
SimplePay 2.x (API v2)

Development Documentation

Payment process and development

05.11.2025



Table of contents

1	Short summary	8
1.1	SimplePay Online Payment v2	8
1.2	Definitions.....	8
1.3	The development and deployment process of the online payment	10
1.4	Timing of the development	10
1.5	Deployment steps involving the merchant	11
1.6	The process of the payment transactions	11
1.6.1	Bank card payment	11
1.6.2	Transfer	12
1.6.3	Pay with Simple account, Apple Pay and Google Pay wallets or qvik QR code (at start endpoint).....	12
1.6.4	qvik (at starteam endpoint)	12
1.6.5	SZÉP card payment	13
1.7	SANDBOX and the live payment system.....	13
1.8	Downloadable resources	14
2	Transaction and its statuses	14
3	Implementation.....	15
3.1	General message format.....	16
3.2	Validating messages	17
3.3	start – creating a bank card payment transaction.....	19
3.4	start – creating a transfer transaction (Instant Transfer).....	20
3.4.1	Transfer timeout settings.....	21
3.5	Transaction start using strong customer authentication (SCA, 3DS)	21
3.6	3DS challenge	22
3.7	Further variables.....	23
3.8	start – response.....	25
3.9	Failed API requests.....	26
3.10	What needs to be done at the merchant's site before redirecting to the payment page	26
3.11	SimplePay payment page.....	26
3.12	back – information on the merchant website	27
3.13	Information depending on the result of the transaction	28
3.13.1	Cancelled payment.....	29
3.13.2	Timeout.....	29

3.13.3	Payment failed.....	30
3.13.4	Successful payment.....	31
3.14	ipn – notification on transaction statuses and executability.....	31
3.15	finish – finishing two-step transactions.....	34
3.16	refund – managing refunds.....	36
3.17	query – querying transaction details.....	37
3.18	transactioncancel – cancellation of the transaction.....	40
	qvik (AFR2.....	41
3.19	, EAM) payments.....	41
3.19.1	Merchant account general settings.....	42
3.19.2	Limitations.....	42
3.19.3	tracking the qvik payment process.....	42
3.19.4	qvik – start payment functions.....	42
3.19.5	qvik – deeplink generation without payment page – starteam.....	43
3.19.6	qvik – refund.....	44
3.19.7	qvik – transaction cancellation.....	44
3.19.8	qvik – IPN.....	44
3.19.9	qvik – timeout processes.....	44
3.19.10	qvik – settlements.....	45
3.20	ApplePay payment partner-side solution.....	45
3.20.1	Merchant registration for ApplePay payments.....	45
3.20.2	Positioning of Well-known file for ApplePay payments.....	46
3.20.3	ApplePay payment process.....	46
3.20.4	ApplePay payment frontend.....	47
3.20.5	ApplePay payment backend, startapplepay.....	47
3.20.6	ApplePay payment backend, doapplepay.....	49
4	PHP SDK.....	50
4.1	The structure of the SDK.....	50
4.2	SDK config settings.....	51
4.2.1	Merchant account data.....	51
4.2.2	URL's.....	52
4.2.3	Switching between test and live transactions.....	52
4.2.4	Logging.....	53
4.3	Setting up SDK IPN.....	53
4.4	start – create transaction.....	54

4.5	start – response.....	55
4.6	SimplePay payment page.....	57
4.7	back	57
4.8	Information on the merchant webpage, depending on the result of the transaction 58	
4.9	IPN.....	58
4.10	finish – managing two-step transactions	59
4.11	refund – managing refunds.....	61
4.12	query – querying the transaction data.....	62
5	Sample code.....	66
5.1	Calculation, verification of HASH.....	66
5.1.1	PHP solution.....	67
5.2	API requests	67
5.2.1	PHP solution.....	67
6	Error codes.....	68
7	Logos and information pages.....	79
8	Data Transfer Declaration.....	80
9	Testing.....	81
9.1	Start of testing.....	81
9.2	The purpose of the tests	81
9.3	The testing site	81
9.4	Technical background of the merchant system	81
9.5	Using third-party solutions.....	81
9.6	Mandatory test elements for bank card payments	82
9.6.1	Successful transaction	82
9.6.2	Failed transaction	82
9.6.3	Timeout.....	82
9.6.4	Cancelled transaction	82
9.6.5	Display of the SimplePay Logo.....	82
9.6.6	Data Transfer Declaration.....	82
9.7	Components not tested.....	82
10	SZÉP card acceptance.....	84
10.1	SZÉP Card merchant's transaction identifier length	84
10.2	SZÉP Card test in sandbox environment	84
10.3	SZÉP Card and 3DS	84

11 Support	85
Annexes	86
I. Payment page implementation in mobile client / social login to Simple account	86
Android	86
Google Pay payment testing in Sandbox environment	87
iOS.....	87
Apple Pay payment testing in Sandbox environment	88
II. Redirect between the Simple application and the merchant application (deeplink) .	89
Pre-payment deeplink from merchant app to Simple app	89
Post-payment redirect deeplink from the Simple app to the merchant app	89
Android	90
iOS.....	91
III. EMV 3D Secure	92
Embedding the SimplePay	94
IV. payment page in an iFrame	94

Document History

Date	Version	Change
30.09.2018	180930	Original issue
01.10.2018	181001	Adding the query function and the sample code
02.10.2018	181002	Adding the finish function and the sample code
04.10.2018	181004	Adding the refund function and the sample code
17.10.2018	181017	Adding OneClick payments and sample codes
30.10.2018	181030	Hash calculation adding and API request sample codes
01.03.2019	190301	Removing OneClick payments from the documentation Adding SDK IPN functions
03.06.2019	190603	Clarifications <ul style="list-style-type: none"> - source code details - two-step payment - SDK
07.01.2020	200107	Adding Strong Customer Authentication (SCA, 3DS) Adding further optional variables to Start function
03.06.2020	200603	Increasing transaction identifier size Regulation of transfer days on merchant's admin interface Navigation from Simple application to the merchant's application Clarifications
15.08.2020	200815	Clarifications for 3DS data fields
03.09.2020	200903	Update of error code
06.09.2020	200906	Sandbox payment page 3DS Challenge test card
18.01.2021	210118	Extension of error codes
15.06.2021	210615	Use of new IT support reporting system (1.3) IPN messages for unsuccessful finished statuses (3.15) Revision of annexes (I and II) related to mobile application Update of error codes (6)
21.10.2021	211021	Update of sample codes (3.2)
12.10.2022	221012	Extension of error codes: 2077, 3500–3508, 5043 (6.) Correction of misspelling "description" → "desc" (3.8) New payment page languages: AR, ZN (3.1) Enhancement of mobile application deeplink functionality (Annex II)
31.01.2023	230131	Temporary suspension of Service Desk, use of itsupport@simplepay.com
10.07.2023	230710	Extension of error codes: 6xxx (6.)
01.03.2024	240301	IPN server IP address update, IPN status updates (Refund), SZÉP Card acceptance, Extension of error codes
24.07.2024	240724	New IPN IP address domains (3.15) Management of AFR2, qvik/EAM payments (3.20) SZÉP Card payments (10)

14.08.2024	240814	New payment page iFrame function change (Annex IV)
22.09.2024	240922	FAQ clarifications (3.20)
24.04.2025	250424	Complete document revision, Update of 3.8 further variables
01.07.2025	250701	Corporate identity modifications
31.07.2025	250731	Update of transaction and IPN statuses in ApplePay payment partner side solution
29.08.2025	250829	Document revision, correction of typos
03.09.2025	250903	Error codes update
16.09.2025	250916	Error codes update
29.09.2025	250929	Error codes update
06.10.2025	251006	Validating messages HASH pattern updates
05.11.2025	251105	Error codes update, SZÉP card Transaction ID length

1 Short summary

1.1 SimplePay Online Payment v2

SimplePay is SimplePay Plc.'s online payment solution. This documentation was created for the v2 version of the merchant REST API and the transaction management to be used for SimplePay payments.

This API provides its services via an interface that is different from that of the earlier v1 API. The interface communicates at all endpoints using data in the JSON format. For this reason, for each topic, this documentation facilitates the understanding of the technical background in two ways.

On the one hand, with **JSON examples** necessary for a particular point, based on which the implementation of the online payment **can be realised in any programming language**. On the other hand, with **PHP SDK sample codes downloadable** from the merchant admin interface. The SDK is a complete payment sample code that creates the data in the above-mentioned JSON format, communicates with the API, and handles the responses received from the API.

The purpose of the API is to receive the merchant transaction requests; its use is limited to the management of the transactions. As a result of this, no change is caused in terms of settlement-related and financial issues to those partners where the SimplePay payment system with v1 API is already in use.

This documentation does not aim to detail the differences between the v1 and the v2 interfaces; however, it may refer to them where it can facilitate understanding.

1.2 Definitions

The definitions frequently used in the documentation.

API v1: SimplePay interface, which is used by merchant systems using the SimplePay service to launch their transactions. The v1 interface continues to be available for those merchant systems that implemented SimplePay payment using this interface.

API v2: the new SimplePay interface available since 2018. Its operation is described in this documentation.

Authorisation: verifying the bank card details provided on the payment page, and checking the amount that can be charged to the card. On the basis of this, the actual payment is authorised or rejected by the bank issuing the bank card.

AFR2: Instant payment system. A money transfer system operating on the basis of harmonised data input standard (QR, NFC, deeplink). AFR2 is accessible only in HUF.

EAM: The Hungarian acronym of Standardised Data Input Modes in the case of AFR2 payments

qvik: the type of transaction initiated in the AFR2 payment system

One-step payment: the card is charged once the payment has been initiated.

Two-step payment: after paying, the amount entered will be blocked on the customer's card, but it will not be charged at that time yet. Following this, the merchant has 21 days to initiate the charging or to release the blocked amount. If it is not charged until day 21, the blocked amount will automatically be released and no charge can be initiated for that transaction.

Sandbox: a testing system with functions simulating the live system. The transactions performed therein do not involve real payment. Only transactions necessary for development-related testing can be initiated in this system. No financial settlement is performed in the sandbox system.

Live system: the system used for managing real-life transactions. It is only made available after the development and the testing has successfully been carried out on the testing system.

SDK (Software Development Kit): a sample code to realise payments that can be used for developing and implementing proprietary systems.

Transaction: payment initiated from the merchant's website, on the outcome of which the SimplePay system will provide feedback to the merchant. The subject of the transaction (the product to be paid for) can be anything within the contractual terms. The payment transaction is initiated from the merchant's system and closes with the IPN (Instant Payment Notification) feedback on the payment. The payment transaction is only a part of the merchant-side purchase process.

Payment page: a component of the SimplePay system. This is where the customer provides the card details and where they receive the necessary information in case of a bank transfer.

Merchant account: a unique merchant administration interface that exists in both the sandbox and the live system. Its unique IDs are used to initiate payment transactions.

Currency: the currency of the transaction that can be Forint (HUF), Euro (EUR) or Dollar (USD). The currency is locked with the merchant account.

Payment method: bank card, transfer (and qvik), instant payment (EAM), Simple, Google Pay and Apple Pay wallets, and OTP SZÉP card can be used in the SimplePay system.

Logo: the SimplePay logo displayed on the merchant's website.

Data Transfer Declaration: the customers' acceptance of transferring the customer data to SimplePay. It may be carried out upon registering on the merchant website or prior to initiating the payment transaction.

Merchant tests: tests carried out by the merchant during the development in order to check if operation is in line with the documentation.

SimplePay tests before deployment: testing is done by the SimplePay IT support, at the merchant's request. Only those websites or payment systems can be deployed where SimplePay tests have been successful and the required technical and informational components are in line with the documentation.

Test bank card: a bank card can be selected on the payment page of the sandbox system depending on whether the developer wishes to initiate a successful or an unsuccessful OTP SZÉP card, Google Pay or Apple Pay transaction.

Deployment: following the successful tests, the merchant account will be activated in the live system, too. Online payments can be received through this account.

start: generating a payment transaction via the API.

back: information page in the merchant's system. This is where the customer lands after paying on the SimplePay payment page.

IPN: background communication between SimplePay and the merchant's system. It provides information of successful payments. The merchant can begin the performance once they have received this.

query: querying the status of the transaction.

finish: in case of a two-step payment the second step is initiating the closing of the transaction (charging) by the merchant

refund: refunding initiated by the merchant

PSD2: second Payment Services Directive of the European Union on digital financial services

SCA: (Strong Customer Authentication) strong customer authentication, or PSD2 principle-based 2-factor customer authentication

3DS: card company standard for implementing two-factor authentication (SCA)

1.3 The development and deployment process of the online payment

The developer is required to perform the following tasks during development.

- Initiating a transaction with the details of the customer and the cart.
- Informing the customer returning from the payment page about the result of the transaction
- Receiving the IPN message
- Placing the SimplePay logo on the website
- Placing the SimplePay Data Transfer Declaration on the website

After the execution of the above tasks, completion of payment development can be indicated at the following email address:

itsupport@simplepay.com

Following this, our colleagues will check the online payment, too. If the tests are successful, deployment of the payment starts.

Specific SimplePay side tests are performed pursuant to chapter "Testing" in this documentation.

1.4 Timing of the development

When implementing the SimplePay payment the followings must be considered in the plans:

- The time that you or your developer require for the development
- The time required by SimplePay to test the website/payment system
- Technical lead time of the deployment

If development has been completed and the developer has checked every required point based on **Chapter 9**, a request may be sent to itsupport@simplepay.com for the website to be tested by SimplePay.

Testing will be carried out within **1–3** business days after the request. Our colleagues will send feedback on the result of the testing in every case.

If the SimplePay technical tests are successful, too, the payment can be deployed on the site. In this case, the live account will be activated **1–5 business days after the successful tests**.

IMPORTANT: In addition to technical compliance, deployment is subject to signing the contract and paying the connection fee.

Based on the above, notify us of completing the development **5–8 business days** before your planned launch date for the bank card payment, in order to ensure it can be met.

1.5 Deployment steps involving the merchant

The development is carried out using a test system (sandbox). The live system will be deployed after successful SimplePay tests using sandbox. The merchant identification data necessary to initiate the transactions are identical in the live and the sandbox systems. Thanks to this, after the live system has been activated, changing a single parameter (the URL) controls which system to direct a transaction to.

There is no connection between the test system and the live system; therefore, your own settings affecting payments must be carried out in the live system, too. The most important of these is the IPN URL if your merchants have a test and live system with a different route (see IPN description later in this documentation).

You will have no further tasks if you have integrated the sample code supplied for the development into your own system.

1.6 The process of the payment transactions

1.6.1 Bank card payment

- a. On the merchant's website, the customer selects the items to be purchased, after which the total sum to be paid is calculated.
- b. The merchant transfers the transaction data to SimplePay via the API (**start**). At this point, the payment transaction is created in the SimplePay system. The system sends back a URL as a response to the data received. The merchant needs to redirect the customer to this URL in the browser.
- c. At the URL provided, the customer lands on the SimplePay payment page where the transaction data provided earlier are displayed. The customer can provide their card details on the payment page. If they already have a card registered in the Simple application, it can be selected to complete the payment, too.
- d. After the card details have been provided the payment is authorised by the bank (authorisation).

- e. Following the authorisation, the customer is redirected to the merchant's website (**back**). Here it is necessary to inform the customer about the result of the authorisation based on the returned data.
- f. Next, the fraud monitoring and prevention process is carried out in the background. If the system detects no issues, it sends feedback to the website in the background (**ipn**). This concludes the payment transaction. After the IPN message is received, the merchant can complete the order.

1.6.2 Transfer

- a. On the merchant's website, the customer selects the items to be purchased, after which the total sum to be paid is calculated.
- b. The merchant transfers the transaction data to SimplePay via the API (**start**). At this point, the payment transaction is created in the SimplePay system. The system sends back a URL as a response to the data received. The merchant needs to direct the customer to this URL in the browser
- c. At the URL provided the customer lands on SimplePay's bank transfer information page. The data necessary for the transfer are provided here, such as the bank account number and the required message.
- d. The customer may optionally return to the merchant's website (**back**) where the customer must be informed about the result of the transaction based on the returned data.
- e. As soon as the transferred amount gets credited to SimplePay's account, the system sends a notification to the website in the background (**ipn**). This concludes the payment transaction. After the IPN message is received, the merchant can complete the order.

1.6.3 Pay with Simple account, Apple Pay and Google Pay wallets or qvik QR code (at start endpoint)

The steps of the payment process fully identical with those described in the [Bank Card Payment](#) section. In addition, these payment methods can also be selected on the payment platform.

1.6.4 qvik (at starteam endpoint)

1. On the merchant's website, the customer selects the items to be purchased, after which the total sum to be paid is calculated.
2. The merchant transfers the transaction data to SimplePay via the API (**starteam**). At this point, the payment transaction is created in the SimplePay system. In response to the data it has received, the system sends a QR code and/or Deeplink and/or NFC. This is what the merchant can show in its own application in accordance with its logic.
3. Payment takes place in the buyer's mobile bank application.
4. The customer may optionally return to the merchant's website (**back**) where the customer must be informed about the result of the transaction based on the returned data.
5. As soon as the transferred amount gets credited to SimplePay's account, the system sends a notification to the website in the background (**ipn**). This concludes the

payment transaction. After the IPN message is received, the merchant can complete the order.

1.6.5 SZÉP card payment

The steps of the payment process fully identical with those described in the [Bank Card Payment](#) section, providing that only the data of a SZÉP Card issued by OTP can be entered through the payment platform.

1.7 SANDBOX and the live payment system

SimplePay is made up of two payment systems that are completely separated from each other. One of the systems can be used for transaction tests during the development, while the other for the live transactions. **The two systems are not connected to each other in any way.**

Because of the above, consideration must be given to the fact that **every setting** initiated in one or in the other system **will only be valid in that particular system**. If you intend to apply it in the other system as well, you must set it in that system, too.

The merchant's account will be activated in the live system after the development and the SimplePay-side tests have successfully been carried out using the sandbox system. If you attempt to initiate a transaction in the live system before that, you will receive an error message.

