DaoPay API 2.0 Integration Guide

Version: 1.5

Table of Contents

1. Overview

1.1. Integration Requirements

As soon as you complete the registration process on https://business.daopay.com/#register, our Sales Team will provide you with **appcode** and **secret key** which is used to sign all your payment requests. To finish your API configuration, we need to receive the following data:

- Website URL
- Payment Status Notifications (PSN) URL
- Return URL – for redirecting your clients after successful payments
- Failure URL – for redirecting your clients after failed payments
- Price point configuration: Before starting the integration, you should have discussed the price points/payment methods/countries you intend to use with our Sales team so they can set everything up while you work on your integration.

1.2. Payment Flow

In order to enable your customers to make payments through DaoPay, you need to implement our payment flow.

Please note: All communication to and from DaoPay should use a signature. For more information on the signature mechanism used, please look at Technical Information as well as Calculating the Signature.
The following steps happen every time a customer makes a payment:

1. Payment Request
First, you need to create a new transaction, which will be used by your customer to make his payment. To do this, you need to call the "create" - function (see create for more details) and specify the payment parameters you want to supply for the individual payment, such as price, country, etc. For a detailed list of the available payment parameters, please refer to Function Parameters - create.

2. Request Response (transactionid)
Once your payment request was processed, we will return a *transactionid* and *userurl* (or an error if there was a problem with your request), as detailed in Response Format - create and API Error Codes.

3. Forward Customer to DaoPay (userurl)
Once you have your userurl, you should forward your customer to our payment screen by redirecting him to the *userurl* you obtained in the last step:

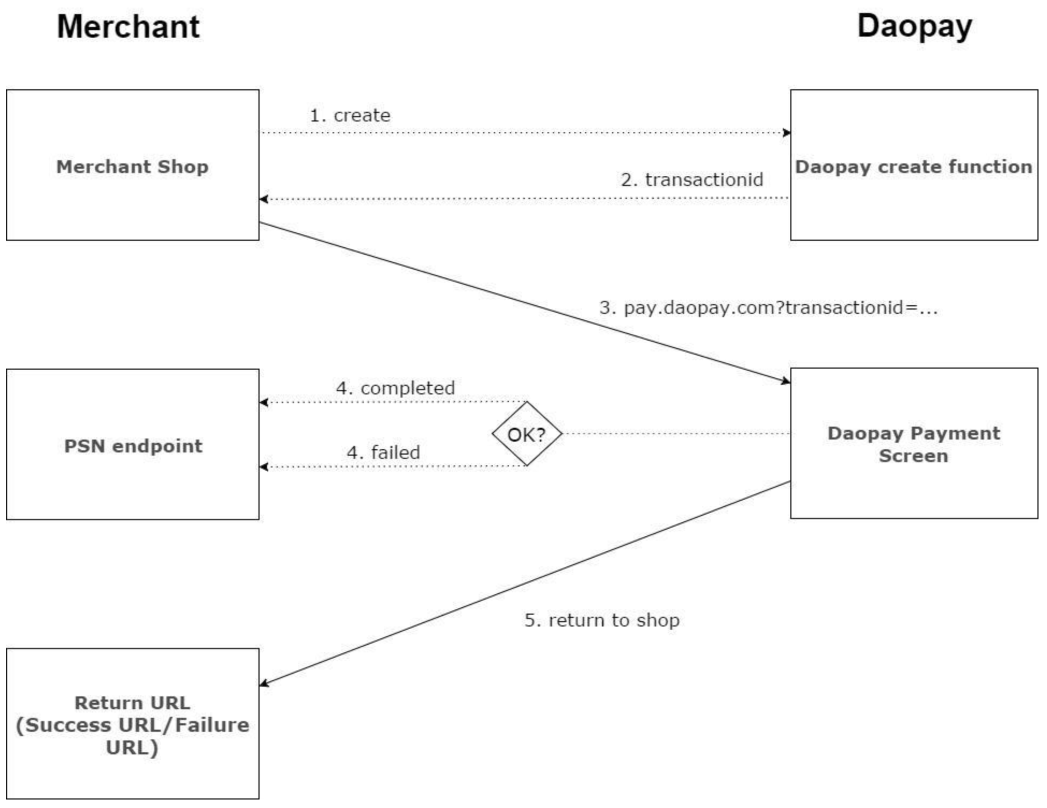
Your customer will now start his payment on our payment screen.
From the moment your customer lands on our payment screen until he finalizes his payment, we send Payment Status Notifications (PSN) to your Webhook (for more details, please refer to Transaction States as well as Payment Status Notifications (PSN)).

4. Return Page

The payment process can be finalized in 2 ways:

a) Your customer made a successful payment.
In this case, your customer will be taken to your *returnurl* (together with the DaoPay *transactionid* and the other parameters that you provided with your original request).

b) Your customer failed to make a successful payment.
For different reasons, your customer might not be able to successfully finalize his payment. This happens when your Customer's payment either fails, is aborted or expires. By default, we redirect all failed payments to the *returnurl*. In order to redirect failed payments to a different URL, please specify a *failureurl* with your request.



1.3. Price Jumping

This section is only relevant for Phone and SMS payments.
By default, only price points that match the desired amount of your request exactly will be matched by our system. Since this might be undesirable in some cases, we offer the "Price Jumping" feature.
Price Jumping can be set up independently for each application at our merchant portal (portal.daopay.com).

Please note: Price Jumping only applies when the exact price of your request couldn't be matched. In those cases (and assuming price jumping is enabled for your application), our system will try to match other prices. In case that pricejumping feature is used but no tolerance is specified, the default value of 10% will be assigned.

The parameters pricejumping and behavior are required to specify the pricejumping behavior. For more information on these parameters, please refer to Function Parameters - create.

Example: Using pricejumping=3 (jumping to both lower and higher price points) with an amount of 1 EUR and a tolerance of 20% would allow price jumping for all prices between 0.80-1.20 EUR.

