Autopay Online Payment Gateway - Documentation

Data wygenerowania: 2026-06-03

Autopay Online Payment Gateway - Documentation

Autopay Online Payments is a comprehensive solution for accepting payments from customers of online stores, supporting various payment methods available on the market, such as transfers, Pay by link, BLIK, Google Pay, Apple Pay.
In this documentation, you will find everything you need to quickly implement the payment gateway in your online store.
The Autopay Online Payments documentation includes sections such as Transaction and Settlement Management, Additional Extensions.

Definitions

Application – Partner's Mobile Application, communicating with the Autopay Online Payment System SDK to register Transactions.

AP – Autopay S.A. Company, headquartered in Sopot, at Powstańców Warszawy 6, registered with the District Court Gdańsk-North in Gdańsk, Commercial Division VIII of the National Court Register under KRS number 0000320590, with tax ID (NIP) 585-13-51-185, REGON 191781561, with share capital of 2 205 500 PLN (fully paid), supervised by the Financial Supervision Commission and registered as a national payment institution under IP17/2013, owner of the System.

BalancePoint (Settlement Point) – an entity within the Payment System representing a Store integrated with the Marketplace Platform and registered in the Payment System using a form provided by AP to the Partner.

ClientHash - a parameter in messages; allows a payment instrument (e.g., Card) to be assigned to a Client in an anonymized way. Based on this, the Partner can initiate subsequent charges in the automatic payment model.

CommissionModel – a commission model established with the Integrator. It describes the commission values for transactions transferred to AP and the Integrator.

Business Day – a day from Monday to Friday, excluding public Polish holidays.

Integration Form – a web page where a form is available, allowing the Client to register, edit, or add a new Service.

Payment Instrument (Payment Channel) – a set of procedures or an individualized device agreed upon by the Client and their provider, used by the Client to initiate a payment order, e.g., Card, PBL.

e-transfer tool – a set of procedures or an individualized device agreed upon by the Partner and AP, used by the Partner to initiate a payment order allowing the withdrawal of funds from the balance to the bank account of the Partner or Client and another payment instrument owned by the Partner or Client. The availability of the functionality depends on individual arrangements between the Partner and AP.

Integrator – Integrators are called partners who have implemented Autopay Online Payments in their systems and enable their activation from within their own solutions. Integrators include entities such as Shoper, Sky-Shop, Gymsteer, Selly verifications, FaniMani, AtomStore, Ebexo, Selly Azymut, PayNow, Comarch.

IPN (Instant Product Notification) - immediate notification sent from the Online Payment System to the Partner Service communicating a change in product status. The structure of the IPN is similar to the ITN (extended only by the node *product*).

ITN (Instant Transaction Notification)- immediate notification sent from the Online Payment System to the Partner Service transmitting a change in the status of the transaction.

ISTN (Instant Settlement Transaction Notification) - immediate notification of a change in the status of a settlement transaction. The system shall immediately transmit notifications of the fact that a settlement transaction has been ordered (withdrawals/returns, if any) and of a change in its status.

ICN (Instant Configuration Notification) - immediate notification of the configuration of a newly registered shop, communicating information about a change in the shop's card status (e.g. in the case of card activation). ICN messages can also be sent in the event of a change in the shop's configuration, a change in its AML data, the enabling/disabling of a payment channel. The provision of functionality is subject to individual arrangements between the Partner and AP.

Card - A payment card issued under the VISA and Mastercard systems, permitted by the regulations of those systems to execute Transactions without its physical presence.

Customer (Payer) - a person who makes a payment on the Website for services or products of a Partner using the System.

Product basket - This is the information about the components of the payment, transferred (in the payment link) to the System for subsequent processing. Each product of the shopping cart is described by two mandatory fields: the constituent amount and a field allowing the transfer of parameters specific to the product.

Payment link - request enabling the start of an Input Transaction (described in part Start of the transaction). It can be used directly on the website (POST method), while in e-mails to customers it should be used *Pre-transaction* in order to obtain a short link to the payment (GET method).

Verification link - URL directing to the Verification Transfer.

Marketplace - a payment solution operating within the framework of the Autopay Online Payment System. It enables the Partner to operate a sales platform where products or services are offered to customers by the Partner's Contractors. Advanced settlement models for Transactions and Settlement Transactions allow payments to be made directly from the Customer to the Partner Contractor, taking into account the Basket of products. The provision of the functionality is subject to individual arrangements between the Partner and AP.