The below fundamental differences exist between the two systems:

Sandbox system

It can **only** be used to initiate **test transactions** to test the technical connection to the system. Only the test cards found on the payment page (OTP SZÉP card, Google Pay and Apple Pay as well), and simulated qvick transfer can be used to initiate transactions.

Transaction IDs are currently 9 digit-long and currently start with the "5" value (5xxxxxxx); therefore, the merchant system must be able to store transaction IDs of at least 9 digits, or it is even better to calculate with 10 digits in the longer term.

Merchant admin interface: <https://sandbox.simplepay.hu/admin/>

API receiving transaction requests: <https://sandbox.simplepay.hu/payment/v2/>

The designation of the actual service must be added to the above transaction URL. For example, in case of the "start" function, the URL is <https://sandbox.simplepay.hu/payment/v2/start>.

Because the sandbox only simulates transactions, **it cannot be used to test the following functions:**

- to charge a real bank card

- services on the payment page that are based on a Simple account
- services on the payment page that are based on a Simple application
- financial processes and settlements
- transactional analytics generation

Live system

It can **only** be used to initiate **transactions involving real cash flow**.

It can only be used to initiate transactions with valid bank cards issued by the bank, OTP SZÉP cards, Simple, Google Pay and Apple Pay wallets, as well as conventional and qvik transfers.

Transaction IDs are currently 9 digit-long and currently start with the "7" value (7xxxxxxx); therefore, the merchant system must be able to store transaction IDs of at least 9 digits, or it is even better to calculate with 10 digits in the longer term.

Merchant admin interface: <https://admin.simplepay.hu/admin/>

API receiving transaction requests: <https://secure.simplepay.hu/payment/v2>

The designation of the actual service must be added to the above transaction URL. For example, in case of the "start" function, the URL is

<https://secure.simplepay.hu/payment/v2/start>.

In the followings, every example will be shown using the sandbox system.

1.8 Downloadable resources

The resources necessary for the development can be downloaded from the merchant administration interfaces of both the sandbox and the live system.

In both systems the following content can be found in the left side menu, under "**Downloads**"

- technical documentation
- sample code
- logo package
- financial documentation, merchant recommendations
- SimplePay error codes
- SSL certificates

2 Transaction and its statuses

The **payment transaction** must be differentiated from the complete merchant-side **purchase process** because the former is only a part of the latter. The purchase process includes selecting the products, compiling the customer's cart, and optionally requesting the customer data necessary for the completion.

Once all of the data necessary for the purchase are available, the SimplePay payment can be initiated. Depending on its outcome, the order can be fulfilled and the purchase process can be completed.

In terms of the purchase process of the merchant, what is of primary importance is the successful/unsuccessful status of the payment; however, it is far more complex than this, and the payment has more statuses.

The following statuses can be queried with a “**query**” API request at any time. We will elaborate on the details of the query request later on, together with the other steps of the process.

The SimplePay payment process has the following statuses.

Status	Event
INIT	The transaction has been created in the SimplePay system
TIMEOUT	Timeout in INIT status
CANCELLED	Payment cancelled on the payment page, or the customer leaves the payment page, or closes the browser.
NOTAUTHORISED	Authorisation failed
INPAYMENT	Ongoing payment, after the “Pay” button is pushed
INFRAUD	Ongoing examination, while the fraud detection is running
AUTHORISED	Successful authorisation after the card details have been provided
REVERSED	Blocked amount reversed (two-step)
FINISHED	Successful, completed transaction

In addition to the transaction statuses, four events are possible when redirecting from the payment page to the merchant system. These will be discussed later in the topic of “**back**” redirection. **These events** may be the results of more complex processes and **may not necessarily match the current transaction status**.

Event	Event
SUCCESS	Successful card authorisation
FAIL	Card authorisation failed, or 3DS check was unsuccessful
CANCEL	The customer cancels their payment on the payment page
TIMEOUT	Authorisation failed

3 Implementation

In brief, payment is carried out in the following way. By compiling the purchase data, the merchant system creates a transaction in the SimplePay system. This can be achieved with a “**start**” request. In the response to the request, it receives a URL to which it needs to redirect the customer. If this redirection is complete, the customer lands on the payment page.

They provide the necessary data on the payment page, or if a card is already registered in the Simple application, it can be selected on the payment page, too.

They initiate the payment with the method selected, following which they are redirected back to the merchant website where the customer receives feedback on the result of the authorisation (**back**).

In the meantime, the fraud filtering (fraud monitoring) process of SimplePay is completed and it notifies the merchant about the success of the transaction (**ipn**) in a POST message in the background (i.e. not in the browser).

The merchant can complete the order after the “**IPN**” message was received and properly responded to.

In addition to the transaction's basic communication, optionally you can develop refund (**refund**), or, in case of two-step transactions, finish (**finish**), or the query of the transaction data (**query**).

IMPORTANT: Before you start the development, make a backup of your webstore and your data.

During development, pay special attention so as the bank card payment feature being developed or being tested cannot be used for live purchases and live payment transactions.

This is especially important for the developer to observe in cases where the SimplePay system is being integrated into a functioning webstore as a new payment method.

In cases where a separate development system is not an option, and the development is carried out in an operating webstore, **warn the customers** not to use bank card payment because **it is still under development/testing**.

3.1 General message format

The v2 API and the payment process are based on the following technical basis. This is the same in case of all requests and responses. Every description and sample code in the followings will be based on this.

Character encoding: **UTF-8**

Method of API messages: **POST**

Besides the API requests, the redirecting method in the browser is **GET**.

The **HMAC HASH** method of signatures (**signature**): **SHA384**

The API expects and sends signatures in the **header** in every communication.

The API expects and sends transaction data in the **http body** in every communication.

Data format: **JSON**

Content-Type in every case is **application/json**

SimplePay returns and transfers condensed (free of expendable white spaces) JSON in the responses and in the IPN messages. Brevity is not expected of the merchant systems. Every time value needs to be provided as a string according to the **ISO 8601** standard (2018-09-15T11:25:37+02:00) and the API returns the time values in this format, too.

Currency must be provided according to the **ISO 4217** standard, which may, at present, include: (HUF, EUR, USD) Countries must be provided according to the **ISO 3166-1 alpha-2** standard (HU, GB, DE etc.).

The languages must be provided according to the **ISO 639-1 alpha-2** standard, which may, at present, include: **AR, BG, CS, DE, EN, ES, FR, HR, HU, IT, PL, RO, RU, SK, TR, ZH**

Every request and response includes a field named "**salt**" that contains 32 random characters. This is to increase the variance of the message and, therefore, the security of the signature.

Amount is always a number greater than zero. It is integer when the currency is HUF, and two decimals may be used for EUR and USD. Period (.) is used as decimal separator. Thousands separator is not applicable.

In the event of failure of an API request, the array "errorCodes" is added to the response, which contains the error codes needed to identify the error.

The **transaction ID** generated by **SimplePay** is currently 9 digits. Therefore, the merchant's database field must be able to store a transaction identifier with **at least 9 digits**, and in the long run, a **10-digit** identifier shall be taken into consideration.

The API is under active development. The responses received by the merchant at the endpoints may be expanded with additional fields in the future due to the continuous expansion of functionality.

Therefore, the **merchant-side system must be able to process the contents of already known fields received from SimplePay, even if new elements appear in the message** that have not yet been implemented in the merchant's system.

In such cases these elements shall be omitted and ignored.

3.2 Validating messages

Messages sent in the body do not include the signature. The signature must be sent in the header of the message designated as "**Signature**".

The basis of the "Signature" is the message body (i.e. the whole JSON string). To calculate the signature, the body requires the SHA384 HMAC HASH value.

The "Signature" will be the Base64 encoded output of the received HASH.

The logic of the calculation described above can be summarised as the following (the functions and the syntax can differ in different programming languages)

```
signature = codeBase64(hmacWithSha384(merchantKey, message))
```

The transaction details below are used as an example.

```
{
  "salt": "f2dbffcef9a3ab94b618ddb59f0da637",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "1853354717597262343977",
  "currency": "HUF",
  "customerEmail": "sdk_test@simplepay.com",
  "language": "HU",
  "sdkVersion": "SimplePay_PHP_SDK_2.1.5_250731:819df13a2304433007f4e3159610f623",
  "methods": [
    "CARD"
  ],
  "total": "25",
  "timeout": "2025-10-06T07:00:34+02:00",
  "url": "https://sdk.simplepay.hu/back.php"
}
```

With the help of the sample JSON string below (**message**) and the sample merchant key (**merchantKey**), you can achieve the following result (**signature**) in any programming language.

The account-specific unique value for the **merchantKey** variable can be found in the merchant control panel among the technical settings of the merchant account under the name "SECRET_KEY".

In the case of tools (e.g. postman.com) used for API testing with the following data, pay attention to the following:

- URL: <https://sandbox.simplepay.hu/payment/v2/start>
- method: POST
- use the following JSON string without enters and other format operations in the call body
- make sure to copy the JSON in a single line without line breaks
- the header contains the "Signature" element with the following value
- the header's "Content-Type" value should be "application/json"
- the transaction has already been paid through the sandbox system; therefore, an error message will appear in the response

body

```
{"salt":"f2dbffcef9a3ab94b618ddb59f0da637","merchant":"PUBLICTESTHUF","orderRef":"1853354717597262343977","currency":"HUF","customerEmail":"sdk_test@simplepay.com","language":"HU","sdkVersion":"SimplePay_PHP_SDK_2.1.5_250731:819df13a2304433007f4e3159610f623","methods":["CARD"],"total":"25","timeout":"2025-10-06T07:00:34+02:00","url":"https://sdk.simplepay.hu/back.php"}
```

merchantKey

```
FxDa5w314kL1Nseq2sKuVwaqZshZT5d6
```

Signature

```
ioi43JEN1utnzPTdJGjuU7we8zgyWh7s0NILYgN0PYLej30nyjFohbI3YZ747J0v
```

Every response to an API request contains a signature, too, which is needed to check on the merchant's side to validate the response.

3.3 start – creating a bank card payment transaction

The API expects the start request at the following URL:

<https://sandbox.simplepay.hu/payment/v2/start>

The **start** is the beginning of the payment transaction. At this point, the transaction details, collected in the merchant system, are transferred to SimplePay.

Among the data can be found the global data of the order, the contents of the cart, invoicing details, delivery information, SimplePay-specific data, URLs etc.

The transaction data must be sent to the above URL in a JSON string described in the "**General message format**" section, with POST method. The following data are required to initiate a transaction.

salt: 32-character long random string,

merchant: the unique ID of the merchant account in the SimplePay system.

orderRef: unique transaction identifier in the merchant system

currency: the currency of the transaction

customerEmail: email address of the customer

language: the language of the payment page

sdkVersion: the version number of the payment in the merchant system

methods: array of the payment method

total: the amount of the transaction

timeout: the time for which the transaction is valid, i.e. the time to initiate the payment

url: the redirect URL where the merchant wants to redirect the customer after the payment

invoice: array for the invoicing data

```
{
  "salt": "b9955d850f81d4dd339a689eaf17cd9a",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "1853354717597270925650",
  "currency": "HUF",
  "customerEmail": "sdk_test@simplepay.com",
  "language": "HU",
  "sdkVersion": "SimplePay_PHP_SDK_2.1.5_250731:819df13a2304433007f4e3159610f623",
  "methods": [
    "CARD"
  ],
  "total": "25",
  "timeout": "2025-10-06T07:14:52+02:00",
  "url": "https://\\/\\/sdk.simplepay.hu\\/back.php",
  "invoice": {
    "name": "SimplePay V2 Tester",
  }
}
```

```
"country": "hu",  
"state": "Budapest",  
"city": "Budapest",  
"zip": "1111",  
"address": "Address 1",  
"address2": "Address 2",  
"phone": "06201234567"  
}  
}
```

The signature for the JSON string containing the data must be calculated as described in the “**Validating messages**” section. The following Base64 encoded HASH belongs to the above data:

```
X1L6FZpk5UwYqjT0PpKZRzYJmuq9Ps0DjU3k5ff-7QwSzC8u-Bj9VWtLfzzF3U4P
```

The calculated signature must be added to the heading of the message as the value of the “Signature”.

```
Content-type: application/json  
Signature: X1L6FZpk5UwYqjT0PpKZRzYJmuq9Ps0DjU3k5ff-7QwSzC8u-Bj9VWtLfzzF3U4P
```

After the body and header of the message have been created according to the above sample, the transaction can be initiated with a POST method towards the “**start**” interface of SimplePay:

<https://sandbox.simplepay.hu/payment/v2/start>

Every endpoint request is based on the same principles, i.e. the compilation of the JSON string containing the data of the message and the calculation of the associated Signature. The Signature is calculated and sent the same way in every case; therefore, this documentation will not discuss it in the case of further API requests.

3.4 start – creating a transfer transaction (Instant Transfer)

The following payment methods can, at present, be applied in the SimplePay system: Simple account, qvik, Google Pay, Apple Pay, OTP SZÉP card payment and bank transfer.

The only difference between the various types of payment is the methods value in the start call.

In the case of a wire transfer, this is “WIRE” value as follows.

```
"methods": [ "WIRE" ]
```

For an Instant Transfer, the buyer is transferred to the information page of transfer data of the URL returned in the “start” call.

The buyer authorises the transaction on the online bank platform of their own bank or in the bank’s mobile application. After execution of the transfer, the amount is credited to the bank account of SimplePay Plc. The SimplePay system notifies the merchant system in an “ipn” message also used for bank card payments, detailed below. Use of the “query” for querying the transaction status is an alternative to the above, also detailed below.

Use of either the “ipn” or “query” is the more appropriate technical solution for the merchant system; the **FINISHED** status of the transaction is the point at which the paid order can be transacted.

The refund also functions in case of an instant transfer, i.e. if the merchant intends to re-use the transferred amount or a part thereof, the “refund” API call can also be used for this purpose.

For an instant transfer, the following bank card payment processes do not function:

- two-step payment
- bank card registration
- SCA/3DS

3.4.1 Transfer timeout settings

Currently no “timeout” variable needs to be sent in the case of a transfer in the “start” call. If it is sent, its value will be overwritten by the value of the field “**Waiting for transfer in banking days**” on the “**Technical data**” tab in the merchant’s admin system. This field is freely editable according to the merchant’s need.

When the “timeout” variable is not sent in the start call, the value set in banking days set in the merchant account will still be considered at the launch of the transaction. If, however, the merchant system sends the “timeout” value, the system will consider the value received by the API instead of the time set at account level in banking days.

Since the timeout may even be 180 days in the “start” call, the most flexible solution is if the date of expiry is sent in each case, even if it is several days or weeks.

3.5 Transaction start using strong customer authentication (SCA, 3DS)

The strong customer authorisation launched in 2020 requires the transmission of extended transaction data. In the event of the following information being sent in the “start” communication, the merchant complies with these rules.

See Appendix 2 for more information on strong customer authentication.

customerEmail: email address of the customer

invoice (billing information) array, which contains the following items

name billing name

country name of the country given in text

state: name of the county given in text

city: city

zip: postcode

address: address

threeDSReqAuthMethod: (optional) The way the buyer registers in the merchant system

possible values:

01: guest

02: registered with the merchant

05: registered with a third-party ID (Google, Facebook, account etc.)

address2: (optional) second headline

phone: (optional) phone number

SimplePay passes on the above information when a payment is initiated. The bank also takes these into consideration when authorising a transaction, which is why sending these and the authenticity of the data are crucial to the success of the transaction.

If the email address of the buyer is not known in the merchant system, then the variable "maySelectEmail" will allow the customer to provide this information on the payment page.

```
"maySelectEmail":true,
```

If the billing information of the buyer is not known in the merchant system, then the variable "maySelectInvoice" will allow the customer to provide this information on the payment page.

```
"maySelectInvoice":true,
```

3.6 3DS challenge

During the 3DS challenge, the card issuing bank wishes to interactively identify the cardholder during the given transaction.

During the live operation of the process, the bank sends a single use identifier in the mobile application or in a text message to the cardholder, which they are required to enter on the bank platform. If the identification code matches, the payment process is continued.

The 3DS challenge process is simulated on the sandbox payment page. To ensure that there is a 3DS challenge during the transaction, during test payment the card designated for the 3DS process must be selected from the drop-down list of cards in the sandbox.

In this case, payment is not performed immediately, but we are first redirected to the simulated bank code request page. During **live** operation, here (or on an equivalent card

issuing bank platform) it is possible to enter the code received in the mobile application or in the text message.

In case of a **sandbox** 3DS Challenge simulation, the system **DOES NOT send a one-time**, but two constant values may be provided:

- **1234** for the successful test process
- **1111** for the unsuccessful test process

3.7 Further variables

Additional elements can be added to the "start" request. Sending these is optional.

items: list of items to be paid. The system calculates the amount to be paid by adding up the items.

- **ref:** product identifier in the merchant system
- **title:** product name
- **desc:** product description
- **amount:** ordered quantity, integer, required
- **price:** unit price, greater than zero, compulsory
- **tax:** VAT percentage, provided as integer, e.g. 27. If sent as a value of zero (0), there is no VAT calculation, that is, in this case, it is a gross price. If the field is not sent, it is set to the default value of 0.

If both "**total**" and "**items**" are sent in the same transaction, "**total**" will be ignored.

```
"items": [
  {
    "ref": "Product ID 2",
    "title": "Product name 2",
    "desc": "Product description 2",
    "amount": "2",
    "price": "5",
    "tax": "0"
  }
],
```

shippingCost: the amount of the delivery cost specified in the currency setup in the account, which will be added to the amount payable. In the case of HUF 12, you can accomplish this as discussed below.

```
"shippingCost": 12,
```

discount: the amount of the discount specified in the currency setup in the account, which will be deducted from the amount payable. In the case of HUF 5, you can accomplish this as discussed below.

```
"discount": "5",
```

customer: name of the customer, if different from the value "invoice"/"name"

```
"customer": "V2 Test User",
```

urls: if the merchant system processes the authorisation result differentiated depending on the event, rather than using a common URL, then you can specify a separate URL for each event.