1.4. Payment Methods

Payment methods describe the different types of payments that can be made through DaoPay. Currently, the following payment methods are available:

Phone and SMS payments

Name	ID	Supported Countries	Supported Currencies
SMS	1	see separate rate-card	
Pay-per-call (Drop Call)	2		
Pay-per-minute (PPM)	3		
Direct Carrier Billing (DCB)	4		

Online bank transfer

Name	ID	Supported Countries	Supported Currencies
Bancontact	29	BE	EUR
Blik	33	PL	PLN
EPS	15	AT	EUR
Estonian banks	16	EE	EUR
GiroPay	18	DE	EUR
iDEAL	19	NL	EUR
Klarna Sofort	28	NL, AT, BE, DE, ES, IT	EUR
Latvian banks	20	LV	EUR
Lithuanian banks	21	LT	EUR
Multibanco	22	PT	EUR
MyBank	23	IT	EUR
PayU	25	PL, CZ	PLN, CZK
POLi	26	AU, NZ	AUD, NZD
PostFinance	30	CH	CHF
Przelewy24	27	PL	PLN
Russian Banks	38	RU	RUB, EUR
Trustly	34	DE, DK, EE, ES, FI, GB, IT, LT, LV, NL, NO, PL, PT, SE, SK	EUR, DKK, GBP, NOK, PLN, SEK
Trustpay	35	CZ, SK	CZK, EUR
Verkkopankki	40	FI	EUR

Credit Card, SEPA, and offline bank transfer payments

Name	ID	Supported Countries	Supported Currencies
Credit Card	9	all	all
SEPA Direct Debit	10	AT, BE, BG, CY, CZ, DE, DK, EE, ES, FI, FR, GR, HR, HU, IE, IS, IT, LI, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM	EUR
Bank Transfer (offline)	13	SEPA-countries	EUR

e-Wallets and Vouchers

Name	ID	Supported Countries	Supported Currencies
Qiwi	32	RU, KZ, UA	EUR
PaySera	24	SEPA-countries	EUR
PaySafeCard	11	all except US	EUR, USD, GBP, CHF
PayPal	31	all	all
Yandex	39	all	RUB, EUR

Payment Groups for Phone and SMS payments

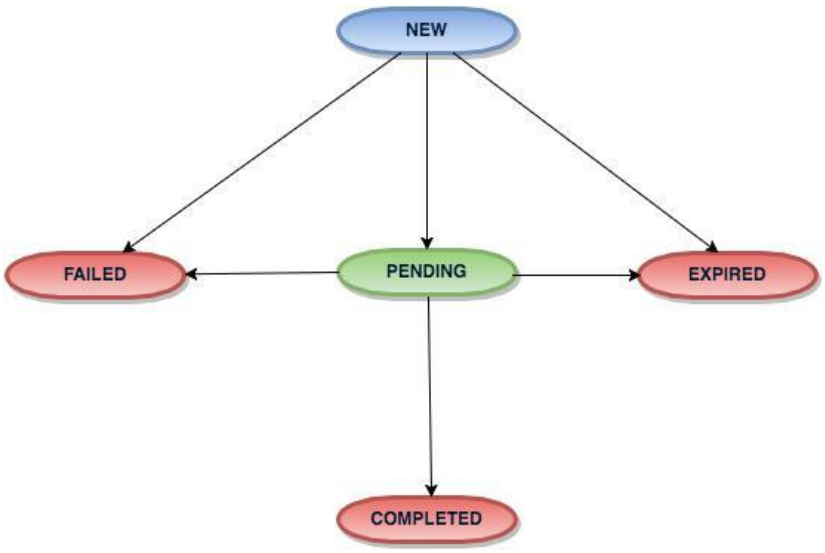
Sometimes, you want to allow several payment methods for a payment request.

In those cases, you can use a payment group ID instead of an individual payment method ID for the *paymentmethod* parameter. Here is a list of the currently available payment groups:

Payment group ID	Name	Description
1002	Voice group	This group includes all voice payment methods (Drop Call, Pay-Per-Minute)
1003	Mobile group	This group includes all payment methods that can be paid with a mobile phone (SMS, Direct Carrier Billing)

1.5. Transaction States

A transaction can be in one of the following states at a specific point in time. For each of these states (except for the state NEW) a PSN is sent out by DaoPay to a webhook specified by the merchant.



State descriptions

State name	Description
NEW	The initial state of any transaction. It remains active until the start of the billing phase or until it fails or expires
PENDING	With the start of the billing phase (e.g. waiting for user to pay, waiting for TAN, etc...), the PENDING state is entered
FAILED	This state indicates that the transaction failed for a reason (e.g. MSISDN blocked, Server not reachable, spending limit reached, Transaction not possible...), or was aborted by the user
EXPIRED	Transactions that don't complete successfully or fail in time (see note above) expire and can no longer be completed or fail
COMPLETED	After a transaction is fully paid, it enters the COMPLETED state

Please note: We send notifications in the order they occur. In some rare cases, you might receive them in the wrong order due to network delays, etc. Therefore, please make sure that you ignore any PENDING notifications after you received the corresponding COMPLETED or EXPIRED notification.

Transaction expiration dates

Transactions that don't get finished within a certain time frame expire automatically. The exact duration for expiration depends on the payment method (see full list of Payment Methods) and the transaction state.

Cases (Payment Method ID)	NEW	PENDING
Default	1 hour	24 hours
Voice payments (2, 3)	24 hours	14 days
Great Britain (4)	1 hour	72 hours

1.6. Content types

The *contenttype* describes the type of content that gets sold over a particular transaction. Depending on the *contenttype*, certain price points may or may not be available.

Possible values for the parameter *contenttype*:

***contenttype* value**	**Name**	**Description**
1	**Online games**	all online game providers/publishers
2	**Virtual currency**	all merchants that sell credits, gold and other virtual currencies (not directly implemented in a game)
3	**Social media / Social networks**	all social networks (with a closed virtual currency structure)
4	**Media / Digital content**	all music downloads, video downloads and newspapers
5	**File sharing services**	all webhosting, filesharing and software download providers
6	**Dating**	all online dating services without adult content
7	**Physical goods**	all kind of physical goods
8	**In-App Payments**	all kind of in-app payments
9	**Ticketing**	all kind of ticket purchases

1.7. Targetgroups

The *targetgroup* describes the group which is expected to use the Service.

Possible values for the parameter ***targetgroup*:**

***targetgroup* value**	**Name**	**Description**
2	minors	products specifically advertised to children
3	adults	Any services where the target group is 18+ (gambling...)

2. Technical Information

HTTP GET

All communication to and from DaoPay is done via the HTTP(S) protocol. As a parameter passing method, we use only HTTP(S) GET-Requests.

SSL

All communication between your server and DaoPay has to be done via HTTPS (SSL).

IP Restrictions

For our Payment Status Notifications (PSN) we recommend verifying the IP address from which your Webhook is called. For more information on PSNs and Webhooks, please refer to Payment Status Notifications (PSN).