Payer Model - a model in which the commission for the transaction carried out is paid by the customer to AP (cost added to the amount of the transaction). In this case, the customer also accepts AP's terms and conditions during payment.

Merchant's model - a model in which the commission is settled between Autopay and the partner

and is not added to the amount of the transaction paid by the customer.

Partner - the entity that is the recipient of funds from the sale of products or services distributed by the Affiliate on the Site.
A partner, in the case of the Marketplace model, is an entity, which is not a consumer, interested in handling the acceptance by AP of payments due to the partner for products or services distributed by the partner.

Pay By Link (PBL) - a tool for making payments via interbank transfer from the customer's account to the AP account. After logging in to online banking - the data needed to execute the transfer (recipient's information data, number of his bank account, amount and date of transfer execution) are filled in automatically thanks to the data exchange system between the bank and AP.

Technical Agent - the entity with the right to access the Partner's Payment Account, which authorises this access (consent or agreement). In the system, the power of attorney is represented by the configuration of the **PlenipotentiaryID**: one entity can have multiple proxies for different Partners.

Marketplace platform - platform on which the option to register Settlement Points is made available.

Automatic payment - payment made without the need each time to enter the Card data or the data for authorising transfer.

One-click payment - is an automatic payment ordered by the customer.

Cyclical payment - is an automatic payment ordered without customer involvement (by the Partner Service).

Pre-transaction - specific (performed in the background) ordering payment link.

Verification transfer - The part of the process related to registration and editing of the Partner's Service(s) in the System. It consists of the Partner performing a fund transfer to verify the data and bank account for the disbursement of funds to the Partner. Data verification is an obligation of the AP arising, inter alia, from the Act of 16/11/2000 on the prevention of money laundering and financing of terrorism. Each verification transfer must receive final verification status (positive or negative) within 30 days of payment of the transaction. If the final verification status is not given within the timeframe specified above, the system will automatically give it a negative status. This process applies to verification where Autopay directs a request to complete the data to the customer and does not receive the return information necessary to carry out this verification.

Payment Account (Balance) - payment account maintained by AP for the Partner, on which the funds deposited from Customers are collected. The provision of the functionality is subject to individual arrangements between the Partner and AP.

RPAN (Recurring Payment Activation Notification) - message about activation of the automatic payment service.

RPDN (Recurring Payment Deactivation Notification) - message about deactivation of the automatic payment service.

Serwis - the Partner's website or websites integrated with the System, where the Customer can purchase products or services from the Partner (or from the Partner's Counterparty in the case of the Marketplace).
In the case of a Marketplace, an object in the Payment System representing the Partner's Marketplace. All transactions started by the said Marketplace are attached to it.

Specyfikacja - documentation describing the communication between the Service and the System.

AP Online Payment System (System) - an IT and functional solution whereby the AP provides the Partner with an application to process customer payments made with the use of Payment Instruments, as well as to verify the status of payments and to receive payments.

Fast Transfer - execution of payments via intra-bank transfer from the Customer's account to AP's account. The payment made via PBL differs from payments made via PBL in that the Customer must fill in all the data needed to make the transfer himself.

Transaction - means a payment transaction in the meaning of the Polish Act of 19 August 2011 on payment services.

Input transaction - part of the payment handling process concerning the payment made by the customer to AP.

Settlement transaction - part of the payment processing process, concerning the transfer made by AP to the Partner's account. In order for a Settlement Transaction to be created, the Input Transaction must be paid for by the Customer. A Settlement Transaction may relate to a single Input Transaction (payment), or aggregate multiple of them.

Act - Polish Act of 19 August 2011 on payment services.

Link validity - parameter specifying the point in time beyond which the Payment Link ceases to be active. It should be set by the `LinkValidityTime` parameter in the Payment Link.

Validity of transactions - parameter specifying the point in time beyond which the Payment Link ceases to be active. It should be set by the LinkValidityTime parameter in the Payment Link.

Autopay widget - a mechanism enabling payment by Card for products/services offered by the Partner, in which Card data are entered by the Client into the mechanism embedded directly in the Partner's Website. Invoking the Card widget format requires the implementation of JavaScript code using a dedicated AP library.

Onboarding widget - a solution that allows the Integrator to embed the Integrator Form (prepared by Autopay) directly on the Integrator's website, so that the Partner is not redirected to the Autopay domain when registering their shop - the whole process is carried out on the Partner's Website.