Possible events:

- **success:** successful authorisation
- **fail:** authorisation failed
- **cancel:** cancelled transaction
- **timeout:** transaction has timed out

```
"urls": {  
  "success": "https://sdk.simplepay.hu/success.php",  
  "fail": "https://sdk.simplepay.hu/fail.php",  
  "cancel": "https://sdk.simplepay.hu/cancel.php",  
  "timeout": "https://sdk.simplepay.hu/timeout.php"  
},
```

If both "url" and "urls" are sent in the same transaction, "url" will be ignored.

twoStep: it can be sent in case of two-step transactions. If it has false as value, the transaction will not wait for a "finish" request, but will trigger an immediate full charge. This can be used to initiate an immediate charge even on two-step accounts.

```
"twoStep": false,
```

delivery: delivery information.

```
"delivery": {  
  "name": "SimplePay V2 Tester",  
  "company": "",  
  "country": "hu",  
  "state": "Budapest",  
  "city": "Budapest",  
  "zip": "1111",  
  "address": "Delivery address",  
  "address2": "",  
  "phone": "06203164978"  
},
```

maySelectEmail: if the email address of the buyer is not known in the merchant system, then the variable "maySelectEmail" will allow the customer to provide this information on the payment page.

```
"maySelectEmail":true,
```

maySelectInvoice: if the billing information of the buyer is not known in the merchant system, then the variable "**maySelectInvoice**" will allow the customer to provide this information on the payment page.

```
"maySelectInvoice":true,
```

maySelectDelivery: if the customer's delivery information is not available for the merchant, the SimplePay payment page can also request it. To do this, all countries must be listed where the merchant delivers, if this variable is sent. On the payment page, the customer can only choose from the specified countries.

```
"maySelectDelivery":[
  "HU",
  "AT",
  "DE"
],
```

3.8 start – response

The SimplePay system gives a synchronous response to the start request. The response header also contains a Signature value. It is recommended to check this on the merchant's side in every case. The response body is a JSON string containing the following:

salt: 32-character long random string

merchant: the SimplePay account of the merchant where the transaction was created

orderRef: the orderRef value provided in the start request

currency: the currency value provided in the start request

transactionId: the identifier of the SimplePay transaction created

timeout: the time limit until the payment can be initiated

total: the amount of the transaction

paymentUrl: the URL where the merchant system needs to redirect the customer to perform the payment.

```
{
  "salt": "KAC6ZRUacmQit98nFKOpjXgkwdC0Grz1",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "101010515680292482600",
  "currency": "HUF",
  "transactionId": 99844942,
  "timeout": "2019-09-11T21:14:08+02:00",
  "total": 25.0,
  "paymentUrl": "https://sandbox.simplepay.hu/pay/pay/pspHU/8f4oKRec5R1B696x1xb0cj1jRhhABA2pwSLQDPW60z"
```

```
oGSDWzDU"  
}
```

The value of **paymentUrl** can be used in more than one way to redirect the customer from the merchant site to the payment page. The simplest of these is to display a button. If the customer clicks on the button, the redirection to the payment page takes place.

```
<form action="https://sandbox.simplepay.hu/pay/pay/pspHU/8f4oKRec5R1B696x1xb0cj1jRhABA2pWzDU" method="POST" id="SimplePayForm" accept-charset="UTF-8">  
    <button type="submit">Start SimplePay Payment</button>  
</form>
```

3.9 Failed API requests

In the event of the API is unable to perform the request for some reason, the array **"errorCodes"** will appear in the response, containing the error codes.

```
{  
  "errorCodes": [  
    5321  
  ]  
}
```

The array **"errorCodes"** specifies the current error code in the same way on all API endpoints, so the documentation will not discuss it further at the implementation of each function.

3.10 What needs to be done at the merchant's site before redirecting to the payment page

The **SimplePay logo must be displayed** on the merchant's website **before the payment can be initiated**. The placement of the logo is discussed in Chapter **"Logos and information pages"** of this documentation.

In addition, the Data Transfer Declaration must be displayed for and accepted by the customer before the payment can be initiated. The content and display criteria for this declaration are described in Chapter **"Data Transfer Declaration"** of this documentation.

3.11 SimplePay payment page

Up to this point the payment transaction has been created in the SimplePay system with the **"start"** function. The merchant system has received a URL from the SimplePay API where the customer must be redirected. At that URL, the SimplePay payment page is loaded showing the data of the newly created transaction.

The payment page displays the payment transaction's details provided earlier, the merchant to whom the customer will pay, and all the necessary information. The panel displaying the customer data and the product details can be switched on and off in the merchant account among the technical settings.

In the sandbox system the card number field is actually a drop-down menu where you can select the test cards to simulate successful and unsuccessful payments. The outcome of the payment depends on the card selected.

In the live system, here can be entered the card number.

The transaction is initiated by clicking on "Payment with bank card" (or, depending on the payment method: "I would like to receive a QR code" or "Card registration"). Then the card authorisation takes place, after which the customer is redirected to the website of the merchant.

On the live payment page, bank cards previously registered in the Simple app can also be used, and new ones can be registered, and there are other features available here too.

The functions that require real banking information or such external systems that are available for SimplePay during live operation only, **are not available on the sandbox payment page.**

Such are

- cards registered in the Simple app
- the Simple mobile app
- 3DS SMS authorisation codes (replaced with fixed codes)
- Installment

The above functions are beyond the control of the merchant system and therefore do not require development.

The following documentation fully describes the services offered by the payment page:

<https://simplepartner.hu/download.php?target=paymentflowhu>

3.12 back – information on the merchant website

In the browser the customer is redirected from the payment page to the merchant website. The redirecting is performed to the URL defined in the start request. Thus, "back" is not the API's function but a part of the merchant system, since **that part** of the transaction, taken place in the browser, **which the customer can see ends on the merchant website.**

This request takes place in the browser with the GET method. The SimplePay system appends two variables to the URL specified upon initiation.

"r" variable

The “r” variable (response) contains the result and the details of the payment. The content is a Base64 encoded JSON string.

“s” variable

The “s” variable (signature) is the signature of the JSON string. Validating the signature is recommended when the request is made in the browser.

In case of the sample transaction shown earlier, the following data are added when the customer is redirected to the URL at the merchant website. The URL was provided earlier during the “start” communication in the “url” variable or as one of the values of “urls”.

```
https://sdk.simplepay.hu/back.php?r=eyJyIjowLCJ0Ijo5OTg0NDk0MiwZSI6I1NVQ0NFU1MiLCJtIjoiUFVCTE1DVEVTVhVVRiIsIm8iOiIxMDEwMTA1MTU2ODAyOTI0ODI2MDAifQ%3D%3D&s=E1%2Fnxvex9TjgjuORI63gEu5I5miGo4CSAD51mEpKIxp7WuVRq6bBeh1QdyEvVGSsi
```

The content of the “r” variable after the Base64 decoding is the following JSON string:

```
{
  "r": 0,
  "t": 99844942,
  "e": "SUCCESS",
  "m": "PUBLICTESTHUF",
  "o": "101010515680292482600"
}
```

The elements of JSON are the following:

- r:** response code (response code)
- t:** SimplePay identifier of the transaction (transaction id)
- e:** event (event)
- m:** identifier of the merchant account (merchant)
- o:** transaction identifier of the merchant (order id)

The payment may return to the merchant's site with four kinds of events. These may not be the same as the current status of the transaction:

- success:** successful authorisation
- fail:** authorisation failed
- timeout:** timeout, when the payment has not been initiated until the set time limit
- cancel:** cancelled payment, browser closed, leaving the payment page

In those cases where the payment has failed, the response code contains the error code of the problem.

3.13 Information depending on the result of the transaction

Adequate information must be given to the customer in the browser about the result of the transaction.

3.13.1 Cancelled payment

If the customer would like to return from the payment page for any reason, they can push the **“Back”** button. Pushing the button will take them back from the SimplePay payment page to the merchant page that was provided for this purpose when the transaction was initiated. The redirection is performed to the URL provided either in the **“url”** variable, or in the **“cancel”** element of the **“urls”** array.

With regard to the information displayed to the user it is very important to consider that no payment has occurred in this case, as the payment process was cancelled before filling in the card details and initiating the payment. **Accordingly, the customer cannot be informed of unsuccessful payment, nor can the SimplePay transaction ID presented here.**

Information example for a case of cancelled payment

The customer must be adequately informed of why they were redirected from the payment page to the merchant website. For example, using one of the following options.

You cancelled the payment

or

Cancelled payment

3.13.2 Timeout

In case of a timeout, the customer landed on the SimplePay payment page after initiating a transaction but provided no card details and initiated no payment by the deadline specified in the **“start”** request.

The time available for payment (the payment deadline) is provided by the merchant system as the value of the **“timeout”** variable when the transaction is created.

In such cases, in case of a timeout, the payment page automatically redirects the customer to the merchant webpage that was provided when the transaction was started. The redirection is performed to the URL provided either in the **“url”** variable, or in the **“timeout”** element of the **“urls”** array.

With regard to the information displayed to the user it is very important to consider that no payment has occurred in this case, since the time allowed for initiating the transaction was exceeded before filling in the card details and initiating the payment. **Accordingly, the customer cannot be informed of unsuccessful payment, nor can the SimplePay transaction ID presented here.**

Information example for a case of timeout

The customer must be adequately informed of why they were redirected from the payment page to the merchant website. For example, using one of the following options.

You exceeded the maximum time available to start the transaction.

or

Timeout

3.13.3 Payment failed

In this case, the customer pushes the “Payment with bank card” (or, depending on the payment method: “I would like to receive a QR code” or “Card registration”) button after providing their card details, thus initiating the transaction, which is then rejected by the issuing bank; therefore, the payment is unsuccessful. After an unsuccessful payment, the merchant must inform the customer.

After entering bank card data, the primary reason for the failure may be that the customer provided incorrect details on the payment page, e.g. mistyped the CVC/CVV value or the 3DS authentication code.

Even when correct details were provided, there might be no sufficient amount available on the card for the transaction. Furthermore, it may be that the customer has reached their set daily limit, or the customer’s card might no longer exist, or might be expired or blocked.

These issues cannot be handled by the merchant because the card-issuing bank only provides the card holder with card and payment information.

Only the owner of the card can solve these problems or initiate a solution at the bank issuing the card or the account servicing bank, e.g. they can raise the daily limit on the card via their internet bank and attempt the payment again.

If the customer is not left helpless about what may have happened, but attempts to solve the problem as a result of having adequate information received, the number of failed transactions will be much lower.

Because of the above it is extremely important that the website provides adequate information in such cases.

The information needs to include two essential elements:

- the SimplePay identifier of the transaction, based on which our customer service can also provide help with questions related to a specific transaction
- clear information that directs the customer towards the solution of the issue

To avoid misconduct, the cause of failure (e.g. limit overrun, blocked card) must not be specified in the information provided for the customer.

Information example for a case of unsuccessful payment

Failed transaction.

SimplePay transaction ID: 7xxxxxxx

Please check if the details provided during the transaction are correct. If all of the details were provided correctly, please contact the bank that issued your card in order to investigate the cause of the rejection.

3.13.4 Successful payment

If successfully completed (when choosing to pay by [bank card](#) or [SZÉP card](#)), authorisation takes place after the card details have been entered, and the customer is redirected to the merchant's website or application.

In this case, displaying the SimplePay transaction identifier is sufficient information.

Information example for the case of successful payment

Successful transaction.

SimplePay transaction ID: 7xxxxxxx

3.14 ipn – notification on transaction statuses and executability

IPN is sent from the following SimplePay IP addresses:

80.249.162.112/28

84.2.229.128/27

195.228.18.224/29

During Instant Payment Notification (**ipn**), SimplePay notifies the merchant system of the finished status of the payment transaction. The SimplePay server sends the message to the merchant by POST method.

Earlier the system only sent notification of payment, refunds and pre-authorisation in relation to two-step payments, i.e. of cases in which the switch of status can also be interpreted financially.

The SimplePay system can also send an IPN message on **unsuccessful finished statuses**. This function is **optional**, requiring a merchant setting. In relation to merchant systems, where the function was previously implemented for successful statuses, **mandatory changes are not required**.

Settings may be performed on the merchant admin platform, on the "System notifications" panel of the "Technical data" page.

The IPN message applies in relation to the following finished statuses

Status	Event
FINISHED	Closing of transaction in case of successful charge. This is how the merchant system is notified that it can execute the order
AUTHORISED	In relation to two-step payment, upon inclusion of the amount, or pre-authorisation
NOTAUTHORISED	If transaction authorisation has failed
REVERSED	In relation to two-step payment, upon release of the amount
CANCELLED	Upon interrupted payment
TIMEOUT	In case of a timeout

The merchant must receive the IPN message **and properly respond to it**. Based on the response, the SimplePay system will be notified that the **merchant** has received the notification, that is, the merchant was **notified of changes to the transaction status**.

In order to receive an IPN message, you must determine the URL where the merchant system expects the message. The URL is an address available online where the merchant receives the IPN message sent by the SimplePay system.

The IPN URL can be set on the merchant control panel.

<https://sandbox.simplepay.hu/admin/>

The address can be set under "**Technical data**". The IPN must be set for each merchant account. If a merchant uses more than one account on a domain (e.g. payments are allowed both in HUF and EUR) then the URL must in every case (for each account) be provided.

With regard to the IPN URL, the following must be considered

- they must be publicly accessible
- there should be no .htpasswd, or any protection
- redirection rules, SSL settings must not make it inaccessible

If the receipt of the IPN notification failed or the response is not adequate at the URL provided, SimplePay will regard it as if the merchant had **NOT** been notified of the status change. In this case, the system automatically resends the message.

The system will inform the merchant about the first unsuccessful transmission via email automatically.

Unsuccessful receipt may occur because the merchant system could not be reached with the IPN notification (status code other than HTTP 200) or the merchant failed to give the proper response.

The length of the IPN timeout is 20 seconds. This is exceeded when the merchant does not respond to the IPN message within this time period following the reception of the notification.

If for any reason the **first IPN** is **unsuccessful**, it is **sent again after the first attempt** at the following intervals:

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

Subsequently the server stops sending out the IPN message. However, the merchant can initiate sending manually from the admin interface at any time.

The following example shows the content of the IPN notification sent by the system. The Signature variable can be found in the message header, the message received must be validated based on its value.

```
{
  "salt": "223G0018VAqdLhQYbJz73adT36YzLtak",
  "orderRef": "101010515680292482600",
  "method": "CARD",
  "merchant": "PUBLICTESTHUF",
  "finishDate": "2019-09-09T14:46:18+0200",
  "paymentDate": "2019-09-09T14:41:13+0200",
  "transactionId": "99844942",
  "status": "FINISHED"
}
```

The above data must be returned in the response, complemented with the time of reception of the IPN notification. The time must be stored in the **"receiveDate"** field.

The Signature variable must also be provided here in the response header, the way it was described before. The value of the Signature must be calculated for the response JSON string containing the complemented **"receiveDate"** value.

```
{
  "salt": "223G0018VAqdLhQYbJz73adT36YzLtak",
  "orderRef": "101010515680292482600",
  "method": "CARD",
  "merchant": "PUBLICTESTHUF",
  "finishDate": "2019-09-09T14:46:18+0200",
  "paymentDate": "2019-09-09T14:41:13+0200",
  "transactionId": "99844942",
  "status": "FINISHED",
  "receiveDate": "2019-09-09T14:46:20+0200"
}
```

3.15 finish – finishing two-step transactions

The API expects the finish request at the following URL:

<https://sandbox.simplepay.hu/payment/v2/finish>

The transactions shown earlier included **one-step** payments. In case of one-step payments, the authorisation starts when the customer provides their bank card details on the payment page. When it is successful (the data that are provided are correct, the card holds sufficient funds to perform the payment transaction), **it is charged** immediately.

In case of **two-step** transactions, **after the authorisation** the amount entered is only blocked, but **it is not charged**. This is the first step.

To launch the charging, a separate finishing request is necessary that can be initiated with the finish function. This is the second step.

Two-step transactions, in which the amount was only blocked (i.e. only the first step occurred), can be closed with the **finish** request. The merchant has 21 calendar days for the closing. If no charging or releasing occurs until then, the SimplePay system will automatically release the amount that was blocked on the customer's card.

This payment process is useful in cases where it is necessary to verify that the customer's card can be charged with the specific amount (first step) but the merchant is required to take more steps to fulfil the order, the result of which it does not yet see, and so it does not wish to charge immediately because of this.

Reasons for this may be, for example, checking the stocks, negotiating with the suppliers or the necessity to place merchant orders in other online systems etc.

The transaction can be closed with the finish request. The closing can be performed with the entire blocked amount or with a smaller amount, too, in case of partial performance. In cases where there is no performance, it can be closed with zero amount, too, i.e. in such cases the entire blocked amount will be released on the customer's card.

The two-step payment option can be switched on in the sandbox system by the SimplePay IT support. Once the service has been activated, the merchant can initiate both one-step and two-step transactions.

If the initiation takes place according to the start function described earlier, it will result in two-step transactions, i.e. after the authorisation the transaction will be waiting to be closed.

In case of two-step accounts, the "start" request can be supplemented with the "**twoStep**" parameter.

In a two-step account, if the twoStep value is **true** or no twoStep is forwarded, a two-step transaction will be initiated. If one is forwarded and its value is **false**, after a successful authorisation (first step) the amount will be charged (second step) automatically, without a "finish" request.

With the help of the twoStep variable, regular two-step transactions can be initiated in a merchant account in parallel with transactions where the amount is charged immediately.

```
{
  "salt": "67b253f7f7ee8c264f01b16e3bcb9611",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "101010515680496082852",
  "currency": "HUF",
  "customerEmail": "sdk_test@simplepay.com",
  "language": "HU",
  "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e",
  "methods": [
    "CARD"
  ],
  "total": "25",
  "twoStep": true,
  "timeout": "2019-09-12T00:53:28+00:00",
  "url": "https://sdk.simplepay.hu/back.php",
  "invoice": {
    "name": "SimplePay V2 Tester",
    "company": "",
    "country": "hu",
    "state": "Budapest",
    "city": "Budapest",
    "zip": "1111",
    "address": "Address 1",
    "address2": "",
    "phone": "06203164978"
  }
}
```

The transaction after a successful authorisation will remain in the **AUTHORISED** status. At this time, the amount to be paid is blocked on the customer's card but it is not charged yet. To initiate the charging the finish request must be launched with the following data.

