening Chrome Custom Tab can happen the following way

```
CustomTabsIntent.Builder builder = new CustomTabsIntent.Builder();
CustomTabsIntent customTabsIntent = builder.build();
CustomTabsIntent.launchUrl(MainActivity.this, Uri.parse(url));
```

You can set launchMode in the Activity which opens the CustomTab. It is in the AndroidManifest.xml file.
The launchMode options are the followings: singleTop, singleTask or singleInstance, depending on the application structure.
In most cases, the singleTop is the proper option.

Defining Intent-filter for CustomTab with completely uniqe scheme in the AndroidManifest.xml file.

```
<activity android:name=".MainActivity" android:launchMode="singleTop">
        <intent-filter android:priority="100">
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <action android:name="android.intent.action.VIEW" />
        <data
                android:host="uniqHost"
                android:scheme="uniqScheme" />
        </intent-filter>
</activity>
<activity android:name=".WebViewActivity" />
```

In redirect URL you have to use uniqe scheme://host structure, based on the earlier defined host and scheme data.

Further documentation:
https://github.com/GoogleChrome/custom-tabs-client

10.1.2 IOS

First and prefered soulition is SFSafariViewController instead of UIWebView UI component. With this method, the login still works inside the application.

```
SFSafariViewControllerConfiguration *c = [[SFSafariViewControllerConfiguration alloc]
init];
SFSafariViewController *sf = [[SFSafariViewController alloc] initWithURL:[self testUrl] conf
iguration:c];
[self presentViewController:sf animated:YES completion:nil];
```

Other solution is opening the Safari browser. In this case, the application closes itself and opens a browser and after successful login, the user gets redirected into the application.

```
[[UIApplication sharedApplication] openURL:[self testUrl] options:@{} completionHandler:nil];
```