We call your Webhook exclusively from the following IP-addresses:

IP address
195.58.177.2
195.58.177.3
195.58.177.4
195.58.177.5

We recommend accepting payment information that comes only from the above IP-addresses. It is strongly recommended that you provide the IP addresses of your servers to the DaoPay Integration Team. In that case, DaoPay will only accept requests from your IP addresses.

Please note: Apache and other web servers may filter authorization headers from requests by default, making it impossible to verify the authorization. If this is the case, please review your web server configuration (for more details on the verification of the authorization - header, please refer to Receiving Requests).

2.1. Visual and Layout

DaoPay Logos

We offer you graphics containing payment methods logos in .png format, which you can use as payment buttons on your web site. For all payment methods offered by DaoPay please use the logos available under the following URL:

```
https://daopay.com/logos/
```

Implementing DaoPay

This section describes two basic ways to implement DaoPay into your product.

We assume that you have a working payment-URL which you want your customer to use to make his payment (Please refer to Payment Flow if you don't have a payment-URL yet).

...as a Link

The first option consists of adding a link to your website. Once your customer clicks on this link, he will be redirected to a website on DaoPay's side, (ideally) finishes his payment successfully and gets redirected to your returnurl.

```
<a href="https://pay.daopay.com/?transactionid=<transactionid>">Pay
with DaoPay</a>
```

Clicking the link will start the payment process. Be sure you replace "<transactionid>" with your transactionid (please refer to create for more information on how to request a transactionid).

...as an iFrame

Our experience shows that you can achieve higher conversion rates (and thus higher payouts) when your customer stays on your web page during the entire payment process. We therefore strongly recommend this method. After your customer has chosen his desired payment method and product, you embed the payment screen on your own web page using an iframe:

```
<iframe src="https://pay.daopay.com/?
transactionid=<transactionid>" width="530" height="350"></iframe>
```

Please note: Due to current technical restrictions, we recommend not using iFrames for mobile applications or services using the 3G (mobile) Flow.

Custom styles for Phone and SMS Payments

Depending on the graphic design of your website, you might want to use a custom visual style for the DaoPay payment screen using the *customstyleid* parameter. Please refer to Functions to see which DaoPay functions support custom styles. Here is a list of the currently available custom styles:

customstyleid	description	background color
1	Default style – fixed size with border-radius	grey
2	Default style – flexible 100% width, no rounded edges	transparent
666	Dark style – fixed size	grey

Please note: If you need a personalized style apart from the styles above, feel free to get in touch with us.

2.2. Sending Requests

In order to send requests, the following steps are required:

1. Add Timestamp and create Request Signature
For more details on this step, please refer to Calculating the Signature and PHP Example.

2. Add Authentication Information to your Request
We use the standard HTTP Authorization header for the authentication of all HTTP requests sent to and received by us.

Please add the following to your request header:

```
Authorization:<signature>
```

3. Send Request and verify Response
Your request can now be sent. If the authentication of your request was successful, we will respond to it with the HTTP status code 200 - OK.

If you don't receive this status code, please compare the received code to the following table to find the error that occurred:

HTTP status code	Description
400	Bad Request - Parameter 'appcode' is missing, null or invalid
400	Bad Request - The timestamp of your request is invalid. Please generate a new request with a current timestamp
401	Unauthorized - The signature verification failed. Please make sure that you calculate the signature correctly
403	Forbidden - The IP address that made the request is not allowed for your application.

2.3. Receiving Requests

When receiving a request, it is important that you verify its signature as well as make additional checks to make sure that the request is valid:

1. Verify that the Timestamp is not older than 15 minutes
For the request to be valid, it must include a "requesttimestamp" parameter (see Calculating the Signature for more details).

Should the value of this parameter be older than 15 minutes, please discard the request and respond with the HTTP status code "**400 - Bad Request**".

2. Verify the IP address from which the request was sent
We send requests from the following IP addresses only:

195.58.177.2 | 195.58.177.3 | 195.58.177.4 | 195.58.177.5

If the request was sent from an IP address different from the above, please discard the request and respond with the HTTP status code "**403 -Forbidden**".

3. Read the signature from the request header
The signature of the request is located in its header in the following form:

```
Authorization: <signature>
```

Please note: In some cases, web servers may strip the Authorization header from incoming requests. In case of Apache, this can be solved by adding the following lines to your .htaccess-file:

```
RewriteEngine On
RewriteCond %{HTTP:Authorization} ^(.*)
RewriteRule .* - [e=HTTP_AUTHORIZATION:%1]
```

4. Calculate and compare the request signature

Compare the request signature to the calculated signature. If they match, the authentication was successful. In this case, please respond to the request with the HTTP status code "**200 - OK**".

Should the signatures mismatch, please discard the request and respond with the HTTP status code "**401 - Unauthorized**"

2.4. Calculating the Signature

In order to calculate the signature, the following data is required:

- A secret key (this can be obtained from our Sales or Integration Team)
- A list of request parameters representing the payment information
- A timestamp that represents the current time (in the Unix time format of seconds since epoch) together with a millisecond value.

Please note: We append the current milliseconds to our timestamps. The value of the timestamp *1397564362*, together with a millisecond value of *123*, would therefore be *1397564362123*.

The signature can be calculated by executing the following steps:

1. Create the query string with urlencoded values
2. Append the timestamp to your concatenated request parameters, for example "... &requesttimestamp=1397564395".
3. Calculate the hash-value (HMAC-SHA512) of the string from last step in combination with your secret key
4. Base64-encode the resulting hash-value
5. Finally, urlencode the string

Please note: The order of the parameters used to calculate the signature must match the order of the parameters in your request. Otherwise, the signature verification will fail!

2.5.1. Request Signing Example

Let's assume you want to sign the following request parameters:

Key	Value
App code	*12345*
Price	*1.0*
Product	*über*

For this example, we will be using the following timestamp and secret key values:

Key	Value
requesttimestamp	1397564362123
secret key	*123*

1. Create the query string with urlencoded values

```
appcode=12345&price=1.0&product=%C3%BCber
```

2. Append the timestamp

```
appcode=12345&price=1.0&product=%C3%BCber
&requesttimestamp=1397564362123
```

3. Calculate hash value (HMAC-SHA512) using the previous string and your secret key

```
<raw binary data>
```

4. Base64-encode the hash value

```
RSxrnBWYHlyGkZpDW4fsu+kHNtiqloyd96ew2Qg4HJTbOSHmGJohqpD
/+bsPOk1jaeMhcR43nnlPcAL/CZpFAg
```

5. Url-encode the string to obtain the final signature of your request

```
RSxrnBWYHlyGkZpDW4fsu%2BkHNtiqloyd96ew2Qg4HJTbOSHmGJohqpD%2F
% 2BbsPOk1jaeMhcR43nnlPcAL%2FCZpFAg
```

Please note: The final urlencode should encode with upper case characters, e.g. "%2F" and not "%2f". Most implementations of urlencode do this automatically (currently, .NET is an exception). If you encounter issues with the signature, please make sure that you are using the correct case!