WhiteLabel - integration model, in which the customer already on the Service selects the payment channel and accepts the rules and regulations (provided the need for their acceptance results from individual arrangements between the Partner and AP), and the start of the transaction includes a completed `GatewayID` field (and in certain cases `DefaultRegulationAcceptanceID` or `RecurringAcceptanceID`).

Initiation of a payment order - the point in time when the payment gateway user selects a

payment channel and is redirected to the page according to the selected payment channel or (for automatic payments, e-wallets or BLIK) an attempt is made to debit the card or account at the payment channel provider.

Environment addresses

TEST

- gate_host: `https://testpay.autopay.eu`
- cards_gate_host: `https://testcards.autopay.eu`

PRODUCTION

- gate_host: `https://pay.autopay.eu`
- cards_gate_host: `https://cards.autopay.eu`

Transaction and settlement processing

Flow chart of the transaction and clearing service

On the Partner Site, after completing the order, the Customer is presented with the option of making a payment using the System. Clicking on the appropriate link initiates the transaction and opens in a new window:

a) a dedicated page of the System prepared by AP, where the Customer is presented with a list of available Payment Channels and a summary of the registered transaction or

b) directly from a Payment Channel (Bank, BLIK or for payment by Card).

On the System's side, the transmitted parameters are validated and the transaction is saved with a fixed validity period. If, at the time of validation, the validity period of the link is already exceeded, the Customer will receive an appropriate message (verification of the validity of the transaction also takes place when the payment status is changed). After positive verification of the transaction parameters (and after selection of the Payment Channel), the Customer authorises the transaction. In its title, in addition to the identifiers assigned by the System, there may also be a fixed description, agreed in advance between the AP and the Partner, or a dynamic value provided by the Partner at the start of the transaction.

The recommended integration model is to transmit a message to start a transaction in the background, i.e. without redirecting the user to the System (see Pre-transaction). In this model, it is possible to use advanced forms of transaction authorisation (WhiteLabel, automatic payments, SDK mobile), diagnosis of the correctness of the transmitted parameters and many other extensions.

Once the transaction is authorised (on the Payment Channel page) the Customer returns from it to the System, where the Customer is automatically redirected to the Partner Service.

> **TIP:** A detailed description of the structure of the return link can be found in part of the Redirection to Partner Site.

The authorisation (payment) status received from the Payment Channel is transmitted from the System to the Partner Service via an ITN message. The System will continue to send messages until the receipt is acknowledged by the Partner Service or the validity period of the notification expires. Transactions which are paid after the expiry of the validity period of the transaction - will be returned to the Customer (sender of the transfer).

Optionally, the System may notify the fact that a Settlement Transaction has been issued. A suitably modified ISTN message is used for this purpose.

Steps for integrating the handling of transactions and settlements

Data required for the integration of transaction and settlement processing

The required data exchanged during integration differs for test and production environments. Below is a list of parameters passed from AP to Partner and in the reverse direction.

General information is also provided, i.e. active payment channels with graphics (as a result of querying the list of available payment channels).

Optionally, there may be additional data transmitted by the Partner to the AP, for example: information about the required content of the shopping cart and the way it is processed (in reports, billing, admin panel), additional requirements (for prepaid balance recharge). For automatic BLIK payments also the default lifespan of activated automatic payments and the default label of activated automatic payments.

Data exchanged in the test environment

Data provided by the Partner to AP:

- Address for ITN messages
- Address for RPAN messages (may be the same as for ITN messages) [for automatic payments].
- Address for RPDN messages (may be the same as for ITN messages) [for automatic payments].
- Payment return address (no parameters)

Data transferred from AP to Partner:

- Address of the online payment system
- ServiceID
- AcceptorID [for wallets in the WhiteLabel model]
- Shared key
- Hashing result
- Test form address
- IP address from which ITNs are sent
- Address to administration panel
- Login

- Password

Data transferred in a production environment

By Partner to AP:

- Address for ITN messages
- Address for RPAN messages (may be the same as for ITN messages) [for automatic payments].
- Address for RPDN messages (may be the same as for ITN messages) [for automatic payments].
- Payment return address (no parameters)
- Email addresses for transaction reports
- Email addresses for invoices and billing reports
- Email addresses for complaints (sent in messages to Payers)

By AP to Partner:

- Address of the online payment system
- ServiceID
- AcceptorID [for wallets in the WhiteLabel model]
- Shared key
- Hashing result
- Test form address
- IP address from which ITNs are sent
- Address to administration panel
- Login
- Password

Implementation of interfaces and processes on the partner's side

In the minimum version of the integration, support for starting a transaction, returning from it and support for ITN communication should be implemented.