The key data of the request are the **originalTotal** and **approveTotal** values.

originalTotal: the amount originally blocked during the transaction

approveTotal: the actual amount to be charged

The value of **originalTotal** can only be the exact amount blocked.

The **approveTotal** value can be provided in three ways

1. The total blocked amount. In this case, the amount of the originalTotal will be charged.
2. An amount that is lower than the original but higher than zero. In this case, the amount provided here will be charged, the amount in excess will be released on the customer's card.
3. Zero value. In this case, the total blocked amount will be released on the customer's card.

```
{
  "salt": "a182f12e696d483985133e299c245b83",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "101010515680496082852",
```

```

"originalTotal": "25",
"approveTotal": "15",
"currency": "HUF",
"sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
}

```

At least one of the **transactionId** and the **orderRef** values must be provided so that the transaction can be identified.

3.16 refund – managing refunds

The API expects the refund request at the following URL:

<https://sandbox.simplepay.hu/payment/v2/refund>

By applying the refund, the amount of the transactions charged earlier can be refunded either in part or in full. The amount must be greater than zero. The maximum value is the amount charged earlier.

If a part of the amount is refunded, further refunds can be initiated against the remaining amount of the original transaction until the amount of the partial refunds reaches the total amount originally charged. Every refund request runs as a separate transaction that is linked to the original transaction.

When initiating the request, the transaction identifier of the original charge, the amount to be refunded, and the currency must be provided along with the basic data, as follows. The transaction ID may be the merchant's transaction ID or the SimplePay transaction ID.

```

{
  "salt": "6a85ef475fa491618a94af9bb0b2065d",
  "orderRef": "101010515680496082852",
  "merchant": "PUBLICTESTHUF",
  "currency": "HUF",
  "refundTotal": 5,
  "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
}

```

Three key fields of the response given to the request:

- **refundTransactionId**: transaction identifier of the refund
- **refundTotal**: the refunded amount
- **remainingTotal**: the remaining amount of the original transaction after this refund (original amount charged minus all refunds up to now)

```

{
  "salt": "WZ7Ncc0qoDSMYG4twsme0dBs6PSnsj1Z",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "101010515680496082852",
  "currency": "HUF",
  "transactionId": 99826864,
  "refundTransactionId": 99826879,
  "refundTotal": 5.0,
}

```

```

    "remainingTotal":10.0
  }

```

3.17 query – querying transaction details

The query call is expected by the API at the following URL:

<https://sandbox.simplepay.hu/payment/v2/query>

With the help of the **query**, the details of a particular transaction can be queried from the SimplePay system. The API sends a synchronous response to requests sent with the POST method.

The query is based on the unique ID of the transaction. The unique merchant identifier (**orderRef**) or the SimplePay transaction identifier (**transactionId**) can be specified.

The simplest way is to initiate a query based on the SimplePay transaction identifier.

```

{
  "merchant": "PUBLICTESTHUF",
  "transactionIds": [
    "99325412"
  ],
  "salt": "c4de372a465b56aa0187b8482570a2cd",
  "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
}

```

Using the query request, the details of more transactions can be retrieved during one query. In this case, more transaction identifiers can be specified in the **transactionIds** list.

```

{
  "merchant": "PUBLICTESTHUF",
  "transactionIds": [
    "99325521",
    "99325516"
  ],
  "salt": "42c938bdaad569154753eb63697f9918",
  "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
}

```

The merchant identifier (**orderRef**) and the SimplePay identifier (**transactionId**) can be used together in the query. In this case, the system will find every transaction based on the two different sets of data. More identifiers can be specified of both types at the same time.

```

{
  "merchant": "PUBLICTESTHUF",
  "orderRefs": [
    "101010515383930534733"
  ],
  "transactionIds": [
    "99325521",
    "99325516"
  ],
}

```

```

    "salt": "d2a919eeb8746453777e790c9ab09377",
    "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
  }

```

In the response, the transactions list will contain the details of the queried transactions.

```

{
  "salt": "sNwsKj5sDsQUS1c4LqjliAH3unCG4Mt3",
  "merchant": "PUBLICTESTHUF",
  "transactions": [
    {
      "salt": "qnoc9Tsw1fCSA3ynsM2vCfPpeTcny1eJ",
      "merchant": "PUBLICTESTHUF",
      "orderRef": "101010515383930534733",
      "total": 25,
      "transactionId": 99325535,
      "status": "FINISHED",
      "resultCode": "OK",
      "remainingTotal": 0,
      "paymentDate": "2018-10-01T13:24:14+02:00",
      "finishDate": "2018-10-01T13:24:34+02:00",
      "method": "CARD"
    },
    {
      "salt": "5sOUArueyVvLHi7zELIkgyy6BczwWl5o",
      "merchant": "PUBLICTESTHUF",
      "orderRef": "101010515383924753263",
      "total": 25,
      "transactionId": 99325521,
      "status": "INIT",
      "remainingTotal": 0,
      "paymentDate": "2018-10-01T13:14:36+02:00",
      "method": "CARD"
    },
    {
      "salt": "KwrgINkzKpUzqOgFtwZaDnSTxZExQD7k",
      "merchant": "PUBLICTESTHUF",
      "orderRef": "101010515383923675972",
      "total": 25,
      "transactionId": 99325516,
      "status": "FINISHED",
      "resultCode": "OK",
      "remainingTotal": 0,
      "paymentDate": "2018-10-01T13:12:47+02:00",
      "finishDate": "2018-10-01T13:14:39+02:00",
      "method": "CARD"
    }
  ],
  "totalCount": 3
}

```

The number of transactions sent back in the result can be found in the **totalCount** value.

The query can be supplemented with the **"detailed"** parameter, and then the details of the transaction data will also be included in the response.

```

{
  "merchant": "PUBLICTESTHUF",
  "detailed": true,
  "transactionIds": [
    "99325516"
  ]
}

```

```

    ],
    "salt": "94fe1c712565697d732a7bce510b9146",
    "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
  }

```

Detailed response

```

{
  "salt": "099RQ1cf2K4nZaxWA50v6ySavi6DiwFa",
  "merchant": "PUBLICTESTHUF",
  "transactions": [
    {
      "salt": "FNoR545Lw2kUQdQOXXy7sJWPBGU4h5T9",
      "merchant": "PUBLICTESTHUF",
      "orderRef": "101010515383923675972",
      "currency": "HUF",
      "customer": "v2 START Tester",
      "customerEmail": "sdk_test@simplepay.com",
      "language": "HU",
      "twoStep": false,
      "total": 15.0,
      "shippingCost": 0.0,
      "discount": 0.0,
      "invoice": {
        "company": "",
        "country": "hu",
        "state": "Budapest",
        "city": "Budapest",
        "zip": "1111",
        "address": "Address 1",
        "address2": "",
        "phone": "06203164978",
        "lname": "SimplePay V2 Tester"
      },
      "delivery": {
        "company": "",
        "country": "hu",
        "state": "Budapest",
        "city": "Budapest",
        "zip": "1111",
        "address": "Address 1",
        "address2": "",
        "phone": "06203164978",
        "lname": "SimplePay V2 Tester"
      },
      "transactionId": 99325516,
      "status": "FINISHED",
      "resultCode": "OK",
      "remainingTotal": 0.0,
      "paymentDate": "2018-10-01T13:12:47+02:00",
      "finishDate": "2018-10-01T13:14:39+02:00",
      "method": "CARD"
    }
  ],
  "totalCount": 1
}

```

The query can be supplemented with the **"refunds"** parameter, and then the data of the refunds initiated on the transaction will be provided next to the transaction data.

```
{
  "merchant": "PUBLICTESTHUF",
  "refunds": true,
  "orderRefs": [
    "101010515384686499284"
  ],
  "transactionIds": [
    "99326020"
  ],
  "salt": "ac17ee8c6e33f89cf7505e40decd33ed",
  "sdkVersion": "SimplePayV2.1_Payment_PHP_SDK_2.0.7_190701:dd236896400d7463677a82a47f53e36e"
}
```

Refunds are in every case preformed as separate transactions. Refunds can be found in the **"refunds"** list within a particular transaction.

```
{
  "salt": "QxQmq0FkqV9khWU6SHJx1KmYyuN74x1E",
  "merchant": "PUBLICTESTHUF",
  "transactions": [
    {
      "salt": "kK1f2RZJdtn5wHi0GB0qfNKE8yFtGwNW",
      "merchant": "PUBLICTESTHUF",
      "orderRef": "101010515384686499284",
      "total": 15,
      "transactionId": 99326020,
      "status": "FINISHED",
      "resultCode": "OK",
      "refundStatus": "PARTIAL",
      "refunds": [
        {
          "transactionId": 99326030,
          "refundTotal": 5.0,
          "refundDate": "2018-10-02T10:29:43+02:00",
          "status": "FINISHED"
        }
      ],
      "remainingTotal": 10.0,
      "paymentDate": "2018-10-02T10:24:09+02:00",
      "finishDate": "2018-10-02T10:29:28+02:00",
      "method": "CARD"
    }
  ],
  "totalCount": 1
}
```

3.18 `transactioncancel` – cancellation of the transaction

Transactions created earlier on can be cancelled at the endpoint. The endpoint only accepts cancellation requests for transactions that are still at INIT status. It will be rejected by the endpoint in all cases where the transaction has moved on to the payment process. Cancellation only means moving the transactions that have been created into the CANCELLED end-status.

The `transactioncancel` call is expected by the API at the following URL:

<https://sandbox.simplepay.hu/payment/v2/transactioncancel>

Cancellation is doable on the basis of the SimplePay transaction ID (transactionId) merchant identifier (orderRef).

```
{
  "salt": "81475695dc068268cbb08aa42f5ebaa6",
  "merchant": "PUBLICTESTHUF",
  "transactionId": "502896703",
  "currency": "HUF",
  "sdkVersion": "SimplePay_PHP_SDK_2.1.0_200825:f66d0203f6e60cab4a901f45368c22fc"
}
```

In the synchronous response to the cancellation request the API returns its transaction ID and current status along with the general data.

```
{
  "merchant": "PUBLICTESTHUF",
  "transactionId": 102896703,
  "orderRef": "101010517218129214792",
  "status": "CANCELLED",
  "salt": "L8qfJm4J94g0TTI833kBz8f7uVnKj7BH"
}
```

3.19 qvik (AFR2, EAM) payments

The Magyar Nemzeti Bank's (MNB's) Instant Payment System (**AFR2**), or its payment system based on transfers using a uniform data input solution (**EAM**). Its solutions include **QR**, **NFC** and **deeplink**.

The payment method is accessible under the name **qvik**.

Transfers of up to HUF 20 million can be initiated in the system launched in September 2024. All Hungarian banks have joined the system.

In the qvik payments process using the unified data entry system (EAM), the SimplePay system displays a QR code as well on the payment page. By scanning with the application of any Hungarian bank, the buyer ends up at the authorisation of the qvik transfer. Having been approved, payment takes place instantaneously in accordance with the AFR2 rules.

When using the unified data entry system for payments, SimplePay can also provide the deep link directly. The merchant system can display this deep link packaged in a QR code on its website.

The buyer can then scan the code using their banking app as described above, and authorise the payment.

The merchant's system can use the deep link on a mobile device to navigate directly to the customer's banking app, where payment is made in the same way after the above authorisation.

Moreover, the deep link can also be sent wirelessly via NFC.

3.19.1 Merchant account general settings

Qvik transactions can be run on the same merchant accounts on which bank card transactions are run. The only technical prerequisite is enabling the function on the SimplePay page.

3.19.2 Limitations

qvik payment

- is only accessible in HUF
- can be used only for one-step payment
- qvik is a transfer-based solution where card storage makes no sense
- qvik transactions may have timeout values depending on the context of payment. They can be found at the merchant account's technical settings.
- qvik payment is applicable only on the new payment page

3.19.3 tracking the qvik payment process

The SimplePay transaction ID (**transactionId**) renders the transaction identifiable at every point throughout the whole process.

- It can be found in the synchronous responses to the API call at the **start** or the **starteam** endpoints.
- It can be found in the comment attached to the buyer's bank transfer. Consequently, it can be found when making the transfer and later on, in the monthly account statement.
- It can be found in the transaction analysis file provided for the partner by SimplePay

3.19.4 qvik – start payment functions

The API expects the **start** request at the following URL:

<https://sandbox.simplepay.hu/payment/v2/start>

If the qvik payment method is enabled on the merchant's account, this type of payment method can be requested in the "**methods**" field of the **start** call—that has been used so far—with the parameter "EAM". In this case, the QR code of the qvik payment will appear when the user lands on the payment page.

```
"methods": ["EAM"],
```

In this case, "methods" may have a variety of values. It may be both bank card and EAM—as determined by the merchant—as follows.

```
"methods":["CARD", "EAM"],
```

The EAM value indicating the qvik payment cannot be transferred together with the WIRE payment method—they are mutually exclusive.

Whenever payment has been initiated using only the CARD type, and qvik is also enabled in the merchant account, the QR code will also appear on the payment page. **That is, if you want to use this payment method as well besides your existing bank card implementation, no further development is required.**

Since the maximum value limit for qvik transactions is HUF 20 million at present, the system will reject any qvik transaction over this amount.

All other settings can be made, at the start endpoint, as described earlier for bank card payments.

3.19.5 qvik – deeplink generation without payment page – starteam

The normal payment page bank card payment can be supplemented, on the early discussed “**start**” endpoint, with the qvik payment method as well. In this case, this is an additional payment possibility on the payment page.

At the same time, the system also enables the merchant to directly generate the deeplink required for the qvik/EAM transaction, which it can use in its own business logic without payment page redirection. One example for this is when one wishes to send the QR code by email.

A deeplink can be generated with the help of a **starteam** endpoint.

The **starteam** call is expected by the API at the following URL:

<https://sandbox.simplepay.hu/payment/v2/starteam>

The possible parameters of such a call are, essentially, the same as those of a “start” call; therefore, only the qvik-specific features are discussed below.

A **starteam** endpoint is associated only with qvik/EAM payments; therefore, “**methods**” does not apply. It needs not be sent.

“**eamType**” is the only new variable, whose value determines what the deeplink generated can be used for.

The following (1–7) values are possible depending on the merchant's business processes and technical solution, and the method of use:

- 1: QR
- 2: Deeplink
- 3: NFC
- 4: QR+ Deeplink
- 5: QR+NFC
- 6: Deeplink +NFC
- 7: QR+DL+NFC

The synchronous response of the “**starteam**” endpoint is the same as that of the “**start**” endpoint; therefore, only the qvik-specific features are discussed below.

The starteam endpoint is associated only with qvik payments; therefore, “**paymentUrl**” is not applicable in the response: this endpoint does not return this.

“**eam**” is the only new variable. The generated deeplink is its value. It can be used in the way specified before in the “**eamType**” field.

3.19.6 qvik – refund

qvik refund can be initiated at the “refund” endpoint, as discussed above. The call is fully identical with the process discussed in relation to the card refund case.

3.19.7 qvik – transaction cancellation

A qvik transaction may be cancelled as long as it is in the INIT status. Such cancelling may be initiated at the “transactionCancel” endpoint, as discussed above.

3.19.8 qvik – IPN

IPN messages used for bank card payments are sent by the SimplePay system for qvik transactions. The call is fully identical with the process discussed in relation to the card IPN case.

3.19.9 qvik – timeout processes

Qvik transactions' timeout management is uniform and depend on payment situation. The payment situation is a parameter that can be set in the merchant admin's interface, specifying the types of EAM transactions that can be used for a given merchant account. When one you wants to manage different payment situations in one's merchant system, one needs to apply them in separate accounts.

The default values are as follows:

Online purchase (IPEW): 20 minutes

Purchase at a physical point of sale (IPPS): 10 minutes

Public utility bill (TBIL, ELEC, PHON, GASB, UBIL, WTER, COMT, OTLC, GOVT, INSU): 180 days

Commercial bill (IVPT): 60 days

Purchase through automated self-service machine (IPU2): 3 minutes

ATM purchase (and cash withdrawal) (CDCD, IPAT): 3 minutes

Payment between persons (MP2P): 5 days

Physical purchase initiated with Loyalty App (MP2B): 10 minutes

In the event that the transaction is created at the start endpoint, i.e. payment by bank card and qvik payment are also possible, then the timeout is calculated on the basis of the following rules.

1. If the merchant sends the transaction's date/time of expiry, that will be the valid timeout. If the date/time of expiry sent by the merchant is beyond the maximum value possible for the payment status set for the account, we notify the merchant of an error **if** the transaction was sent **with only** the **EAM** (qvik) payment method.
2. If the merchant does not send a date/time of expiry, it will be set based on the default timeout value set for the account. Qvik payment is not possible if this is beyond than the payment status set for the account
3. If no timeout value is set for the account, the system default value of 5 minutes is used, and qvik payment is only authorised if it is shorter than or equal to the possible value set for the payment status for the account

3.19.10 qvik – settlements

Financial settlement information:

- Hungarian version: <https://simplepartner.hu/download.php?target=reportformathu>
- English version: <https://simplepartner.hu/download.php?target=reportformaten>

Report formats' specification with sample files:

- Hungarian version: <https://simplepartner.hu/download.php?target=merchantguidehu>
- English version: <https://simplepartner.hu/download.php?target=merchantguideen>

3.20 ApplePay payment partner-side solution

In this case, the entire payment process remains within the merchant's system and there is no need to redirect the customer to the SimplePay payment page.

The process on a mobile device consists, from the buyer's perspective, of clicking on the ApplePay button at the end of the purchase when selecting payment methods, choosing which of the cards saved in the Apple system they want to pay with from the pop-up list, and initiating the payment. Everything else takes place in the background, the merchant's system providing the information eventually.

A QR code is displayed once payment has been initiated on the desktop computer, which the buyer scans with their Apple device and then completes the rest of the process as described above for mobile devices.

3.20.1 Merchant registration for ApplePay payments

Before starting payments, the merchant has to register in the Apple system. SimplePay's merchant admin page offers a simple solution for registration.