2.5.2. PHP Example

Creating Signature PHP

```
/*
   Input parameters:
   $parameters: An array of key-value pairs that contains
all payment parameters
   $timestamp: The current Unix time (seconds) + the
current milliseconds (i.e. 1411370116036)
   $secretkey: Your secret key
*/

function computeApiSignature($parameters, $timestamp,
$secretkey) {
    $timestamp = number_format($timestamp, 0, '', '');

    // Build a Query String
    foreach($parameters as $key => $value) {
      $queryString .= "$key=" . urlencode($value) . "&";
    }

    // Append timestamp
    $queryString .= "requesttimestamp=" . $timestamp;

    // hash_hmac, output in raw-format=true
    $hmac = hash_hmac('sha512', $queryString, $secretkey, true);

    // Base64 encode
    $base64 = base64_encode($hmac);

    // Urlencode
    $final = urlencode($base64);

    return $final;
}
```

2.5.3. Java Example

Create Signature Java

```
/*
   Input parameters:
   String requestParameters: A list of payment parameters in
HTTP GET-form ("appcode=123&price=1.0&product=test")
   String secretKey: Your secret key
*/

// Prepare timestamp
String timestamp = String.valueOf(System.currentTimeMillis());

// Append timestamp to requestUrl
final String payload = requestParameters + "&requesttimestamp="
+ timestamp;

// Generate HMAC
byte[] hmac = getHmac(secretKey, payload, "HmacSHA512");

// urlencode signature
final String encodedSignature = URLEncoder.encode
(DatatypeConverter.printBase64Binary(hmac), "UTF-8");
```

Generate HMAC

```
private static byte[] getHmac(String secretKey, String
payload, String hmacType) throws UnsupportedEncodingException,
NoSuchAlgorithmException, InvalidKeyException {
   final Mac mac;
   byte[] hmac;
   final byte[] secretKeyBytes;

   if (secretKey == null || secretKey.trim().isEmpty())
       { secretKeyBytes = new byte[]{0};
   } else {
       secretKeyBytes = secretKey.getBytes("UTF-8");
   }

   SecretKeySpec keySpec = new
SecretKeySpec(secretKeyBytes, hmacType);
   mac = Mac.getInstance(hmacType);
   mac.init(keySpec);
   hmac = mac.doFinal(payload.getBytes("UTF-8"));

   return hmac;
}
```

3. Functions

We provide various ways to handle your payments.

Following you can find a list of all functions that are currently available.

3.1. Create

The *create* function can be used to create a *transactionid* for a particular price. This *transactionid* can then be used to send the buyer to the payment screen.

The *create* function can be called through the following endpoint:

```
https://api.daopay.com/v1.2/create
```

3.1.1. Function Create – default parameters

You can make requests by calling our endpoint and providing your payment parameters, for example:

```
https://api.daopay.com/v1.2/create?appcode=<your appcode>
```

The table with all available payment parameters can be found below. Some payment methods require usage of additional parameters, which are listed in separate tables.

Default parameters

Parameter name	Format	Description	Required
appcode	1234 Integer	The appcode is used to uniquely identify your application.	yes
amount	1.00 Float number	The price you want to bill your customer.	yes
currency	EUR 3 chars	The currency of the price that should be billed. Please refer to ISO 4217 for more information.	yes
countrycode	AT or 89.12.1.34 2 chars or IPv4	This parameter can be used to preselect the country in which your customer is located. Please refer to ISO_3166-1 for details on the valid country codes. You may also pass an IPv4 address. In that case, the IP will be resolved to a country by our IP lookup functionality.	yes

Default parameters (continue)

Parameter name	Format	Description	Required
productname	*"100 Gold Coins"* String	A description of the product you want to bill. **Please note:** Due to layout reasons, your *productname* should not be longer than 80 characters.	yes
productdescription	"*Gold Coins are used to buy ingame items from the ingame shop...*" String	A more detailed description of the product.	no
customtransactionid	*<transactionid>* String	A unique ID for transactions within your system /database.	yes
userid	*<userid>* String	A unique ID that can be used to identify your individual customers/users.	yes
contenttype	1 Integer	This identifies the type of product that should be billed ("gaming",..). Please refer to Contenttypes for more details and a list of all possible values. **Please note:** This parameter is only required if the product you want to sell differs from your default content type, or if you are an aggregator and aggregate different content types.	yes (for Aggregators only)
submerchantid	*<submerchantid>* String	If you are an aggregator, online gaming plattform or if you sell virtual currency to different merchants, please use this unique ID to identify your individual merchants or games.	yes (for Aggregators only)
paymentmethod	*1* Integer	This parameter specifies the payment method you want to use. Please refer to Payment Methods for more details and a list of possible values.	yes
language	*DE* 2 chars	The language that should be displayed to your customer (in case where payment method allows selection). Please refer to ISO 639-1 for possible values	no

Default parameters (continue)

Parameter name	Format	Description	Required
targetgroup	2 Integer	If service has a specific target group please specify it using this parameter. Please refer to Targetgroups for more details and a list of possible values. **Please note:** This parameter is only required if the target group differs from the default target group.	no
returntarget	*_self* String	Specifies how your customer will be returned to the *returnurl* or *failureurl*: _*self*: Opens the URL inside the payment screen frame. _*parent*: Opens the URL inside the parent frame. _*blank*: Opens the URL in a new window or tab. _*top*: Opens the URL in the full body of the window.	no
returnurl	*http://www.site.com/success.html* URL	If specified, this parameter overrides the default URL to which your customer is redirected after a payment. If you don't have a *failureurl* set, failed payments will also be redirected to your *returnurl*.	no
failureurl	*http://www.site.com/failure.html* URL	If specified, this parameter overrides the default URL to which your customer is redirected after a failed/expired/aborted payment.	no
psnurl	*http://www.site.com/psn.html* URL	If specified, this parameter overrides the default URL to which PSNs for this payment are sent. For more information on PSNs, please refer to Payment Status Notifications (PSN)	no
customparameter	*param1:value1, param2:value2* String	You can use this parameter to add any number of custom parameters. The maximum length of all your parameters and values should not exceed 255 characters. **Please note:** The individual parameters and values need to be combined as shown in the format column.	no