> **TIP:** It is advisable to familiarise yourself with the scheme of operation. If necessary, it is also advisable to familiarise yourself with additional parameters or services.

Integration tests and migration to the production environment

During testing, the white fields of the sheet should be completed and sent back to the AP to confirm correct integration before migration to the production environment.

> **TIP:** Prior to production deployment, it is recommended to perform tests in accordance with the test scenarios in the basic version and, for more advanced integrations, also according to additional scenarios.

Start of the transaction

Description of the start of the transaction

The Partner service initiating the transaction transmits to the Online Payment System the parameters necessary to complete the transaction and the subsequent transmission of the payment status.

All parameters are passed via the POST method (*Content-Type: application/x-www-form-urlencoded*).

The protocol is case-sensitive in both names and values of parameters. Values of transmitted parameters should be encoded in UTF-8 (and transport protocol - encode before sending, unless the tool used to send the message does not do this itself, encoding example: URLEncode).

List of transaction start parameters

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System. Numbers are acceptable.
2	OrderID	YES	string{1,32}	Transaction identifier of up to 32 Latin alphanumeric characters. The field value must be unique for the Partner Service. Acceptable alphanumeric Latin characters and characters in the range: -_
3	Amount	YES	amount	Transaction amount. A dot '.' is used as decimal separator. Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot. **NOTES:** The permissible value of a single Transaction in the Production System is respectively: • *for PBL* – min. 0.01 PLN, max. 100000.00 PLN (or up to the amount set by the Bank issuing the payment instrument); • *for Payment Cards* – min. 0.10 PLN, max. 100000.00 PLN (or up to the individual limit of a single transaction at the Card Issuer's Bank); • *for Fast transfers* – min. 0.01 PLN, max. 100000.00 PLN (or up to the individual limit of a single transaction at the Bank for an intra-bank transfer); • *for BLIK* – min. 0.01 PLN, max. 75000.00 PLN (or up to the individual limit of a single transaction at the Bank for an intra-bank transfer); • *for deffered payments* – min. 100.00 PLN, max. 2000.00 PLN; • *for Alior Installments* – min. 50.00 PLN, max. 7750.00 PLN_

Hash order	name	required	type	description
4	Description	NO	string{1,79}	Title of the transaction (payment); at the beginning of the transfer title, the transaction identifiers assigned by the Online Payment System are placed, to which the value of this parameter is appended. In some cases, independent of the AP, the title of the transfer may be additionally modified by the Bank, in which the payment made by the customer took place. The value of the parameter allows for alphanumeric Latin characters and characters in the range: `. : - , spacja`.
5	GatewayID	NO	integer{1,5}	Identifier of the Payment Channel by which the Customer intends to settle the payment. This parameter is responsible in particular for the presentation model of the Payment Channels: • - on the AP page - parameter value "0"; • - on the Partner Site - the value of the parameter corresponds to the Payment Channel selected by the Customer, e.g. GatewayID=3. All Payment Channels to be embedded on the Website are made available to the Partner as part of the service **gatewayList**.
6	Currency	NO	string{1,3}	Transaction currency; the default currency is PLN (the use of another currency must be agreed during integration). One currency is supported within ServiceID. Acceptable values only: PLN, EUR, GBP and USD.
7	CustomerEmail	NO	string{3,255}	Customer email address.
19	ValidityTime	NO	string{1,19}	Transaction expiry time; when exceeded, the link ceases to be active and any deposit is returned to the sender of the transfer; example value: 2021-10-31 07:54:50; if the parameter is missing, the default value of 6 days is set. The maximum validity of a transaction is 31 days (if the parameter value is set further forward than 31 days, the validity time will be reduced accordingly). E.g. a transaction initiated at 2020-05-01 08:00:00, with ValidityTime = 2021-05-01 08:00:00, will receive validity until 2020-06-01 08:00:00.(Time in CET)

Hash order	name	required	type	description
34	LinkValidityTime	NO	string{1,19}	Link expiry time; when this time is exceeded, the link becomes inactive, but this does not affect the deposit time; example value: 2014-10-30 07:54:50; please ensure that the transaction time is adjusted to the link expiry time (you may also need to enter the parameter **ValidityTime**, to extend its standard validity).
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions.