The registration page is accessible by clicking Apple Pay on the Account managing page after enabling the function for business. The process can be initiated on the page by clicking "Record new domains". The address that can be interpreted in the merchant system, where the merchant will apply the function, is to be entered in the pop-up window.

The fully qualified domain name (FQDN) has to be entered in the pop-up window. The FQDN contains the entire domain name, e.g. **samehost.example.com**, where "**samehost**" is the given host or subdomain name, "**example**" is the domain name with "**com**" being the top-level domain (TLD).

FQDN wikipedia:

Hungarian version: https://hu.wikipedia.org/wiki/Fully_qualified_domain_name

English version: https://en.wikipedia.org/wiki/Fully_qualified_domain_name

After entering the FQDN one can add multiple elements to the list by clicking "**Add**". Entering multiple FQDNs is advisable when the Apple Pay solution is to be accessible through multiple subdomains in a given merchant system.

Once every required FQDNs have been added to the list, the data can be saved by clicking "**Record**", after which every FQDN that has been added will appear in the list, and their status will be "**New**".

The entry on which registration is intended to be started can be marked by ticking the checkbox at the beginning of each row. The registration of the merchant website's FQDN towards Apple is started by clicking "**Registration**". When this process has been completed successfully, the status will be "**Registered**".

The merchant's ID will appear after registration in the "**Apple Pay Merchant Identifier**" field. This will only be needed for the initiation of transactions in a later stage of the implementation process.

3.20.2 Positioning of Well-known file for ApplePay payments

Apple controls the registered websites with the help of a so-called "well-known file". The file contains an identifier that identifies the SimplePay system to which the merchant has registered their website by entering the FQDN.

This file can be downloaded from the above registration page. The file is to be placed on the merchant's website, under the registered FQDN, through the following access route.

The file has to be publicly accessible for verification by Apple.

Sample access route as if the above example **samehost.example.com** FQDN were registered.
<https://samehost.example.com/.well-known/apple-developer-merchantid-domain-association>

3.20.3 ApplePay payment process

The payment process is made up of the following elements:

- in the merchant's system, the buyer selects the products, on the basis of which the amount payable is finalised
- a JavaScript frontend is to be used for initiating payment, showing a button meeting Apple's requirements
- as the customer clicks on the button, JavaScript runs a function registering the transaction with Apple via the SimplePay API and returns the result
- if this is successfully completed, then after the selection of the card, the JavaScript function runs, using the received token to initiate the debit transaction via the SimplePay API
- the API sends a message back to the merchant's system on the successful completion of the process

- based on the feedback, the merchant notifies the buyer of the successful completion of the payment transaction
- in addition to the API response, the buyer should wait to receive the IPN message as well because that is the point where the merchant can execute the settlement

3.20.4 ApplePay payment frontend

The merchant's system has multiple options for implementing the payment frontend. One example for these is the SimplePay SDK.

At the same time, the frontend can be generated in the merchant's system in a variety of ways, for which the developer can access information on the Apple developer page.

<https://developer.apple.com/documentation/applepayontheweb/apple-pay-is-api>

The following is an illustration of implementation on the basis of the sample JavaScript in the SDK.

The following are performed by the SDK Apple Pay JavaScript sample code:

- checking whether the payment concerned is technical feasible (whether the tool, the operating system is OK)
- taking over the necessary transaction parameters, including the amount payable
- validation of the merchant and the transaction to be started (/v2/startapplepay endpoint)
- in the case of successful validation, initiating payment by crd (/v2/doapplepay endpoint)
- notification about the result of the payment transaction

The following constant values must be assigned values according to the specific parameters of the merchant system in the SDK Apple Pay JavaScript sample code:

merchantIdentifier: the value of the “Apple Pay Merchant Identifier” generated on the SimplePay admin page

validationEndpoint: the endpoint in the merchant system to which the frontend JavaScript transmits the transaction data for validation. This element of the merchant system communicates with the SimplePay /v2/startapplepay endpoint.

authorizationEndpoint: the endpoint in the merchant system to which the frontend JavaScript transmits the data received after validation. This element of the merchant system communicates with the SimplePay /v2/doapplepay endpoint.

3.20.5 ApplePay payment backend, startapplepay

The startapplepay call is expected by the API at the following URL:

<https://sandbox.simplepay.hu/payment/v2/startapplepay>

In essence, the options of the SimplePay “startapplepay” endpoint are identical with those of the “start” endpoint; therefore, the following is a discussion of the differences only.

The following business logics applied at the “start” endpoint do not apply at the “startapplepay” endpoint:

- functions relating to card storage
- transfer-based transactions (WIRE, EAM)
- transactions generated for ling generation

- deeplink generation for redirection directly into Simple application
- back redirect parameters

One additional object needs to be sent at the endpoint in the JSON string: **domain**
The domain's value applies in the merchant system entered in registration FQDN

```
{
...
  "domain": "samehost.example.com"
...
}
```

Sample JSON request

```
{
  "salt": "0ad66088193d216347be6fbdbbb37c20",
  "merchant": "PUBLICTESTHUF",
  "orderRef": "ApplePay_10891617490169132208",
  "currency": "HUF",
  "sdkVersion": "SimplePay_PHP_SDK_2.5_250521:be716f6077d829a581de909ffd75e160",
  "total": 10,
  "customerEmail": "sdk_test@otpmobil.com",
  "language": "HU",
  "domain": "samehost.example.com",
  "timeout": "2025-09-15T11:25:37+02:00"
}
```

Sample JSON response

```
{
  "merchant": "PUBLICTESTHUF",
  "salt": "Ya95mYG6U0qqe2WIPKeyx013s81VAjk",
  "transactionId": "581841992",
  "applePaySession": {
    "epochTimestamp": 1749017418352,
    "expiresAt": 1749021018352,
    "merchantSessionIdentifier": "SSHCE5A2091BEC546FD8FACC2A313A2E8AF_2101F68F6980DFE07DEFE987B1CAF2961766C119C8FDCBB33566B1A97F33C9C3",
    "nonce": "836a2281",
    "merchantIdentifier": "D4D7D4CCCD55649F9B1BC1F06432938F62FE991826M4F4CD6829F058F422A26A",
    "domainName": "samehost.example.com",
    "displayName": "Operation test",
    "signature": "308008.....000000",
    "operationalAnalyticsIdentifier": "Operation test:D4D7D4CCCD55649F4B1BC1F06432938E62FE99182604F4CD6829F058F422A26A",
    "retries": 0,
    "pspId": "2A7CFE1FC43C6DE9FC90CA89E46C067DB94C570AF75A7355CD754BF1B683B6C5"
  },
  "timeout": "2025-06-04T08:15:17+02:00"
}
```

The response contains "**applePaySession**", in addition to the general data provided by the SimplePay API endpoints, i.e. is the result of validation between Apple and SimplePay.

In case validation fails for any reason, "**applePaySession**" will not be included in the response, instead, the error code will be included in "**errorCodes**".

3.20.6 ApplePay payment backend, doapplepay

The doapplepay call is expected by the API at the following URL:

https://sandbox.simplepay.hu/payment/v2/doapplepay

In essence, the options of the SimplePay "**doapplepay**" endpoint are identical with those of the stored "**do**" endpoint; therefore, the following is a discussion of the differences only.

The following business logics applied at the "**do**" endpoint do not apply at the "**doapplepay**" endpoint:

- functions relating to stored card payments
- back redirect parameters
- 3ds exemptions (type)

One additional object needs to be sent at the endpoint in the JSON string: **applePayToken**
Its value is the JSON value generated by Apple and returned to the client in the "**startapplepay**" response within the session, encoded in BASE64.

Sample JSON request

```
{
  "salt": "37fcee449022dfb0680f7853c5c1caa8",
  "merchant": "PUBLICTESTHUF",
  "transactionId": 581841992,
  "applePayToken": "eyJwYX.....J9fQ==",
  "sdkVersion": "SimplePay_PHP_SDK_2.5_250521:be716f6077d829a581de909ffd75e160",
  "currency": "HUF"
}
```

Sample JSON response

```
{
  "merchant": "PUBLICTESTHUF",
  "salt": "PAiRimYcI4eVXD3Qwz4naWhtyprasFh0",
  "orderRef": "ApplePay_10891617490169132208",
  "transactionId": 581841992,
  "currency": "HUF",
  "total": 10.0,
  "applePayStatus": "STATUS_SUCCESS"
}
```

The response contains "**applePayStatus**", in addition to the general data provided by the SimplePay API endpoints, i.e. is the result of the payment transaction between Apple and SimplePay.

If the transaction was unsuccessful for any reason whatsoever, the value of "**applePaySession**" in the response will be XXXX. Moreover, the error code will be included in "**errorCodes**".

4 PHP SDK

The PHP sample code (SDK) that can be used to implement the payment system is available for merchants using or developing the SimplePay system, together with this documentation.

SDK is a sample solution providing **one possible way of implementation** and only focuses on the technical realisation of the payment function. The SDK implements the API operation described in Chapter 3. The SDK sample code generates the JSON content necessary for the requests and the signatures required to validate messages. It performs communication towards the SimplePay system. It receives, validates and makes accessible the received responses.

Beyond the above, the SDK does not perform any other functions of the merchant system.

The sample code can be used under the terms of the GNU GPL3 license to initiate transactions in the SimplePay system. The possibility to use the SimplePay system is regulated in the SimplePay merchant agreement.

The sample code is capable of initiating payment transactions; therefore, **it is most essential to have a clear view of how the proprietary, host system operates** to be able to understand its logic accurately and to be able **to insert online payment** in the appropriate place.

With the help of the SDK, a functioning test system can be created within 15 minutes. Once you have unpacked this code on your server and set it up in accordance with the following, you can immediately try bank card payments in the SimplePay test system (sandbox). If you do this, the working sample code will show you exactly what you will need to implement in your own system.

Using the unchanged SDK as your own separate test system will greatly ease troubleshooting during development. In this case, if you run into a problem in the embedded code, you can immediately compare and check how a particular function operates or what can cause the problem in case of transactions initiated with identical data.

4.1 The structure of the SDK

The SDK is made up of the components described below.

/src

The folder contains the sample source code realising bank card payments in the **SimplePayV21.php** file. The file contains every necessary source code for all the functions that can be realised.

The same folder contains the **config.php** file. It is in this file where, among others, the merchant account data must be set, without which no transactions can be initiated.

/log

The default log folder of the SDK. If you would like to redirect logging to somewhere else, you can set your own logging path in the **config.php** file.

php files in the main directory

Every file is a sample code for a merchant-side function. Sample codes typically perform the following:

- they collect the data necessary for the transaction
- they run the class in the **src/SimplePayV21.php** file, configured with the appropriate parameters, which is necessary to perform the payment function
- they process and display the response data received

4.2 SDK config settings

Unpack the sample code on your server. Open the `src/config.php` file. The file contains an array and out of its elements only the merchant data and the URL need to be configured for the initial tests as described in the following. For the time being, the other parameters do not require any action for the first test.

4.2.1 Merchant account data

With the sample code, all three possible currencies can be used in parallel. If you make sales in more currencies on your website and if you have HUF, EUR and USD accounts, the sample code, based on the currency of the ongoing transaction, can switch automatically between the accounts having different currencies.

The account data can be provided in the following fields

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

You can find the data for these fields in the SimplePay merchant admin interface at the following URL: <https://sandbox.simplepay.hu/admin>

The header on the "**Account admin**" page shows the basic data of the account. One of them is the Merchant identifier (**MERCHANT**).

The key necessary to calculate the signature can be found among the "**Basic data**" on the "**Technical data**" subpage (**SECRET_KEY**).

These data vary in case of every account (currency).

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

4.2.2 URL's

Once payment has taken place, the payment page will redirect the customer to the merchant URL provided. Depending on which is more suitable for the merchant system, this URL can be provided using two types of logic.

In the first case, the redirection will occur to a common URL, not depending on result of the transaction. In this case, the merchant side will be able to determine the result of the transaction based on the values in the parameters when the URL is requested.

```
'URL' => 'https://' . $_SERVER['HTTP_HOST'] . '/back.php',
```

In the other case, the transaction will be redirected to a specific URL depending on what the result was.

```
'URLS_SUCCESS' => 'https://' . $_SERVER['HTTP_HOST'] . '/success.php',  
'URLS_FAIL' => 'https://' . $_SERVER['HTTP_HOST'] . '/success.php',  
'URLS_CANCEL' => 'https://' . $_SERVER['HTTP_HOST'] . '/cancel.php',  
'URLS_TIMEOUT' => 'https://' . $_SERVER['HTTP_HOST'] . '/timeout.php',
```

When payment is successful, the customer will be redirected to the URL provided in the **URLS_SUCCESS** variable from the SimplePay payment page.

When payment is unsuccessful, the customer will be redirected to the URL provided in the **URLS_FAIL** variable from the SimplePay payment page.

When payment is cancelled on the SimplePay payment page, the customer will be redirected to the URL provided in the **URLS_CANCEL** variable.

The customer is redirected to the URL provided in the **URLS_TIMEOUT** variable if the customer does not provide card details or does not initiate the payment for a period determined at the initiation.

IMPORTANT: test bank card details to be used for the transactions can be selected on the sandbox payment page.

4.2.3 Switching between test and live transactions

The use of the sandbox is turned on by default in the SDK.

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

If you would like to switch to the live system following successful tests, this variable needs to be set as **false**. The sandbox account will remain after the deployment, too, allowing you to perform tests if any development becomes necessary.

4.2.4 Logging

Logging provides important help in troubleshooting during development. It is turned on in the SDK by default, the only requirement is that the `sdk/log` folder should be writable.

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

4.3 Setting up SDK IPN

The Instant Payment Notification (IPN) is the end of the payment transaction. At this point, the SimplePay system notifies the merchant webstore about the success of the transaction. IPN is a background process between SimplePay and the merchant. The SimplePay system sends the IPN to the URL provided by the merchant with POST method.

IPN messages are **only sent for successful and performable transactions**. This clearly indicates the success of the transaction. If it is successful, the payment status in the merchant system can be set to paid—depending on the store—and the order can be performed.

The IPN URL is an address available online where the merchant **receives** the IPN message sent by the SimplePay system. This is the address where the SimplePay system will send the notification.

This data needs to be registered in the SimplePay system, which can be set in the merchant control panel. The data can be set under “**Technical data**”.

The access path of the `ipn.php` file in case of the sample code is:

“`https://domainev.hu/ipn.php`”

The IPN must be set for each account. If you use more than one account on a domain (e.g. payments are allowed both in HUF and EUR) then the URL must be provided for both cases (both accounts).

With regard to the IPN URL, the following must be considered

- they must be publicly available
- they must not be `.htpasswd` protected
- redirection rules, SSL settings must not make it inaccessible

The use of IPN will make the test system fully functional. You can find further details on the use of IPN in the chapter about the testing of the IPN.

4.4 start – create transaction

Sample code in the SDK: **start.php**

The sample code creates the JSON string included in the description of the “**start**” request in the chapter “**Implementation**”, sends it to the SimplePay API, then receives and validates the response.

The operating logic of **start** has been discussed in detail in chapter “**Implementation**”. In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

```
//Import config data and SimplePay class
require_once 'src/config.php';
require_once 'src/SimplePayV21.php';

// new SimplePayStart instance
$trx = new SimplePayStart;

//add config data
$trx->addConfig($config);

//add transaction data
$trx->addData('currency', 'HUF');
$trx->addData('total', '100');
$trx->addData('orderRef', '101010515363456734591');
$trx->addData('customerEmail', 'sdk_test@simplepay.com');
$trx->addData('language', 'HU');

$timeoutInSec = 600;
$trx->addData('timeout ', date("c", time() + $timeoutInSec));

$trx->addData('methods', array('CARD'));
$trx->addData('url', $config['URL']);

// invoice
$trx->addGroupData('invoice', 'name', 'SimplePay V2 Tester');
$trx->addGroupData('invoice', 'company', '');
$trx->addGroupData('invoice', 'country', 'hu');
$trx->addGroupData('invoice', 'state', 'Budapest');
$trx->addGroupData('invoice', 'city', 'Budapest');
$trx->addGroupData('invoice', 'zip', '1111');
$trx->addGroupData('invoice', 'address', 'Address 1');
$trx->addGroupData('invoice', 'address2', '');
$trx->addGroupData('invoice', 'phone', '06203164978');

//create transaction in SimplePay system
$trx->runStart();
```

A slightly more extended file matching the above sample can be found in the SDK under the title **start.php**.

The two includes at the beginning of the file contains the config details and the PHP code that performs the payment.

This is followed by the instantiation of the **SimplePayStart** class. This is the class that contains the sample code which creates the JSON string shown above required for the start communication and fills it up with data, calculates the Signature, sets the header, and sends the message to the SimplePay server.

The **addConfig()** function can be used to add the config data.

In the followings, we will add the data required for the transaction. The SDK offers two functions to do this.

addData()

You can add name/value pairs to the transactions with this function. Such could be, for example, a currency. The first parameter of the function is the name of a field, the second is its value.

```
$trx->addData('currency', 'HUF');
```

addGroupData()

You can add data to a group with this function. Such could be, for example, the city within the group of invoicing data. The first parameter of the function is the name of the group, the second is the name of the field, the third is its value.

```
$trx->addGroupData('invoice', 'city', 'Budapest');
```

runStart()

Once every necessary data has been added to the transaction, the Signature must be calculated, this data must be added to the header and the request must be initiated towards the SimplePay server.

All this is performed by the runStart() function.

```
//create transaction in SimplePay system  
$trx->runStart();
```

4.5 start – response

The sample code processes and validates the response to the “start” request JSON string.

The operating logic of **start** has been discussed in detail in chapter “**Implementation**”.

In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

In the SDK, the outcome of the `$trx->getReturnData()` function contains the result of the `runStart()` that was initiated earlier. The outcome of the function is an array which contains the JSON response, and also its content as elements in the array.

After the `runStart()` function, running the `getHtmlForm()` will create a completed form using the value of the `paymentUrl` component.