3.1.2. Additional parameters used only for Phone and SMS payments

Parameter name	Format	Description	Required
operatorid	*"1223354"* String	This parameter identifies a single operator the buyer is using. If direct billing is not available for all operators in the country you intend to use, this parameter is required.	no
callerid	*+4312345678* String	If you already know the telephone number of your customer, you can provide it using this parameter. This allows us to verify if the number is allowed to make payments. Please refer to E.164 regarding the required format.	no
pricejumping	1|2|3 Integer	Describes the desired pricejumping behavior: 1: Jump to the nearest **higher** available amount 2: Jump to the nearest **lower** available amount 3: Jump to the nearest amount (lower **or** higher)	no
pricetolerance	*20* Percentage (0-100)	The percentage (of the amount) specifies the interval within which price jumping is possible. If it's not specified, the default value of 10% will be used.	no
customstyleid	7 Integer	If specified, this parameter overrides the default style for this payment. For more information on available styles, please refer to Visual and Layout.	no

3.1.3. Root payment for online bank transfers

Some of the online banking payment schemes support **root payment** which allows to send customers directly from your checkout page to their e-banking login page bypassing the intermediary “choose your bank” page. In order to use this functionality your need to send an additional parameter in your payment request which serves as bank identifier.

Parameter name / Payment method	Format	Description	Required
bankid to be used with **Russian Banks**	String with 3 possible values: "*sberbank*" "*alfabank*" "*tinkoff*"	This parameter specifies Russian bank where a customer gets redirected to, in order to complete the online payment	yes
bankid to be used with **iDEAL**	String with BIC of underlying bank, see list of possible values below	This parameter specifies bank in the Netherlands where a customer gets redirected to, in order to complete the online payment	no
bankid to be used with **EPS**	String with BIC of underlying bank, see list of possible values below	This parameter specifies bank in Austria where a customer gets redirected to, in order to complete the online payment	no

iDEAL (NL)	
Bank name	**bankid**
ABN AMRO	ABNANL2A
ASN Bank	ASNBNL21
Bunq Bank	BUNQNL2A
Handelsbanken	HANDNL2A
ING	INGBNL2A
Knab Bank	KNABNL2H
Moneyou	MOYONL21
Rabobank	RABONL2U
Regiobank	RBRBNL21
SNS Bank	SNSBNL2A
Triodos Bank	TRIONL2U
Van Lanschot	FVLBNL22

EPS (AT)	
Bank name	**bankid**
Raiffeisen Bank	RLNWATWW
Bank of Austria	BKAUATWW
Erste Bank	GIBAATWW
BAWAG P.S.K.	BAWAATWW
Oberbank	OBKLAT2L
Denizbank	ESBKATWW
BTV Bank	BTVAAT22
BKS Bank	BFKKAT2K
Anadi Bank	HAABAT2K
VKB Bank	VKBLAT2LXXX
Dolomiten Bank	OVLIAT21XXX
Marchfelder Bank	MVOGAT22XXX
BANK99	SPBAATWWXXX

3.1.4. Additional mandatory parameters for specific Payment types

Several payment methods require to use additional mandatory parameters for initiating the transaction, which can be found in the table below.

Parameter name / Payment method	Format	Description	Required
accountholder to be used with **Trustly**	*John Smith* String	First name and last name of bank account holder who is paying via Trustly. This payment scheme requires it to be sent with every payment request	yes
callerid to be used with **Qiwi**	*+4312345678* String	Country code + mobile phone number of customer connected to Qiwi wallet. It can be: Russia: +7 and 10-digits number Ukraine: +380 and 9-digits number Kazakhstan: +7 and 10-digits number	yes

Please note: All API requests need to be properly signed to be accepted (for more details, please refer to Calculating the Signature).

3.1.2. Response Format – create

Once we processed your transaction request, we will respond with a *transactionid* that uniquely describes your requested transaction as well as the *userurl*, which is the URL that you should send your Buyer to after creating his transaction:

Success response example

```
{
    "request": {
        "responsetimestamp": "2015-01-30T18:00:00.000+02:00",
        "apiversion": "0.2",
        "requesturl": "https://api.daopay.com/v1.2/create?
appcode=99999&amount=10&currency=EUR&countrycode=DE&.....",
        "parameters": {
            "appcode": 99999,
            "currency": "EUR",
            "amount": 10,
            "countrycode": "DE",
            ...
     }
},
    "transactions": {
    "transactionid": "024b3cf8-9f51-4042-808b-609f68a8c656",
"userurl": "https://pay.daopay.com/?transactionid=024b3cf8-
9f51-4042-808b-609f68a8c656"
    }
}
```

Please note: If your request couldn't be successfully processed by our system, we will return an error response, which contains information on the error source by describing an error code (for a full list of these error codes, please refer to API Error Codes):

4. Payment Status Notifications (PSN)

We will notify you in real time about the status of an ongoing payment or the status of a subscription by using Payment Status Notifications (PSN). Each state change results in a new PSN.

For more information on transaction and subscription states, please refer to Transaction States.

It does so by calling a Webhook that you provide (Webhooks are user-defined callbacks over HTTP, for more information on them, please refer to http://webhooks.org). The Webhook is used to receive *PSNs*, sometimes also called *Status Callbacks* or *Instant Payment Notifications (IPN)*.

Our Payment Status Notification system is a similar mechanism to what PayPal and Amazon call "Instant Payment Notifications" and which Google Checkout calls "Notification Callbacks". It's a custom program (or CGI, servlet, etc) that you implement in your preferred programming language and that we call through an HTTP GET request to inform you in real time when the status of a payment changes. A status change can be an SMS that has been received, or a payment that has been completed. You can then use these Payment Status Notifications to update your customer accounts, generate a serial number, or to do other things necessary in order to automatically process payments.

Note that the Status Notification Webhook is called in the background and does not have a user interface. You may use your return-URL to display information to the user after his payment. Based on our PSN you can reward the buyer with his purchase as soon as his transaction enters the state COMPLETED.

All PSN calls sent from us are signed. Please make sure to verify the signature of the PSN and to ignore it in case that the signature doesn't match (For more details, please refer to Receiving Requests).

Our Integration Team will configure the "default" URL of your Webhook based on your input.

Please note: We recommend only using HTTPS as a protocol to access your Webhook. Additionally, only allow your Webhook to be accessed from the IP-addresses defined in Technical Information.