Method of initiating a transaction

The transaction is initiated by sending an HTTPS call a combination of the above parameters to the address of the online payment system established during registration of the service.

IMPORTANT: The number of transactions launched by the Partner in one minute can be a maximum of 100, unless the Partner and AP agree on a higher limit as part of the concluded agreement.

Example of starting a transaction:

Address:

- https://{gate_host}/path

Parameters:

- ServiceID=2
- OrderID=100
- Amount=1.50

```
Hash=2ab52e6918c6ad3b69a8228a2ab815f11ad58533eeed963dd990df8d8c3709d1
```

Sending a message without all **required** parameters (**ServiceID, OrderID, Amount and Hash**) or containing incorrect their values, will cause the payment process to stop with a transaction error code and a brief error message (no return to the Partner Service page).

IMPORTANT! The parameter pair **ServiceID** and **OrderID** uniquely identifies the transaction. It is not permissible for the value of the **OrderID** parameter to be repeated throughout the entire period of service provided by the System to a single Partner Service (**ServiceID**).

The optional parameter **GatewayID** is used to specify the Payment Channel through which the payment is to be made. This allows the Payment Channels selection screen to be transferred to the Service. The current list of Payment Channel IDs, including logos, is available via the **gatewayList** method.

The transaction initiation message can be transmitted in the background, i.e. without redirecting the user to the Online Payment System. In this model, the selection of the Payment Channel itself is also made by the Customer on the Partner Service.

Redirection to Partner Site

Description of redirection to Partner Site

Immediately upon completion of the transaction authorisation by the Customer, he/she is redirected from the Payment Channel site to the Payment System online site, where the Customer is automatically redirected to the Partner Service. The redirection is implemented by sending an HTTPS request (using the GET method) to a predetermined return address on the Partner Service. The protocol is case-sensitive in both names and parameter values.

List of redirection parameters for the Partner Site

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	OrderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Example of a message redirecting the Customer from the online payment system to the Partner Site

```
https://shop_name/return_page?ServiceID=123458&OrderID=123402816&Hash=5432d69a66d721b2f5
f785432bf5a1fc1b913bdb3bba465856a5c228fe95c1f8
```

Instant notifications (ITN)

Description of instant notifications

The System transmits notifications of changes in the status of a transaction as soon as it receives such information from the Payment Channel, and the message always relates to a single transaction.

> **NOTE:** The domain must be public and accessible via the System. The domain must be secured by a valid certificate issued by a public certification authority (Certificate authority) The server must present a complete certificate chain (Chain of Trust) Communication must be based on TLS protocol version 1.2 or 1.3 *Other forms of connection security, e.g. VPN, mTLS must be individually agreed with the person responsible for implementation.

Example:

```
https://shop_name/status_receive
```

Notification of a change in the status of an input transaction consists of the sending by the System of an XML document containing the new transaction statuses.

The document is sent via HTTPS (default port 443) - using the POST method, as an HTTP parameter with the name transactions. This parameter is stored using the Base64 transport encryption mechanism.

Document format (XML)

```
    <?xml version="1.0" encoding="UTF-8"?>
        <transactionList>
                <serviceID>ServiceID</serviceID>
                <transactions>
                        <transaction>
                                <orderID>OrderID</orderID>
                                <remoteID>RemoteID</remoteID>
                                <amount>999999.99</amount>
                                <currency>PLN</currency>
                                <gatewayID>GatewayID</gatewayID>
                                <paymentDate>YYYYMMDDhhmmss</paymentDate>
                                <paymentStatus>PaymentStatus</paymentStatus>
<paymentStatusDetails>PaymentStatusDetails</paymentStatusDetails>
                        </transaction>
                </transactions>
                <hash>Hash</hash>
        </transactionList>
```

> **NOTE:** A **transactions** node can only contain one **transaction** node (so the notification is always for one transaction). The values of the **orderID** and **amount** elements relating to each transaction are identical to the values of the corresponding fields provided by the Partner Service at the start of the respective transaction. The exception to this is models where the commission is added to the transaction amount. In such cases, the amount value in the ITN is

increased by this commission. The validation of amounts can then be carried out on the basis of the optional ITN field **startAmount**. However, this field must be requested during integration.