The form elements that trigger a transaction can be defined as follows.

```
$trx->formDetails['element'] = 'button';
```

The options are as follows:

- **button:** creates a button waiting for the customer to click. The redirection to the payment page will take place after clicking on it
- **link:** creates a link waiting for the customer to click. The redirection to the payment page will take place after clicking on it
- **auto:** instantly redirects to the payment page without waiting for customer interaction

```
$trx->getHtmlForm();
```

Using this the customer can be redirected to the payment page immediately. This can be found in the `form` field of the array.

```
Array
(
    [responseBody] => {"salt":"00Rbo29VweDZ8rCVW3THCmgDkh7Qtj3H", "merchant": "PUBLICTESTHUF", "orderRef": "101010515363456734591", "currency": "HUF", "transactionId": "99310118", "timeout": "2018-09-07T20:46:13+02:00", "total": 100.0, "paymentUrl": "https://sandbox.simplepay.hu/pay/pay/pspHU/aCKKahLcGamHCdLSARoOdC2jS5yBqgUSwSt6X4KsgWmod13zDc"}
    [responseSignature] => 3WGqCnWJArhA224xVdUY1fPh91tpd6va6JvBrPNuHK449TZTgsRn3DBu5UBGbcTn
    [responseSignatureValid] => 1
    [salt] => 00Rbo29VweDZ8rCVW3THCmgDkh7Qtj3H
    [merchant] => PUBLICTESTHUF
    [orderRef] => 101010515363456734591
    [currency] => HUF
    [transactionId] => 99310118
    [timeout] => 2018-09-07T20:46:13+02:00
    [total] => 100
    [paymentUrl] => https://sandbox.simplepay.hu/pay/pay/pspHU/aCKKahLcGamHCdLSARoOdC2jS5yBqgUSwSt6X4KsgWmod13zDc
    [form] => <form action="https://sandbox.simplepay.hu/pay/pay/pspHU/mueK6aKcDuhbPYpqQHQT0T10jk11BWgXRwSvwR4rvQ3sEXQbjBw" method="POST" id="SimplePayForm" accept-charset="UTF-8">
        <button type="submit">Start SimplePay Payment</button>
    </form>
)
```

Displaying the **form** element gives you a button or link, according to the value of `$trx->formDetails['element']`, given earlier. After displaying this, redirecting to the payment page will start either by customer clicking or automatically.

```
print $trx->transaction['form'];
```

4.6 SimplePay payment page

Up to this point the payment transaction has been created in the SimplePay system with the “**start**” function. The merchant system has received a URL from the SimplePay API where the customer must be redirected. At that URL, the SimplePay payment page is loaded showing the data of the newly created transaction.

The payment page displays the payment transaction's details provided earlier, the merchant to whom the customer will pay, and all the necessary information.

The functions available on the payment page have been discussed in detail in chapter “Implementation”.

The details of the card to be used for payment can be entered on this interface. In the sandbox system the card number field is actually a drop-down menu where you can select the test cards to simulate successful and unsuccessful payments. The outcome of the payment depends on the card selected.

The transaction is initiated by clicking on the “**PAY**” button. Then the card authorisation takes place, after which the customer is redirected to the website of the merchant.

4.7 back

After the transaction above, the redirection will be performed to the merchant website where authentication and processing of parameters sent by the GET method will be performed.

The operating logic of **back** has been discussed in detail in chapter “Implementation”. In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

```
//Import config data and SimplePay class
require_once 'src/config.php';
require_once 'src/SimplePayV21.php';

// new SimplePayBack instance
```

```
$trx = new SimplePayBack;

//add config data
$trx->addConfig($config);

//result
$result = array();
if (isset($_REQUEST['r']) && isset($_REQUEST['s'])) {
    if ($trx->isBackSignatureCheck($_REQUEST['r'], $_REQUEST['s'])) {
        $result = $trx->getRawNotification();
    }
}
```

The value of the `$result` variable is an array, the content of which is identical with the data in the JSON string.

```
Array
(
    [r] => 0
    [t] => 99310118
    [e] => SUCCESS
    [m] => PUBLICTESTHUF
    [o] => 101010515363456734591
)
```

4.8 Information on the merchant webpage, depending on the result of the transaction

The merchant website or the mobile application shall display information to the customer on the result of the transaction.

The necessary **back** information has already been discussed in chapter “**Implementation**”, including successful, failed and aborted transactions, as well as in case of timeout. The same information shall be provided in the case of using SDK.

4.9 IPN

The IPN message is sent by the SimplePay system to the merchant system. The message marks the end of a successful transaction, based on which the merchant can fulfil the order.

The operating logic of the IPN and its SimplePay-side setup can be found in Chapter 3.14. Hereinafter, its implementation using PHP SDK is described in this chapter.

```
//Import config data and SimplePay class
require_once 'src/config.php';
require_once 'src/SimplePayV21.php';

//input
$json = file_get_contents('php://input');
```

```
// new SimplePayIpn instance
$trx = new SimplePayIpn;

//add config data
$trx->addConfig($config);

// check signature
if ($trx->isIpnSignatureCheck($json)) {
    $trx->runIpnConfirm();
}
```

In the first step of IPN processing, the **isIpnSignatureCheck()** function will validate the request.

If the validation is successful, the **runIpnConfirm()** function will create the response message and the necessary Signature value. By default, the function will display the necessary response, too, which means that no further development is required apart from the code above.

Since the Signature value of the response must be returned in the header, which means that the header is modified, it is important that on the URL processing and responding to the IPN message **NO other data is displayed before the above code section runs.**

Since this is a process running in the background, no other data needs to be displayed at this point.

It is not obligatory to display the IPN message with the **runIpnConfirm()** function.

In this case, the **getIpnConfirmContent()** function must be requested as in the following.

```
$confirm = $trx->getIpnConfirmContent();
```

The function will create the necessary response and the content of the Signature, too, but it will not display them and the header will not be modified, either. The data will be included in the **\$confirm** array returned in the outcome of the function, extracting from which the data can be displayed at the location that is suitable for the merchant system.

4.10 finish – managing two-step transactions

The **finish** request is the end of the two-step transactions.

The sample code creates the required JSON string, sends it to the SimplePay API, then receives and validates the response.

The operating logic of **finish** has been discussed in detail in chapter “**Implementation**”. In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

Sample code in the SDK: **finish.php**

```
//Import config data
require_once 'src/config.php';

//Import SimplePayment class
require_once 'src/SimplePayV21.php';

// new SimplePayFinish instance
$trx = new SimplePayFinish;

//config
$trx->addConfig($config);

// transaction ID
$trx->addData('transactionId', '99326614');

// original total value
$trx->addData('originalTotal', 15);

// approved total value
$trx->addData('approveTotal', 15);

//currency
$trx->transactionBase['currency'] = 'HUF';

//start finish communication
$trx->runFinish();

//result
$result = $trx->getReturnData()
```

The response is included in the `$result` value

```
Array
(
    [responseBody] => {"salt":"1iAOP84G3nh0cNBCVijXqhY02PookbUv","merchant":"PUBLICTESTHUF","orderRef"
:"101010515385149346625","currency":"HUF","transactionId":99326614,"approveTotal":15.0}
    [responseSignature] => Hf3ZHmhtFpTK3HoLjVmYwQgCAFYu9RDFa5nx296pFwpTHw6XBLVq3eP0+VSSejwL
    [responseSignatureValid] => 1
    [salt] => 1iAOP84G3nh0cNBCVijXqhY02PookbUv
    [merchant] => PUBLICTESTHUF
    [orderRef] => 101010515385149346625
    [currency] => HUF
    [transactionId] => 99326614
    [approveTotal] => 15
)
```

4.11 refund – managing refunds

The SDK performs the **refund** request as in the following.

The sample code creates the required JSON string, sends it to the SimplePay API, then receives and validates the response.

The operating logic of **refund** has been discussed in detail in chapter “**Implementation**”. In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

Sample code in the SDK: **refund.php**

```
//Import config data
require_once 'src/config.php';

// Import SimplePayment class
require_once 'src/SimplePayV21.php';

// new SimplePayRefund instance
$trx = new SimplePayRefund();

//config
$trx->addConfig($config);

// add transaction ID
$trx->addData('transactionId', '99326864');

// amount to refund
$trx->addData('refundTotal', 5);

// add currency
$trx->addData('currency', 'HUF');

// start refund
$trx->runRefund();

// result
$result = $trx->getReturnData()
```

The response to the request is an array containing the following data.

```
Array
(
    [responseBody] => {"salt":"WZ7Ncc0qoDSMYG4twsme0dBs6PSnsj1Z","merchant":"PUBLICTESTHUF","orderRef":
:"101010515385577564359","currency":"HUF","transactionId":"99326864","refundTransactionId":99326874,"ref
undTotal":5.0,"remainingTotal":10.0}
    [responseSignature] => +vtH30/4TyiuLuWEqn4qCwyX+8xdLiuOPFczwrLRIFm7m44uJpF/VRmXgoS33pnk
    [responseSignatureValid] => 1
    [salt] => WZ7Ncc0qoDSMYG4twsme0dBs6PSnsj1Z
    [merchant] => PUBLICTESTHUF
    [orderRef] => 101010515385577564359
```

```
[currency] => HUF
[transactionId] => 99326864
[refundTransactionId] => 99326874
[refundTotal] => 5
[remainingTotal] => 10
)
```

Three key values of the response to the request in the array:

- **refundTransactionId**: transaction identifier of the refund
- **refundTotal**: the refunded amount
- **remainingTotal**: the remaining amount of the original transaction after this refund (original amount charged minus all refunds up to now)

4.12 query – querying the transaction data

Transaction data can be queried from the SimplePay system using the **query** function.

The **query** function can be applied as a fully functional alternative to IPN. The end status of the transactions can be viewed by querying, if no IPN handling is to be carried out in the merchant system.

The sample code creates the required JSON string, sends it to the SimplePay API, then receives and validates the response.

The operating logic of **query** has been discussed in detail in chapter “**Implementation**”. In addition, the implementation of the operating logic discussed therein using PHP SDK is described in this chapter.

Sample code in the SDK: **query.php**

```
//Import config data and SimplePay class
require_once 'src/config.php';
require_once 'src/SimplePayV21.php';

// new SimplePayQuery instance
$trx = new SimplePayQuery;

//add config data
$trx->addConfig($config);

//add SimplePay transaction ID
$trx->addSimplePayId('99325516');

//query
$trx->runQuery();

//result
$result = $trx->getReturnData();
```

SimplePay transaction identifiers can be added to the request with the **addSimplePayId()** function as in the following.

```
$trx->addSimplePayId('99325516');
```

Merchant transaction identifiers can be added to the request with the **addMerchantOrderId()** function as in the following.

```
$trx->addMerchantOrderId('101010515383930534733');
```

Multiple elements can be added to the query by requesting the **addSimplePayId()** and the **addMerchantOrderId()** functions multiple times, by which we can receive the data of multiple transactions at the same time.

In the above sample code, the query is performed by the **\$trx->runQuery()** function.

```
$trx->runQuery();
```

We can extract the response to the query with the **\$trx->getReturnData()** function, the content of which is structured as in the following. In the response, the transactions array will contain the details of the queried transaction(s). The value of **totalCount** contains the number of transactions found.

```
Array
(
    [responseBody] => {"salt":"YyUNzD6Qhw9iWGGq9hhpd7gfyIFuLL1o","merchant":"PUBLICTESTHUF","transactions":[{"salt":"zgtZuLZZVkuTDIPuv4mkxZQFsx4LNLSY","merchant":"PUBLICTESTHUF","orderRef":"101010515383923675972","transactionId":"99325516","status":"FINISHED","resultCode":"OK","remainingTotal":0.0,"paymentDate":"2018-10-01T13:12:47+02:00","finishDate":"2018-10-01T13:14:39+02:00","method":"CARD"}],"totalCount":1}
    [responseSignature] => QJRVV4lCxZ0zibXMsre+MEgnW+6T2xLCjNlFnmPEZ74wCnD5jW3r6+SFJAnvzcma
    [responseSignatureValid] => 1
    [salt] => YyUNzD6Qhw9iWGGq9hhpd7gfyIFuLL1o
    [merchant] => PUBLICTESTHUF
    [transactions] => Array
        (
            [0] => Array
                (
                    [salt] => zgtZuLZZVkuTDIPuv4mkxZQFsx4LNLSY
                    [merchant] => PUBLICTESTHUF
                    [orderRef] => 101010515383923675972
                    [total] => 25
                    [transactionId] => 99325516
                    [status] => FINISHED
                    [resultCode] => OK
                    [remainingTotal] => 0
                    [paymentDate] => 2018-10-01T13:12:47+02:00
                    [finishDate] => 2018-10-01T13:14:39+02:00
                    [method] => CARD
                )
        )
)
```

```
[totalCount] => 1
)
```

The query can be supplemented with the **"detailed"** parameter, and then the details of the transaction data will also be included in the response.

```
$trx->addData('detailed', true);
```

Detailed response

```
Array
(
    [responseBody] => {"salt":"djXXCGhaqbJx0Q9S6HgmaLo21b2kMjfm","merchant":"PUBLICTESTHUF","transactions":[{"salt":"WG9Uxej8IiU1seKt1ww8piAsRYnqynPj","merchant":"PUBLICTESTHUF","orderRef":"101010515383923675972","currency":"HUF","customer":"v2 START Tester","customerEmail":"sdk_test@simplepay.com","language":"HU","twoStep":false,"total":15.0,"shippingCost":0.0,"discount":0.0,"invoice":{"company":"","country":"hu","state":"Budapest","city":"Budapest","zip":"1111","address":"Address 1","address2":"","phone":"06203164978","lname":"SimplePay V2 Tester"},"delivery":{"company":"","country":"hu","state":"Budapest","city":"Budapest","zip":"1111","address":"Address 1","address2":"","phone":"06203164978","lname":"SimplePay V2 Tester"},"transactionId":"99325516","status":"FINISHED","resultCode":"OK","remainingTotal":0.0,"paymentDate":"2018-10-01T13:12:47+02:00","finishDate":"2018-10-01T13:14:39+02:00","method":"CARD"}],"totalCount":1}
    [responseSignature] => u5M4bBRe2P8Tr8nAeI/FMa0Hziqet1UE87GIgtkbAzCaw9VPbhe+9oXW0fyVKK4Q
    [responseSignatureValid] => 1
    [salt] => djXXCGhaqbJx0Q9S6HgmaLo21b2kMjfm
    [merchant] => PUBLICTESTHUF
    [transactions] => Array
        (
            [0] => Array
                (
                    [salt] => WG9Uxej8IiU1seKt1ww8piAsRYnqynPj
                    [merchant] => PUBLICTESTHUF
                    [orderRef] => 101010515383923675972
                    [currency] => HUF
                    [customer] => v2 START Tester
                    [customerEmail] => sdk_test@simplepay.com
                    [language] => HU
                    [twoStep] =>
                    [total] => 15
                    [shippingCost] => 0
                    [discount] => 0
                    [invoice] => Array
                        (
                            [company] =>
                            [country] => hu
                            [state] => Budapest
                            [city] => Budapest
                            [zip] => 1111
                            [address] => Address 1
                            [address2] =>
                            [phone] => 06203164978
                            [lname] => SimplePay V2 Tester
                        )
                    [delivery] => Array
                        (
                            [company] =>
```

```

        [country] => hu
        [state] => Budapest
        [city] => Budapest
        [zip] => 1111
        [address] => Address 1
        [address2] =>
        [phone] => 06203164978
        [lname] => SimplePay V2 Tester
    )

    [transactionId] => 99325516
    [status] => FINISHED
    [resultCode] => OK
    [remainingTotal] => 0
    [paymentDate] => 2018-10-01T13:12:47+02:00
    [finishDate] => 2018-10-01T13:14:39+02:00
    [method] => CARD
)

)

    [totalCount] => 1
)

```

The query can be supplemented with the **"refunds"** parameter, and the data of the refunds initiated on the transaction will be provided with the transaction data.

```
$trx->addData('refunds', true);
```

In case of responses supplemented with refunds, the **"refunds"** array will contain the data of the refunds initiated on the transaction in case of every transaction.

```

Array
(
    [responseBody] => {"salt":"QxQmqOfkqV9khWU6SHJx1KmYyuN74x1E", "merchant":"PUBLICTESTHUF", "transactions":[{"salt":"kK1f2RZJdtn5wHi0GBOqfNKE8yFtGwNW", "merchant":"PUBLICTESTHUF", "orderRef":"101010515384686499284", "transactionId":99326020, "status":"FINISHED", "resultCode":"OK", "refundStatus":"PARTIAL", "refunds":[{"transactionId":99326030, "refundTotal":-5.0, "refundDate":"2018-10-02T10:29:43+02:00", "status":"FINISHED"}], "remainingTotal":10.0, "paymentDate":"2018-10-02T10:24:09+02:00", "finishDate":"2018-10-02T10:29:28+02:00", "method":"CARD"}], "totalCount":1}
    [responseSignature] => sHitW57bS9UkeskilZh3mzo0LwuuIzxQBgFKDe770S0FosfN08VLbFidUyNIMNaq
    [responseSignatureValid] => 1
    [salt] => QxQmqOfkqV9khWU6SHJx1KmYyuN74x1E
    [merchant] => PUBLICTESTHUF
    [transactions] => Array
        (
            [0] => Array
                (
                    [salt] => kK1f2RZJdtn5wHi0GBOqfNKE8yFtGwNW
                    [merchant] => PUBLICTESTHUF
                    [orderRef] => 101010515384686499284
                    [transactionId] => 99326020
                    [status] => FINISHED
                    [resultCode] => OK
                    [refundStatus] => PARTIAL
                    [refunds] => Array
                )
            )
        )
)

```


- **secretKey** contains the unique SECRET_KEY value of the merchant account

5.1.1 PHP solution

```
$signature = base64_encode(hash_hmac('sha384', $jsonString, $secretKey, true));
```

5.2 API requests

The general message format of API requests can be found in chapter **3.1**. In the followings, this chapter will introduce how it can be implemented using various programming languages.