4.1. Function Parameters (one-time payments)

As soon as a new transactionid is generated, you should set the transaction's status to NEW.

Please note: Our system does not send a notification for this initial status.

The function parameters depend on the type of PSN. Below you can find the parameters used for the different types of PSNs.

4.1.1 PENDING

Parameter name	Value Format	Description
appcode	*99999* Integer	The appcode is used to identify your application
transactionid	*fa6a8417-321d-4fea-851f-ab182d35cc70* Alphanumeric chars [a-z0-9] - 36 chars long	A unique ID used to identify the current payment
customtransactionid	*<your-transactionid>* String	The custom transactionid you provided when creating this transaction. This can be your defined format type, like a specific hash string. It can serve as your Order ID.
status	*PENDING* String	The state of your payment
statusdescription	*"The payment is in progress."* String	A more detailed description of the status
customparameter	*<your custom-parameter>* String	Your custom parameters you provided when creating this transaction.
resendcount	*1* Integer	Displays the amount of times this PSN was sent out. This parameter is only added if the original PSN call failed at least once.
userid	*user123* String	The userid of your customer. This is the same ID you provided when creating the payment.
requesttimestamp	*1397564362123*	This parameter is provided with every request. It is needed to verify the signature of the request (for more details, please refer to Calculating the Signature)

4.1.2 COMPLETED

Parameter name	Value Format	Description
appcode	*99999* Integer	The appcode is used to identify your application
transactionid	*fa6a8417-321d-4fea-851f-ab182d35cc70* Alphanumeric chars [a-z0-9] - 36 chars long	A unique ID used to identify the current payment
customtransactionid	*<your-transactionid>* String	The custom transactionid you provided when creating this transaction. This can be your defined format type, like a specific hash string. It can serve as your Order ID.
status	*COMPLETED* String	The state of your payment
statusdescription	*"The payment was completed successfully."* String	A more detailed description of the status
substatus	*1000* Integer	The code of the exact substatus. Possible substatus values: *1000*...Successful payment *1001*...Compensation
paidamount	*24.44* Float number	The amount paid by your client.
payout	*18.97* Float number	The payout that you will receive for this transaction.
currency	*EUR* 3 chars	The currency of the paidamount and payout. Please refer to ISO 4217 for more information.
countrycode	*DE* 2 chars	Countrycode of payment
callerid	*4915732789123* Integer	Your client's telephone number (only for Phone and SMS Payments)
hashedcallerid	*ba310178e29a040eef6ee86d42d9a2* 30 chars	Hashed value of your client´s telephone number

Parameter name	Value Format	Description
paidtime	*2015-01-30T18:00:00.000+02:00* YYYY-MM-DDThh:mm:ss.sssTZD	This parameter describes the date and time on which the transaction was paid completely. For more details, please refer to ISO 8601. Please make sure to specify the time zone designator as well as 3 digits for the fractions of second.
customparameter	*<your custom-parameter string>* String	Your custom parameters you provided when creating this transaction.
operatorid	*DE-E-Plus* String	DaoPay's internal operator name (countrycode + operatorname). Only provided for Phone and SMS payments
resendcount	*1* Integer	Displays the amount of times this PSN was sent out. This parameter is only added if the original PSN call failed at least once.
userid	*user123* String	The userid of your customer. This is the same ID you provided when creating the payment.
vatcollected	1.40 Float number	The amount of VAT that still needs to be paid to the country that the payment originated from. **Please note**: In this case you need to handle the payment of VAT yourself. Only relevant for Phone and SMS payments
vatcleared	1.40 Float number	The amount of VAT that was already paid in the country of the payment for this transaction. No further VAT needs to be paid in this case. Only relevant for Phone and SMS payments
requesttimestamp	*1397564362123*	This parameter is provided with every request. It is needed to verify the signature of the request (for more details, please refer to Calculating the Signature)

4.1.3. FAILED

Parameter name	Value Format	Description
appcode	*99999* Integer	The appcode is used to identify your application
transactionid	*fa6a8417-321d-4fea-851f-ab182d35cc70* Alphanumeric chars [a-z0-9] - 36 chars long	A unique ID used to identify the current payment
customtransactionid	*<your-transactionid>* String	The custom transactionid you provided when creating this transaction. This can be your defined format type, like a specific hash string. It can serve as your Order ID.
status	*FAILED* String	The state of your payment
statusdescription	*"The user is blocked by operator."* String	A more detailed description of the status
substatus	*3000* Integer	The code used to identify the type of error.
customparameter	*<your custom-parameter string>* String	Your custom parameters you provided when creating this transaction.
resendcount	*1* Integer	Displays the amount of times this PSN was sent out. This parameter is only added if the original PSN call failed at least once.
userid	*user123* String	The userid of your customer. This is the same ID you provided when creating the payment.
requesttimestamp	*1397564362123*	This parameter is provided with every request. It is needed to verify the signature of the request (for more details, please refer to Calculating the Signature)

4.1.4. EXPIRED

Parameter name	Value Format	Description
appcode	*99999* Integer	The appcode is used to identify your application
transactionid	*fa6a8417-321d-4fea-851f-ab182d35cc70* Alphanumeric chars [a-z0-9] - 36 chars long	A unique ID used to identify the current payment
customtransactionid	*<your-transactionid>* String	The custom transactionid you provided when creating this transaction. This can be your defined format type, like a specific hash string. It can serve as your Order ID.
status	*EXPIRED* String	The state of your payment
statusdescription	*"The payment expired due to inactivity."* String	A more detailed description of the status
customparameter	*<your custom-parameter string>* String	Your custom parameters you provided when creating this transaction.
resendcount	*1* Integer	Displays the amount of times this PSN was sent out. This parameter is only added if the original PSN call failed at least once.
userid	*user123* String	The userid of your customer. This is the same ID you provided when creating the payment.
requesttimestamp	*1397564362123*	This parameter is provided with every request. It is needed to verify the signature of the request (for more details, please refer to Calculating the Signature)

4.1.5. Status-description parameter values

Possible *statusdescriptions* for the different statuses:

Status code	**Description**
1000	*"Successful payment"*
1001	*"Successful compensation"*
2004	*"Invalid PIN entered repeatedly"*
2005	*"The payment could not be billed due to insufficient funds."*
2006	*"The user is blocked by his operator"*
2007	*"Invalid CLI entered repeatedly"*
2998	*"The payment failed"* (ERROR DAOPAY INTERNAL UNKNOWN)
2999	*"Error reaching the user's operator"* (ERROR INTERNAL UNKNOWN)
3000	*"The payment was aborted by the user"*

Please note: Applications in test mode always send out the substatus 1000. In that case, please read the *statusdescription* to find out the PSN status.