List of returned parameters for instant notifications

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System. Numbers are acceptable.
2	orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
3	remoteID	YES	string{1,20}	Alphanumeric transaction identifier assigned by the online payment system. It is worth storing it with the order for further processing (for multiple transactions with the same **OrderID**, for returns, etc.). Such a situation may occur, for example, if the Customer changes the Payment Channel, calls up the same transaction start again from the browser history, etc. The system allows blocking such cases, but the option is not recommended (it would not be possible to pay for an abandoned transaction).
5	amount	YES	amount	Transaction amount. A dot '.' is used as decimal separator. Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot.
6	currency	YES	string{1,3}	Currency of transaction.
7	gatewayID	NO	string{1,5}	Identifier of the Payment Channel through which the Customer settled the payment.
8	paymentDate	YES	string{14}	The time when the transaction was authorised, transmitted in the format YYYYMMDDhhmmss. (CET time)

Hash order	name	required	type	description
9	paymentStatus	YES	enum	Transaction authorisation status, takes values (description of status changes further on): • **PENDING** - transaction initiated. • **SUCCESS** - correct authorisation of the transaction, the Partner Service will receive the funds for the transaction - goods/services can be issued. • **FAILURE** - the transaction was not completed correctly.
10	paymentStatusDetails	NO	string{1,64}	Detailed transaction status, value can be ignored by the Partner Service.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

TIP: Element **hash** (message) is used to authenticate the document. For a description of how the hash is calculated, see section Security of transactions.