We will use the following variables in every API request example:

- **url** the API endpoint of SimplePay
- **jsonString** contains the content of the message
- **headers** contains the message header

5.2.1 PHP solution

```
$curlData = curl_init();  
curl_setopt($curlData, CURLOPT_URL, $url);  
curl_setopt($curlData, CURLOPT_POST, true);  
curl_setopt($curlData, CURLOPT_POSTFIELDS, $data);  
curl_setopt($curlData, CURLOPT_RETURNTRANSFER, true);  
curl_setopt($curlData, CURLOPT_USERAGENT, 'curl');  
curl_setopt($curlData, CURLOPT_TIMEOUT, 60);  
curl_setopt($curlData, CURLOPT_FOLLOWLOCATION, true);  
curl_setopt($curlData, CURLOPT_HTTPHEADER, $headers);  
  
//optional return header for result Signature check  
curl_setopt($curlData, CURLOPT_HEADER, true);  
  
$result = curl_exec($curlData);  
curl_close($curlData);
```

6 Error codes

Internal code	English text
0	Successful transaction
999	General error code.
1003	Rejected transaction
1200	Invalid signature
1400	Card storage error
1500	Stored card not found
1529	SimplePay internal error
1600	Invalid external identifier
1650	Transaction already exists (and is not marked as one that can be reinitiated).
1800	Invalid amount for stored card
1900	Internal error
2000	Invalid currency
2003	Entered password invalid
2004	General error code
2006	Entered merchant not found
2008	Entered email address incorrect
2010	Entered transaction ID incorrect
2013	Insufficient coverage on card
2014	Payment requires a password
2016	User interrupted payment
2019	Timeout in acceptor communication
2020	Acquiring bank error
2021	Card issuer requires interactive 3DS verification
2023	Repeated 3DS authentication failed. The transaction authentication has been closed with a failure result.
2030	Card cannot be cancelled because it has a positive balance / Entered amount incorrect
2040	Invalid currency
2063	Card inactive
2064	Incorrect bank card data
2065	Revocation of entered card is necessary / card does not exist
2066	Card may not be charged / due to exceeded limit
2068	Card data error or non-existent card.
2070	Incorrect type of card
2071	Incorrect bank card data / card does not exist
2072	You have entered your bank card details incorrectly. Please check the details and contact the bank that issued your bank card if necessary.
2073	You have entered your bank card details incorrectly. Please check the details and contact the bank that issued your bank card if necessary.

2074	Name of cardholder more than 32 characters
2077	Invalid CVC.
2078	General error, card issuing bank does not indicate cause of error
2079	Acquiring bank based on routing not available
2121	Invalid recurring amount
2130	Wrong input code.
2131	Wrong address entered.
2132	Wrong date of birth.
2999	SimplePay internal error.
3000	General 3DS error.
3001	Invalid 3DS response message.
3002	3DS process error - There is a technical error in the SimplePay system, or the card issuer has given a rejected/no response.
3003	3DS process error.
3004	Redirect required during 3DS challenge (the cardholder must be redirected to the issuer's ACS server using the provided URL).
3005	Interactive 3DS authentication required.
3006	3DS process error - No response from card company or delayed response.
3012	3DS process error, non-3DS image card, 3DS problem at a participant
3013	3DS process error
3101	Invalid 3DS response message
3102	Invalid 3DS version.
3103	3DS internal error
3201	3DS data gap
3202	3DS data error
3203	3DS data format error or missing required data.
3204	3DS internal error
3301	3DS internal error
3302	3DS internal error
3303	Internal 3DS error, invalid 3DS endpoint.
3304	3DS internal error
3305	3DS data error
3306	3DS data error
3307	3DS data error
3402	3DS authentication process timeout.
3403	Temporary 3DS system error.
3404	Persistent 3DS system error.
3405	3DS system error
3500	3DS ACS general error at issuer bank
3501	3DS ACS the user aborted the interactive authentication (challenge) process on the card issuer's side.
3504	3DS ACS timeout on the card issuer's side.

3505	3DS ACS timeout no response or delayed response from the card issuer's side.
3506	3DS ACS transaction error on the card issuer's side.
3507	3DS ACS unknown error on the card issuer's side.
3508	3DS ACS transaction error in a card issuer's system component.
3911	3DS data error
3912	3DS data error
5000	General error code
5010	Merchant account not found
5011	Transaction cannot be found
5012	Merchant account does not match
5013	Transaction already exists (and is not marked as one that can be reinitiated).
5014	The type of the transaction is not appropriate
5015	Transaction with ongoing payment
5016	Transaction timeout (in case of requests received from the accepting/acquirer side).
5017	Transaction cancelled (in case of requests received from the accepting/acquirer side).
5018	Transaction has been paid (no new operation can be initiated).
5019	Invalid transaction ID.
5020	Checking the value provided in the request or the original transaction amount ("originalTotal") failed
5021	The transaction has been closed (therefore no new Finish operation can be initiated).
5022	The transaction is not in the expected state required for the request.
5023	Unknown / incorrect account currency.
5024	Invalid email address.
5025	Invalid name.
5026	Transaction rejected (as result of an unsuccessful fraud inspection).
5027	Transaction blocking issue.
5028	Transaction not reconciled.
5029	Transaction not refunded yet.
5030	Operation not permitted
5033	Cancelling of refund transaction not authorised.
5040	Stored card not found
5041	Stored card expired
5042	Stored card inactivated
5043	Phantom stored card.
5044	Recurring not authorised
5045	Remote ID required for recurring registration.
5046	IDTS required for recurring registration.
5047	Recurring IDTS.
5048	Recurring until necessary
5049	Recurring until varies

5050	Invalid payment method.
5051	Invalid reference number.
5052	Invalid external reference.
5053	Invalid time stamp.
5054	Invalid invoicing given name.
5055	Invalid invoicing family name.
5056	Invalid invoicing family name.
5057	Invalid invoicing phone number.
5058	Invalid invoicing address.
5059	Invalid invoicing city.
5060	Invalid delivery given name.
5061	Invalid delivery family name.
5062	Invalid delivery phone number.
5063	Invalid delivery address.
5064	Invalid delivery city.
5065	Invalid amount.
5066	Invalid currency.
5067	Mandatory invoicing email address missing.
5068	Invalid delivery email address.
5069	You have reached maximum amount allowed to be spent.
5070	Error in generating new direct payment.
5071	Stored card invalid length
5072	Stored card invalid transaction
5080	Original ID differs in recurring registration.
5081	Recurring token not found
5082	Recurring token in use
5083	Token times necessary
5084	Token times too large
5085	Token until necessary
5086	Token until too large
5087	Token maxAmount necessary
5088	Token maxAmount too large
5089	Recurring and oneclick registration cannot be launched simultaneously in one transaction
5090	Recurring token necessary
5091	Recurring token inactive
5092	Recurring token expired
5093	Recurring account variation
5094	Invalid timeout definition.
5095	Payment method selected in advance is necessary.
5096	EAM use not authorised.

5097	Invalid EAM type.
5100	Invalid signature.
5101	Missing order reference.
5102	Missing order amount.
5103	Missing order currency.
5104	Missing date.
5105	Error during confirmation.
5106	Invalid refund amount.
5110	The amount to be refunded is not adequate.
5111	Sending either the orderRef or the transactionId is required
5113	The name and version number of the requesting client program ("sdkVersion") are required.
5150	Missing iOS merchant ID.
5151	Missing external reference.
5199	Invalid card data.
5200	Invalid invoice.
5201	The identifier of the merchant account ("merchant") is missing.
5202	Access denied.
5203	Invalid data.
5204	Invalid product code.
5205	Invalid product name.
5206	Invalid product information.
5207	Invalid product price.
5208	Invalid VAT amount.
5209	Invalid amount.
5210	Invalid product version.
5211	Invalid product group.
5212	Invalid price.
5213	The transaction identifier of the merchant ("orderRef") is missing.
5214	Invalid date.
5215	Invalid discount rate.
5216	Invalid delivery amount
5217	Invalid method.
5218	Invalid redirect URL.
5219	The e-mail address ("customerEmail") is missing or not in an e-mail format.
5220	The language ("language") of the transaction is not adequate
5221	Invalid discount format.
5222	Invalid delivery format.
5223	The currency ("currency") of the transaction is not adequate or missing.
5224	Invalid time limit.
5225	Invalid bank transfer time limit.

5226	Invalid communication state.
5227	Communication error.
5228	Refund limit overrun.
5229	Refund deadline expired.
5230	Refund not authorised.
5301	User redirection required.
5302	The incoming request contains a signature ("signature") that is not adequate. (The signature of the request received by the merchant API was not successfully checked.)
5303	The incoming request contains a signature ("signature") that is not adequate. (The signature of the request received by the merchant API was not successfully checked.)
5304	Unsuccessful call due to timeout.
5305	Forwarding the transaction to the payment system (accepting/acquirer side) is not successful.
5306	Creating the transaction failed
5307	The currency ("currency") provided in the request does not match the one set in the account.
5308	The two-step transaction received in the request is not authorised in the merchant account
5309	Recipient missing in the invoicing details ("name" in case of natural persons and "company" in case of legal persons).
5310	In the invoicing details, city is required.
5311	In the invoicing details, postal code is required.
5312	In the invoicing details, the first line of the address is required.
5313	In the list of items to be purchased ("items"), the name of the item ("title") is required.
5314	In the list of items to be purchased ("items"), the unit price of the item ("price") is required.
5315	In the list of items to be purchased ("items"), the ordered amount ("amount") must be a positive integer.
5316	In the delivery information, the recipient is required ("name" in case of natural persons and "company" in case of legal persons).
5317	In the delivery information, city is required.
5318	In the delivery information, postal code is required.
5319	In the delivery information, the first line of the address is required.
5320	The name and version number of the requesting client program ("sdkVersion") are required.
5321	Format error / invalid JSON string
5322	Invalid country
5323	Closing amount invalid
5324	List of items ("items") or transaction total ("total") necessary
5325	Invalid URL
5326	Missing cardId

5327	Maximum number (50) of merchant transaction identifiers (“orderRefs”) to be queried is exceeded.
5328	Maximum number (50) of SimplePay transaction identifiers (“transactionIds”) to be queried is exceeded.
5329	In the transaction initiation period to be queried the “from” must precede the “until” time.
5330	In the transaction initiation period to be queried the “from” and the “until” must be provided together.
5331	Invalid transaction source.
5332	Transaction ID and target required.
5333	Missing transaction ID
5334	Bank card data required.
5335	Bank card BIN required.
5336	Anti-fraud hash required.
5337	Error when complex data is transcribed into text.
5338	Acquirer’s transaction ID required.
5339	In connection with the transactions to be queried, either the initiation period (“from” and “until”) or the list of identifiers (“orderRefs” or “transactionIds”) must be provided.
5340	Transaction not linked to wallet.
5341	Partner option necessary.
5342	Partner invoices required.
5343	Incorrect transaction status
5344	Incorrect transaction status
5345	VAT amount smaller than 0
5346	Invalid transaction mode.
5347	Invalid Auchan payment in instalments.
5348	JWT required.
5349	Transaction not authorised in clearing account (AMEX, TSP)
5350	Invalid email
5351	Invalid day
5352	Simple business account error / account does not exist
5353	Invalid starting and closing parameters.
5354	Invalid device ID.
5355	No SoftPOS invoice.
5356	Invalid template.
5357	Invalid order reference.
5358	Invalid period.
5359	SoftPOS not accessible.
5360	Transaction’s remote assignment failed.
5361	Transaction’s remote cancellation failed.
5362	Transaction remote expiry setting failed.

5363	Original transaction required.
5364	Invalid additional information.
5365	Partner ID specification is a mandatory requirement.
5366	Invalid EAM status.
5367	Adding payment reference is a mandatory requirement.
5368	Borrower's bank BIC is required.
5369	End-to-end ID specification is a mandatory requirement.
5370	Invalid product price.
5371	Invalid product quantity.
5372	Amount paid to merchant.
5373	Apple Pay not allowed
5380	Email parameters required.
5381	Email type required.
5382	Customer's email address required.
5401	Invalid salt, not 32-64 long
5402	Basis of transaction required.
5403	Simple transaction underlying the payment.
5404	Invalid transaction state.
5405	The transaction is not in CDE work session.
5413	Created transfer transaction
5501	Browser accept mandatory
5502	Browser agent mandatory
5503	Browser IP mandatory
5504	Browser Java mandatory
5505	Browser language mandatory
5506	Browser colour mandatory
5507	Browser height mandatory
5508	Browser width mandatory
5509	Browser TZ mandatory
5511	Invalid browser accept
5512	Invalid browser agent
5513	Invalid browser IP address
5514	Invalid browser Java
5515	Invalid browser language
5516	Invalid browser colour
5517	Invalid browser height
5518	Invalid browser width
5519	Invalid browser TZ
5530	Invalid type
5555	Error during the Raven call.
5601	Invalid identification code.

5602	Invalid message.
5603	Invalid transaction situation.
5604	Invalid retail data.
5605	Invalid device.
5606	Invalid invoice ID.
5607	Invalid customer data.
5608	Invalid verification code.
5700	Missing successful authentication communication.
5701	Offer already sent.
5702	Missing data on payment by instalment.
5703	Invalid option index.
5704	Domain selection not available.
5705	Pay total amount not available.
5706	Error during acquiring call.
5707	Instalment payment data interpretation error.
5708	Invalid instalment payment number.
5709	Missing instalment payment option.
5710	Missing successful offering communication.
5711	Settlement invoice ID required.
5712	Series size necessary.
5812	Invalid card type.
5813	Card declined
5814	API v1 version blocked.
5900	Invalid application platform.
5901	Invalid time stamp.
5902	Mobile application not found.
5903	Deeplink data package not found.
5904	Platform or internal IDs required.
5905	Recurring URLs required.
5906	Multiple URLs entered.

Payment request (RTP) error codes

Internal code	English text
6100	RTP (Real-Time Payment) not authorised.
6101	Invalid package reference.
6102	Invalid order reference.
6103	Invalid total amount.
6104	Invalid currency.
6105	Customer required.
6106	Customer's email address or bank account number required.
6107	Invalid customer email address.
6108	Invalid customer bank account number.
6109	Invalid recurring periods.
6110	Recurrence period required.
6111	Recurrence period overrun.
6112	Invalid recurring date.
6113	Invalid recurring period.
6114	Empty payments.
6115	Invalid payment result.
6116	Invalid merchant.
6117	SDK version required.
6118	The transaction already exists.
6119	Invalid language.
6120	Invalid transaction state.
6121	Invalid package or order reference.
6122	Invalid query parameters.
6123	Order ID size overrun.
6124	The starting date is after the closing date.
6125	The starting date and the closing date are needed together.
6126	Missing RTP ID.
6127	Invalid status.
6128	Invalid deadline.
6129	Order reference or transaction ID is required for querying.
6130	Giro error.
6131	Transaction does not exist.
6132	Customer or bank account is required.
6133	Too long note.
6134	The bank account has already been set.
6135	Limit on the number of transactions overrun.
6136	Blank amount to be paid at RTP.

6140	Transaction initialisation.
6141	Waiting for bank account data.
6142	Transaction ready to be sent.
6143	Transaction sent.
6144	Transaction received.
6145	Transaction accepted.
6146	Transaction rejected.
6147	Transaction expired.
6148	Transaction reversed.
6149	Failed transaction.
6150	Unchangeable transaction.
6151	Paid transaction.
6153	Transaction not found.
6154	Transaction in end status.
6155	Invalid customer bank account.
6156	Reversal in progress.
6157	Transactions required.
6999	General error.
7100	Merchant not found.
7999	General error.

7 Logos and information pages

The SimplePay logo must be displayed in a permanently visible section of the part of the payment acceptor's site (e.g. in the footer), or during the payment method selection for the transaction.

The SimplePay logo is trademark and copyright protected; therefore, merchants can only use the SimplePay logo in the manner and for the purposes determined in the SimplePay GTC.

The logo cannot be transparent and its size can only be adjusted as long as good visibility is ensured. The file containing the SimplePay logos may be downloaded from the URL below: <https://simplepartner.hu/download.php?target=logo>

The logo must at the same time serve as a link to the payment information page. The Payment Information to be linked from the logos are available at the URL below:

In Hungarian: https://simplepartner.hu/PaymentService/Fizetesi_tajekoztato.pdf

In English: https://simplepartner.hu/PaymentService/Payment_information.pdf

The logo and the payment information it links to can be displayed with the following sample code

The content of the src (marked in red) is the access path of the logo on your server.

```
<a href="https://simplepartner.hu/PaymentService/Fizetesi_tajekoztato.pdf" target="_blank">  
    
</a>
```

8 Data Transfer Declaration

As the merchant transfers the order/customer data to a third party, the **customer must** explicitly **accept the Data Transfer Declaration**.

There are more possibilities to place the declaration

- directly displaying it at the payment
- in the General Terms and Conditions of the page
- in the Data Transfer Declaration of the page

IMPORTANT: it is not sufficient to place the declaration on the website if the customer does not encounter it and has not accepted it.

It may be accepted with a checkbox or by unambiguously indicating when the transaction is initiated that they accept the declaration by initiating the payment.

In the text of the declaration below, the highlighted sections of the declaration must be completed with the real-life merchant data in the following manner.

Merchant's name of company: the name of company provided in the contract.

Registered seat: the registered seat provided in the contract.

Web address of the Payment Acceptor: the domain name provided in the contract, for applications this must be supplemented with the name of the application.

Denomination of the data transmitted by the merchant: all the customer data that are transmitted during the transaction, e.g. name, email address, etc.

Declaration in English

I hereby acknowledge that my following personal data stored by the data controller **[Merchant's company name] ([seat])** in the user database **[web address of the Payment Acceptor]** will be transferred to SimplePay Plc., as data processor. The scope of data transferred by the controller: **[List of data transferred by the merchant]**

The nature and purpose of the data processing operations performed by the processor are available in the SimplePay Privacy Policy at the following link:

<https://simplepay.hu/adatkezelesi-tajekoztatok/>

Declaration in Hungarian

Tudomásul veszem, hogy a(z) **[Company Name] ([Company address])** adatkezelő által a(z) **[Paying Acceptance Web site]** felhasználói adatbázisában tárolt alábbi személyes adataim átadásra kerülnek a SimplePay Zrt., mint adatfeldolgozó részére. Az adatkezelő által továbbított adatok köre az alábbi: **[data transmitted by the merchant]**

Az adatfeldolgozó által végzett adatfeldolgozási tevékenység jellege és célja a SimplePay Adatkezelési tájékoztatóban, az alábbi linken tekinthető meg:

<https://simplepay.hu/adatkezelesi-tajekoztatok/>

9 Testing

Every webstore or mobile application is tested by SimplePay before deployment.