4.2. Response Codes PSN

The following table describes the specific Response / Error codes we expect for the PSN function call. These are HTTP status response codes.

HTTP status code	Decription	Notes
200	OK	Use this code for valid requests
400	BAD REQUEST - requesttimestamp missing	Use this code if the request doesn't contain a requesttimestamp
400	BAD REQUEST - requesttimestamp invalid	Use this code if the requesttimestamp is invalid (for more details, please refer to Receiving Requests)
400	BAD REQUEST - unknown status	Use this code if no status was passed to your PSN or if the status value isn't a known status
401	UNAUTHORIZED authentication header missing	Use this code the request doesn't contain an authorization header
401	UNAUTHORIZED signature mismatch	Use this code if the signature you calculated doesn't match the signature provided with the request (for more details, please refer to Calculating the Signature)
403	FORBIDDEN	Use this code if the PSN was called from an unknown IP address

We will also store any other HTTP status, but please ensure to map the above ones correctly to return them in the specified cases.

Please note: If we receive any HTTP status code other than 200 as a response from your Webhook, we will resend the PSN in question. Please contact our team if you would like us to not resend PSNs for other HTTP status codes.

5. Additional Information

5.1. Supported Country Codes

For country codes, we use the 2-letter ISO code (ISO 3166 alpha-2). For more information on this standard, please refer to http://www.iso.org/iso/country_codes.

5.2. Supported Currency Codes

For currency codes, we use the ISO 4217 standard. For more information on this standard, please refer to http://www.iso.org/iso/home/standards/currency_codes.htm. For a list of the possible values, please refer to http://www.xe.com/iso4217.php.

5.3. API Error Codes

This is a list of the currently used error codes. Additional error codes might be added in the future.

Error code	Decription
1005	Parameter '*countrycode*' is missing, null or invalid. Must be 2 characters according to ISO 3166-1 alpha-2 codes or an IPv4 conforming value
1006	Value of parameter '*language*' is invalid. Must be 2 characters according to ISO 639-1
1009	Parameter '*amount*' is missing, null or invalid. Must be a positive double or float with two decimals places and a dot '.' as decimal point.
1010	Parameter '*currency*' is missing, null or invalid. Must be 3 characters according to ISO 4217
1011	Parameter '*productname*' is missing or null.
1013	Parameter '*customtransactionid*' is missing or null.
1014	Parameter '*userid*' is missing or null.
1015	Parameter '*contenttype*' is missing, null or invalid.
1016	Parameter '*targetgroup*' is invalid.
1017	Value of parameter '*paymentmethod*' is invalid.
1018	Value of parameter '*pricejumping*' is invalid.

Error code	Decription
1019	Value of parameter '*pricetolerance*' is invalid.
1022	Parameter '*operatorid*' is null or invalid.
1090	Parameter '*submerchantid*' is missing, null or invalid.
1103	Value of parameter '*customstyleid*' is not an existing customstyleid.
1111	This '*callerid*' is blocked.
1112	This '*callerid*' doesn't have enough spending limit left.
1113	This '*userIP*' is blocked.
1114	UserIP detected country and transaction country mismatch.
1115	Msisdn verification failed.
1120 **1121** **1122** **1123**	The requested pricepoint is not available.
1036	Parameter '*returnurl*' is invalid.
1037	Parameter '*failureurl*' is invalid.
1038	Parameter '*psnurl*' is invalid.
1201	Parameter '*subscriptionid*' is missing, null or invalid.
1202	Parameter '*billingperiod*' is missing, null or invalid.
1203	Parameter '*billingfrequency*' is missing, null or invalid.
1204	Parameter '*stop*' is invalid.
1205	Parameter '*subscriptionname*' is missing, null or invalid.
1206	Parameter '*customsubscriptionid*' is missing, null or invalid.
1208	Parameter '*creditexchangerate*' is missing, null or invalid.
1209	Parameter '*creditsname*' is missing, null or invalid.
1301	Invalid *transactionid*
1302	This transactionid was already confirmed.

6. Clearing API

6.1. Introduction Clearing API

This manual explains the payment process, with a special focus on transactions where money cannot be collected from a Buyer (Chargebacks).

If you are a **payments manager** or work in the **finances** department of your company, this section will help you understand how and why chargebacks occur and how you can handle them.

If you are a **software engineer**, this guide contains technical information on how you can get notified of chargebacks and how you can use these notifications to update a user's account and reverse transactions, if required.

6.2. Regular Money Flow for Phone and SMS payments

To understand Chargebacks and other risks associated with payments, it is helpful to gain insight into how funds are transferred from a Buyer to a Seller when using DaoPay.

DaoPay maintains a worldwide network of relationships with phone operators in different countries, who in turn maintain relationships with their customers. After a Buyer has made a payment with DaoPay, the respective operator bills and collects funds from them (in local currency). Once an operator has received funds from the Buyer, he forwards them to DaoPay.

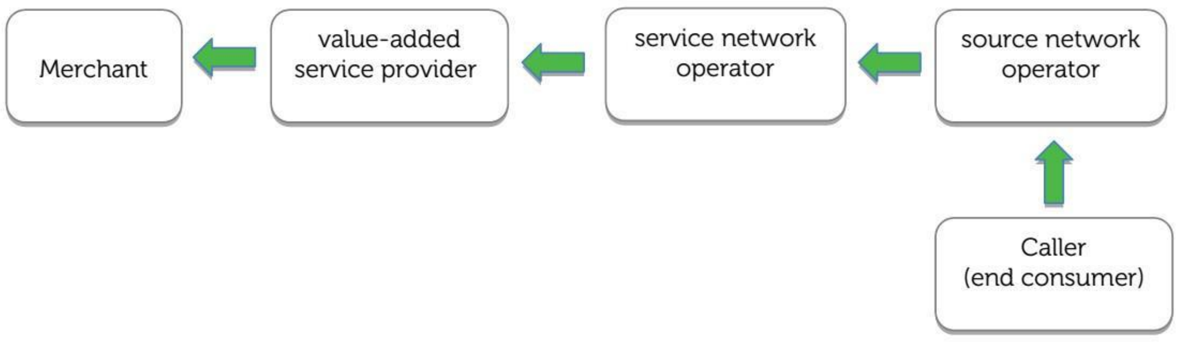


Figure 1: Regular Money Flow from Buyer (right) to Seller (left) – Phone & SMS Payments

6.3. Chargebacks

As always, there are a few exceptions to the rule. Some countries enforce that consumers can reverse or withhold payments after their purchase; in these countries, it may occur that a buyer does purchase an item, but will then not pay his or her phone bill. This is called a **Chargeback**. Chargebacks are not specific to just phone payments, but occur with other payment methods as well, especially with credit cards or checks.

6.3.1. How Chargebacks are generated

Not every chargeback is fraudulent, though; Buyers may also generate Chargebacks when

- There was a delay with the payment
- Their mobile phone was stolen and then used for online payments
- A home phone was used without the owner’s permission
- Children did not ask for their parents’ permission, spent “too much” with their phone, and parents refuse to pay the bill
- They did not receive the items they purchased

6.3.2. Why Chargebacks matter

Chargebacks matter because you have probably delivered items or performed services for a customer but are not receiving money for your sale. Depending on what you sell, you may be able to reverse the transaction, i.e., if you sell virtual currency, you might want to choose to deduct those funds from your user’s account; if you sell premium accounts, you could revert this particular account back to non-premium status.

6.3.3. Handling Chargebacks

When receiving Chargeback (and Collection) information, it is important to handle it properly.

There are 2 ways to track the Chargeback status of a particular payment (for a list of the parameters provided with the Clearing API, please refer to Clearing API Message Structure):

Using the balance parameter

The balance parameter can be used to track the overall balance of a given transaction. Due to its ease of use, this is our suggested method of Chargeback handling. We suggest implementing the following behavior:

1. **Balance < 0:**
   We suggest blocking the Buyer responsible for this transaction (if he is not already blocked). Depending on your system, this might mean removing premium features that were purchased with this transaction, or not allowing the Buyer to use your system at all.

2. **Balance = 0:**
   In this case, the outstanding amount for this transaction could be collected. If the user doesn't have further transactions that generated Chargebacks, he should be unlocked again.

Using the billingtype and amount parameters

If you want to track the exact occurrence of Chargebacks and Collections, we recommend using the **billingtype** and **amount** parameters (for a list of the parameters provided with the Clearing API, please refer to Clearing API Message Structure).

The following combinations of these 2 parameters are possible:

Billingtype	Amount	Case	Description
3	negative amount	Chargeback	A regular Chargeback.
3	positive amount	Chargeback Cancellation	A previous Chargeback might get cancelled if the reason for the Chargeback gets invalidated in time.
4	positive amount	Collection	Money from a previous Chargeback that could be collected.
4	positive amount	Collection Cancellation	If there is a problem with a previous Collection, that collection might get cancelled.

6.4. Technical Information Clearing API

6.4.1. Message Parameters

Parameter Name	*Value* Format	Description
billingtype	*1* number	Type of billing record. 3...CHARGEBACK 4...COLLECTION
id	*77616645* number	unique identifier for this message
appcode	*12345* number	The appcode is used to identify your application

Parameter Name	***Value*** **Format**	**Description**
modified	*2020-07-12* date "YYYY-MM-DD"	Date of update of (e.g. chargeback). **Please note**: This date is expressed for the CET/CEST time zone.
transactionid	*fa6a8417-321d-4fea-851f-ab182d35cc70* Alphanumeric chars [a-z0-9] - 36 chars long	A unique ID used to identify the current payment
customtransactionid	*<your-transactionid>* String	The custom transactionid you provided when creating this transaction. This can be your defined format type, like a specific hash string.
userid	*user123* String	The userid of your customer. This is the same ID you provided when creating the payment.
submerchantid	*<submerchantid>* String	If you are an Aggregator, Onlinegaming Plattform or if you sell virtual currency to different merchants, please use this unique ID to identify your individual Merchants or Game
currency	*EUR* 3 chars	The currency of the amount that will be transferred to the merchants account. Please refer to Supported Currency Codes for more information.
amount	*1.00* float	The payout amount in local currency (can also be a negative value).
duedate	*2020-07-12* date "YYYY-MM-DD"	Due date for the payout. **Please note**: This date is expressed for the CET/CEST time zone.
hashedcallerid	*ba310178e29….e86d42d9a2* 30 chars	Hashed value of your client´s telephone number
transactiondate	*2020-05-02* *date "YYYY-MM-DD"*	The date on which the transaction has been made. **Please note**: This date is expressed for the CET/CEST time zone.
requesttimestamp	*1397564362123*	This parameter is provided with every request. It is needed to verify the signature of the request (for more details, please refer to Calculating the Signature).

Additionally for Chargebacks & Collections		
Parameter Name	***Value*** **Format**	**Description**
balance	*-1.21* float	Showing the balance between chargebacks and collections
carrierinvoicenumber	*"inv-234094095"* String	The carrier's invoice number (if available)
carrierinvoicedate	*2014-07-12* date "YYYY-MM-DD"	The carrier's invoice date (if available). **Please note**: This date is expressed for the CET/CEST time zone.

6.4.2. Clearing API Message Example

Assuming the clearing API URL of the merchant is set to: https//my-clearing-endpoint.com/

Chargeback (billingtype 3)

```
https://my-clearing-endpoint.com/?billingtype=3&id=77616645&
appcode=1172&modified=2020-08-14&
transactionid=180364-1405175288526&
customtransactionid=26979306&userid=user1&
submerchantid=Submerchant1&currency=EUR&amount=-7.14&
duedate=2014-10-10&
hashedcallerid=e8f917233f8bcfc98fc9ca31ca8c0f3e&
transactiondate=2020-05-02&
balance=-7.14&
carrierinvoicenumber=9775864898&carrierinvoicedate=2020-07-24
```

Collection (billingtype 4)

```
https://my-clearing-endpoint.com/?billingtype=4&id=77671436&
appcode=1172&modified=2020-08-17&
transactionid=180364-1405175288526&
customtransactionid=26979306&userid=user1&
submerchantid=Submerchant1&currency=EUR&amount=7.14&
duedate=2020-10-10&
hashedcallerid=e8f917233f8bcfc98fc9ca31ca8c0f3e&
transactiondate=2020-05-02&
balance=0&
carrierinvoicenumber=9775864898&carrierinvoicedate=2020-07-24
```

6.4.3. Clearing API Response Codes

When receiving a Clearing API message, please make sure to verify that the incoming API request is valid.
For more information on the available response codes as well as on how to verify incoming requests, please refer to Response Codes PSN.