9.1 Start of testing

Testing starts upon notice by the merchant or its developer. After the completion of the development, give notification of access for SimplePay side checks to the completed payment system at itsupport@simplepay.com, along with any information necessary for accessing payment.

9.2 The purpose of the tests

The tests check the implementation of payment on websites or in the mobile application subscribing to the SimplePay service from the aspect of the customers.

9.3 The testing site

The development can be carried out in a merchant test system if there is one available. The merchant test system does not need to be under the domain name determined in the contract; it can be tested anywhere where it is publicly accessible.

If there is no separate test system available, the required tests can be carried out in an already operating website, too. In this case, it is recommended to display a warning at the payment method selector that bank card payment is under testing and should therefore not be selected for purchases.

9.4 Technical background of the merchant system

The tests are independent of the server environment, the operating system or the programming language used by the merchant to develop or operate its system into which it integrates the SimplePay payment.

9.5 Using third-party solutions

The test only covers the SimplePay compliance of the website / mobile application / online payments of the contracted merchant. In the course of this, it is not examined what other development, third-party software or online services are used for the technical implementation of the payment function or its various parts.

In line with the above, the tests do not cover the system that provides the technical (partial) solution, in every case they cover the payments that can be initiated from the partner website, mobile application of the merchant subscribing to the SimplePay service.

Such components developed or operated by third parties may be, for example:

- webstores
- embeddable modules
- gateway solutions, API's
- custom-developed software

If the merchant uses third-party solutions to implement the SimplePay payment system, the merchant shall be responsible in every case for the one-time set-up or the continuous operation of the software or service used by it.

9.6 Mandatory test elements for bank card payments

9.6.1 Successful transaction

- The transaction runs appropriately until the end
- The information described in chapter “**Successful payment**” is displayed on the **back** page
- Receipt of IPN messages and proper response based on **IPN** chapter **4.9** or by **querying** the transaction's end status.

9.6.2 Failed transaction

- The transaction runs appropriately until the end
- The information described in chapter “**Failed payment**” is displayed on the **back** page

9.6.3 Timeout

- The transaction runs appropriately until the end
- The information described in chapter “**Timeout**” is displayed on the **timeout** page

9.6.4 Cancelled transaction

- The transaction runs appropriately until the end
- The information described in chapter “**Cancelled payment**” is displayed on the **cancel** page

9.6.5 Display of the SimplePay Logo

- The SimplePay logo is displayed as described in chapter “**Logos and information pages**”

9.6.6 Data Transfer Declaration

- The necessary declaration is displayed before starting the transaction, as described in chapter “**Data Transfer Declaration**”

9.7 Components not tested

SimplePay tests are conducted in every case through the browser or the mobile application handed over for testing and cover only SimplePay requirements and payment transactions.

As a result, **the followings are not included in the tests:**

- logging in to the server serving the system (via ssh or any other channel)
- web login to the admin interface of the system used for the website
- configuring the server/database supporting the system
- login to the system database
- login to the system admin interface
- checking, editing the source code of the system
- testing the technical operation of the system (beyond what is required by SimplePay)
- testing the business logic of the system (beyond what is required by SimplePay)

- testing the appearance of the system (beyond what is required by SimplePay)
- separately testing, configuring third-party software that is integrated into the system
- separately testing, configuring services operated by third parties (gateway, API) that are integrated into the system

10 SZÉP card acceptance

Payment with SZÉP Cards—issued by OTP—may also be initiated with the start endpoint. In this case, the system can recognise the SZÉP Card on the basis of the card BIN (the first 6 characters of the card number) and send it on accordingly for debiting.

When the service was available the earlier SZÉP Card pockets were merged already; therefore, **the pocket—whose indication used to be a mandatory requirement—no longer needs to be specified and submitted.**

In case the merchant system wishes to send the pocket, it can be received by the API. In such cases the data should be sent in the **pocket** variable. Due to the merger only the value "09" of the previous value set is currently applied here, while the Aktív Magyarok pocket, created in 2025, can be identified with "08". If any other value is sent, it will not be taken into account by the API.

```
{  
  "pocket": "09"  
}
```

The transaction's statuses are the same as that of normal bank card payment.

SZÉP card payment can only work on merchant accounts separated from bank card payments. Consequently, transactions on the live and sandbox systems **require a merchant account configured to accept SZÉP cards only.**

10.1 SZÉP Card merchant's transaction identifier length

Unlike what is described in the SimplePay API documentation, in the case of TSP (i.e. SZÉP Card payment issued by OTP), the merchant transaction identifier must comply with the following:

- **length: maximum 32 characters**
- **content: alphanumeric string**

SZÉP Card payment, special attention must be paid to this!

10.2 SZÉP Card test in sandbox environment

Test transaction with SZÉP Card can be initiated on Sandbox with pre-set test card. Click on the card icon to select the card you wish to use for payment on the payment form. The following needs to be selected in the case of a SZÉP card:

number: 6101324296690851

expiry: 0926

holder: Sandbox Test

10.3 SZÉP Card and 3DS

3DS does not apply in the case of SZÉP card acceptance. There being no strong customer authentication, the challenge process cannot take place either.

11 Support

For further information or technical support, please contact us at itsupport@simplepay.com. For faster processing, please provide us the data identifying the problem or your inquiry.

Transaction

If you have questions regarding transactions, please provide us the **SimplePay** identifier of the payment. Its format is **5xxxxxxxx** in the case of sandbox and **7xxxxxxxx** in the case of live transactions currently.

Interface

Whether the transaction was started using V1 (not supported in the case of new connections) or V2 API.

Merchant account

Regarding technical settings, please provide the identifier of the **merchant's account** within the SimplePay system. The identifier is the account's **MERCHANT value**.

Payment system

The system your inquiry relates to. **Sandbox system** exclusively for tests, and **Live system** for live payment transactions.

Deployment

For deployment tests, please contact us at itsupport@simplepay.com and let us know

- under which contracted domain name the testable payment was developed
- which account is in use (MERCHANT)
- where we can access the testable system
- in the case of mobile applications please send it or, depending on the platform, set up tester access to the previously agreed tester email address

Annexes

I. Payment page implementation in mobile client / social login to Simple account

The SimplePay payment page can also be embedded in a mobile client in addition to a desktop one. In such a case, processes applied in relation to the desktop are performed, i.e. redirecting to the payment page, then redirecting back to the merchant system, but this necessitates a technical solution characteristic of mobile applications.

Furthermore, in this case, on the SimplePay payment page, the customer has the opportunity to pay using their bank card registered in the Simple mobile application. In such cases, the bank card details do not need to be submitted on the payment page. In order to use the registered bank card, the customer on the payment page must log into the Simple system where payment can be initiated. They can use a Google or Facebook account for social login.

Use of the social login does not require any development on the merchant side, but the method of embedding the payment page in the mobile application may significantly affect the security and efficiency of the process.

In the merchant **mobile application, we recommend use of the CustomTab and SF SafariViewControllor applications**, based on the examples below.

However, **we do not recommend** display of the SimplePay payment page in WebView / UIWebView for vulnerability reasons, and problems may also occur with a social login (Google / Facebook) into the Simple account.

Android

Using Chrome Custom Tab (preferred solution)

Sample code:

<https://simplepartner.hu/download.php?target=androidexample>

Chrome Custom Tab is opened instead of WebView as in the following

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

On the Activity opening the CustomTab, in the AndroidManifest.xml file the launchMode must be set to singleTop, singleTask or singleInstance, depending on the structure of the app in question. Usually singleTop is sufficient, the others may be used for more complex applications.

Defining an Intent-filter with a completely unique scheme for the Activity opening the CustomTab 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="uniqueHost"
            android:scheme="uniqueScheme" />
    </intent-filter>
</activity>
```

When launching transactions, a url complying with the unique **scheme://host** structure described in the previous point must be forwarded to the redirect url.

Additional documentation:

<https://developer.chrome.com/docs/android/custom-tabs/overview/>

Google Pay payment testing in Sandbox environment

For testing Google Pay payments in Sandbox environment the Google Pay sandbox cards appearing after selecting the payment method can be used.

Note: the card marked (Test Card: Visa Visa ••1111) appearing first among the available cards performs a failed card function in the SimplePay Sandbox system; therefore, any payment initiated with it will result in an error transaction (also when testing Google Pay payments).

iOS

The preferred solution is the use of the SFSafariViewController instead of the UIWebView UI component. With this solution, login will practically continue to take place within the application.

Sample code:

<https://simplepartner.hu/download.php?target=iosexample>

```
let safariViewController = SFSafariViewController(url: url)
safariViewController?.delegate = self
present(safariViewController!, animated: true, completion: nil)
```

Another potential solution is opening the Safari web browser. With this solution, it will leave the application and open a browser on the telephone, then, following a successful login, it will redirect the user back to the application.

```
UIApplication.shared.open(url, options: [:], completionHandler: nil)
```

Apple Pay payment testing in Sandbox environment

The following are required for testing Apple Pay payment in Sandbox environment:

1. Apple developer account created at the <https://appstoreconnect.apple.com/access/users/sandbox> address
2. Setting of adequate region: Country: Hungary
3. Signing in at <https://www.icloud.com/> with email and password
4. Entering developer account through iOS/Mac device
5. Starting SimplePay transaction with Apple Pay enabled
6. Payment in browser using Apple Pay.
 - a. If no Apple Pay sandbox card has been recorded yet, adding new one from: <https://developer.apple.com/apple-pay/sandbox-testing/> e.g. the following:

MasterCard: 5204 2452 5046 0049

Expiration Date: 01/30

CVC: 111

VISA: 4761 2297 0015 0465

Expiration Date: 01/27

CVV: 175

7. Enter telephone pass code.
8. Successful transaction.

II. Redirect between the Simple application and the merchant application (deeplink)

Pre-payment deeplink from merchant app to Simple app

If payment is initiated from a merchant mobile application and you only intend to initiate a Simple app payment from there, you can generate a deeplink pointing directly to the Simple app in the start call, in which case the merchant system receives the deeplink in the **paymentUrl** field in the response received at the start endpoint.

```
"paymentUrl": "simple://psp/pspHU.SqXNQXMqWzLfu0w7#link"
```

This feature can be enabled/disabled on the merchant admin interface on the "Technical Settings" page in the "Payment Page Settings" panel in the "Simple App Navigation Deeplink" checkbox.

Post-payment redirect deeplink from the Simple app to the merchant app

If the SimplePay payment page is embedded in the merchant's mobile application with CustomTab / SFSafariViewController, the customer has several options to initiate payment:

- they can provide their card details
- they can login to the Simple system using their email and password or through a social login option
- they can use the Simple application

If the user selects the Simple application on the payment page, which means they navigate from the merchant application to the Simple application and then make the payment there with their saved bank card, it may be important to be able to navigate back from the Simple application to the merchant application after payment.

Therefore, when the merchant's system initiates the transaction, it shall include the deeplink required to redirection between the two systems in the start request.

The deeplink URL can be customised by the merchant, but shall follow the standard **schema://host** structure, like: merchant://payment/<transactionID>

```
myAppS01234://payment/101010515833121594393
```

The deeplink value shall be transferred in the **start** request as the value of the mobilApp, as follows

```
"mobilApp": {  
  "simpleAppBackUrl": "myAppS01234://payment/123456789"  
},
```

The same can be specified in the SDK **start** request as follows

```
$trx->addGroupData('mobilApp', 'simpleAppBackUrl', 'myAppS01234://payment/123456789');
```

The merchant's mobile application shall be prepared to receive the customer returning from the Simple application on the specified route. If the deeplink URL is transferred incorrectly or incompletely, SimplePay cannot correct it, in which case navigation is not ensured.

Navigation does not differentiate between successful and failed transactions, it can navigate to a specific point, so the merchant application has to display the appropriate closing screen.

The content of the closing screen may be based on the data in the **query** request or the **IPN** received through backend.

The merchant may use the SimplePay payment page site as the destination of the inbound navigation. After the redirection from the Simple app, the SimplePay payment page displayed there will automatically perform the additional redirection, which can be closed as a **back** redirect, as a transaction in a normal browser. The post-redirect process can depend heavily on the unique operation of the merchant's application.

Android

The specified deeplink shall be defined in the AndroidManifest.xml file so that the application can process it

```
...  
<activity  
  android:name=".DeepLinkActivity"  
  android:label="@string/deeplink_name">  
  <intent-filter>  
    <category android:name="android.intent.category.DEFAULT" />  
    <category android:name="android.intent.category.BROWSABLE" />  
    <action android:name="android.intent.action.VIEW" />  
    <data  
      android:host="payment/123456789"  
      android:scheme=" myAppS01234" />  
  </intent-filter>  
</activity>  
...
```

When the `DeepLinkActivity` class used in the example is called during redirection, it already contains the data:

```
public class DeepLinkActivity extends Activity {
    @Override    protected void onCreate(Bundle savedInstanceState) {
        ...
        Uri data = getIntent().getData();
        ...
    }
}
```

iOS

The specified deeplink shall be defined in the `Info.plist` file so that the application can process it.

```
<key>CFBundleURLTypes</key>
  <array>
    <dict>
      ...
      <key>CFBundleURLSchemes</key>
      <array>
        <string>myAppS01234</string>
      </array>
      ...
    </dict>
  </array>
</key>
```

Requests received by the URL scheme can be filtered in the `AppDelegate` class.

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    ...
    func application(_ app: UIApplication, open url: URL, options: [UIApplication.OpenURLOptions
Key : Any] = [:]) -> Bool {
        guard url.absoluteString.hasPrefix("myAppS01234") else { return false }
        ...
        return true
    }
}
```

III. EMV 3D Secure

Legislative and supervisory bodies of the European Union and of the Member State, the relevant European Union Directive (Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015; **PSD2**), or by adapting it to national legislation, a so-called Mandatory **Strong Customer Authentication (SCA)** is required with effect from **14 September 2019** for the acceptance of all cash substitute currency issued within the European Economic Area.

Hungarian legal background

Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 has been adapted to the following legislation:

- Act CCXXXVII of 2013 on Credit Institutions and Financial Enterprises
- Act LXXXV of 2009 on the Pursuit of the Business of Payment Services
- Act CCXXXV of 2013 on Certain Payment Providers

The PSD2 modifications were included, among others, in Act CXLV of 2017 published in the 184th Hungarian Bulletin of 2017 and the Delegated Regulation No. 2018/389 of the European Commission.

With regard to bank card acceptance, the regulation also covers acceptance over the internet (so-called VPOS). In accordance with PSD2 compliance, and to make card acceptance over the internet safer, the cardholder's enhanced security check, the so-called EMV 3D Secure 2.0 service, had to be introduced and fully implemented by 14 September 2020.

3D Secure in practice

The technology related to the regulation has been fully introduced by SimplePay Plc.

If bank card payment is used in a so-called three-player way, **according to this document**, so the merchant system does not use extra service elements based on card storage, **then the merchant has no further action to make.**

At the same time, the provision of data from the merchant system **required by the EMV 3D Secure 2.0 standard is essential for the implementation of 3D Secure.** To do this, the merchant system needs to parameterise each transaction initiated with the data specified in the description of the **"start"** request in this documentation.

You can find information about the requirement of the EMV 3D Secure standard at the following link:

<https://www.emvco.com/emv-technologies/3-d-secure/>

What is EMV 3D Secure 2.0 service, and why is it necessary?

Convenient online purchases have become widespread in recent years, initiated even from mobile devices over the internet. Along with this, the number and volume of fraud, computer or internet abuse, data theft has increased.

Not only regulators but also card companies and banks are constantly working on new, more efficient solutions to ensure safe and reliable bank card payments. The previously available 3D Secure 1.0 only enabled the identification of the payer during online purchases initiated from a browser. This option was not available for purchases initiated from smart devices, and was only available for mobile phone payments when initiating a payment through a browser (not an application).

EMV 3D Secure 2.0 offers even more secure customer authentication and can be used not only for purchases made from browser-driven interfaces but also for in-app purchases and for payments over mobile phones and other smart devices. EMV 3D Secure 2.0 relies on customer authentication methods such as biometrics (such as fingerprints or face recognition) or one-time passwords. The transaction transfers more data to the issuing banks than it currently does, enabling the customer to be qualified more thoroughly, and to detect any fraud or abuse quickly and effectively.

What technical preparation does the implementation of 3D Secure 2.0 require?

The law requires that under 3D Secure 2.0, with a few exceptions, cardholders must be identified using strong customer authentication when initiating transactions, using bank cards issued within the European Economic Area, at payment acceptors operating within the European Economic Area. For proper authentication, both the card-issuing bank and the acquiring bank (or the payment acceptor) must use a 3D Secure solution (known as Acquirer Solution MPI or Merchant Plug-in in 3D Secure 1.0) that allows the buyer to be identified.

The card acceptance process via the internet, when developed and operated on the basis of this documentation, is the standard three-player model, which means that the cardholder will be redirected from the webstore interface to the SimplePay web interface provided by SimplePay Plc. The cardholder will enter the required card information on this payment page, the transaction will take place on this interface, and then the customer will be redirected to the webstore.

When using this payment method, the bank is responsible for the secure management and transmission of the bank card information (between the payment interface and the bank). No technical activities are needed for this on the merchant's side when the merchant sends the purchase information required for the 3D Secure process.

IV. Embedding the SimplePay payment page in an iFrame

Embedding the SimplePay payment page in an iFrame is no longer possible as a new request.

In the case of accounts on which this solution has been implemented and is actively used, the feature will remain available until it is completely removed.

You will find the "Enable iFrame" checkbox in the "General" panel on the "Account Management" / "Technical Data" page.

When this is activated, a text input field called "URL" appears. The domain name where the merchant's website operates and where the SimplePay payment page is embedded in the iFrame is to be entered in this field.

Note: modifications also need to be made to the SimplePay backend system for this function to work. Therefore, please notify our colleagues of your URL setting requirements at itsupport@simplepay.com.