Response to the instant notification

In response to the notification, an HTTP status of 200 (OK) is expected and a text in XML format (unencoded Base64), returned by the Partner Service in the same HTTP session, containing an acknowledgement of receipt of the transaction status.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <confirmationList>
            <serviceID>ServiceID</serviceID>
            <transactionsConfirmations>
                    <transactionConfirmed>
                            <orderID>OrderID</orderID>
                            <confirmation>Confirmation</confirmation>
                    </transactionConfirmed>
            </transactionsConfirmations>
            <hash>Hash</hash>
    </confirmationList>
```

Description of the confirmation fields for immediate notifications

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID derived from the message.
2	orderID	YES	string{32}	The transaction identifier, assigned in the Partner Service and communicated in the start of the transaction, derived from the message.
3	confirmation	YES	string{1,25}	The element is used to convey the status of verification of the authenticity of the transaction by the Partner Service. The value of the element is determined by checking the correctness of the values of the **serviceID** and **currency** parameters, comparing the values of the **orderID** and **amount** fields in the notification message and in the message initiating the transaction, and verifying the consistency of the calculated hash from the message parameters with the value passed in the message hash field. Exceptions are models where the commission is added to the transaction amount. In such cases, the amount value in the ITN is increased by this commission. Amount validation can then be carried out on the basis of the optional ITN field startAmount. However, this field must be requested during integration. Two values are provided for the element **confirmation**: • **CONFIRMED** - the parameter values in both messages and the hash parameter match - the transaction is genuine; • **NOTCONFIRMED** - the values in the two messages are different or a hash mismatch - transaction not authentic;
nd.	hash	YES	string{1,128}	The hash element (in the message response) is used to authenticate the response and is calculated from the values of the response parameters. For a description of how the hash is calculated, see the section Security of transactions.

In the absence of a correct response to the sent notifications, the System will make further attempts to communicate the latest status of the transaction after the specified time has elapsed. The Partner Service should perform its own business logic (e.g. confirmation email), only after the first <message about a given payment status.

TIP: It is worth taking a look at *Message re-transmission scheme ITN/ISTN/IPN/RPAN/RPDN*.

Detailed description of the behaviour and change of payment statuses (paymentStatus)

The customer's choice of payment method will send a status of **PENDING** each time. In the subsequent ITN message, the system will provide the status **SUCCESS** or **FAILURE**.

NOTE The **PENDING** status will not be sent if:

- the Customer abandons or returns from the payment method list screen without selecting a specific method. In this case, the **FAILURE** status will be sent immediately. The PENDING status will not appear as the customer has not started the payment process.
- The final status (**SUCCESS** or **FAILURE**) will be delivered before the ITN is sent with the status **PENDING**.

For a single transaction (with unique parameters **OrderID** and **RemoteID**), there can be no change of status from **SUCCESS** to **PENDING** or **SUCCESS** to **FAILURE**.

In any case, there may be a change of detail status - **paymentStatusDetails** (subsequent messages about a change of detail status are for information only and should not lead to a repeat of the paid service/product shipment, etc.).

In special cases of use, there may be a change of status:

a) **FAILURE** to **SUCCESS** (e.g. after an AP consultant has approved a transaction paid with an incorrect amount. Such behaviour requires special business arrangements and is not enabled by default),

b) **SUCCESS** to **FAILURE** (e.g. after triggering multiple transactions with the same **OrderID** but different **RemoteID**). Such a case occurs when a Customer initiates multiple payments with the same **OrderID** (e.g. the Customer changes his decision on which Payment Channel he wants to pay the transaction with). Each of the payments initiated by him generates ITNs and the individual transactions should be distinguished by the **RemoteID** parameter. As the time of receipt of the **FAILURE** status can vary greatly, it may happen that such a status is received after **SUCCESS** has been received (of course with a different **RemoteID**). In such a case, the ITN message should be acknowledged, but should not entail the cancellation of the transaction status in the Partner's system.

Handling transaction statuses from ITN - Simplified model

In a model where it is not necessary to notify the Customer by email/sms of non-SUCCESS statuses, the amount of information
stored in the Service database and the tracking of RemoteID changes can be reduced.

All You need is:

- for statuses other than **SUCCESS**, each time confirm the ITN with the correct response structure, **CONFIRMED** status and correctly counted Hash field value,

- in the event of receipt of **First** status **SUCCESS**, also add the update of the status, its time and RemoteID in the Service database and the execution of business processes (notifications to the Customer of status changes, execution of paid service/product shipment, etc.),

- in the event of a subsequent **SUCCESS** status, each time confirm the ITN with a correct response structure, **CONFIRMED** status and correctly counted Hash field value, without updating the Service database and without business processes.

Handling transaction statuses from ITN - full model

In a model where the entire history of status changes of transactions and/or notification to the customer of major status changes
of transactions is needed, logic approximating the following description should be used.

Dotychczasowy ogólny Status Płatności	PaymentStatus w ITN	Różne RemoteID	Proces biznesowy (powiadomienia do klienta o zmianie statusu)	Proces biznesowy (wykonanie opłacanej usługi/wysyłki produktu itp.)	Zawartość pola confirmation odpowiedzi	Aktualizacja ogólnego statusu transakcji, jej czasu i wartości RemoteID	Uwagi
Brak	Pending	Nd	Tak	Nie	CONFIRMED	Tak	
Brak	Failure	Nd	Tak	Nie	CONFIRMED	Tak	
Brak	Sucess	Nd	Tak	Tak	CONFIRMED	Tak	
Pending	Pending	Nie	Nie	Nie	CONFIRMED	Nie	
Pending	Failure	Nie	Tak	Nie	CONFIRMED	Tak	
Pending	Sucess	Nie	Tak	Tak	CONFIRMED	Tak	
Failure	Pending	Nie	Nie	Nie	CONFIRMED	Nie	
Failure	Failure	Nie	Nie	Nie	CONFIRMED	Nie	
Failure	Sucess	Nie	Tak	Tak	CONFIRMED	Tak	
Sucess	Pending	Nie	Nie	Nie	CONFIRMED	Nie	Niemozliwe
Sucess	Failure	Nie	Nie	Nie	CONFIRMED	Nie	Niemozliwe
Sucess	Sucess	Nie	Nie	Nie	CONFIRMED	Nie	
Pending	Pending	Tak	Nie	Nie	CONFIRMED	Nie	
Pending	Failure	Tak	Tak	Nie	CONFIRMED	Tak	
Pending	Sucess	Tak	Tak	Tak	CONFIRMED	Tak	
Failure	Pending	Tak	Nie	Nie	CONFIRMED	Tak	
Failure	Failure	Tak	Nie	Nie	CONFIRMED	Nie	
Failure	Sucess	Tak	Tak	Tak	CONFIRMED	Tak	
Sucess	Pending	Tak	Nie	Nie	CONFIRMED	Nie	Niemozliwe
Sucess	Failure	Tak	Nie	Nie	CONFIRMED	Nie	Niemozliwe
Sucess	Sucess	Tak	Nie	Nie	NOTCONFIRMED	Nie	Niemozliwe

Security of transactions

Description of transaction security

The Online Payment System uses several mechanisms to increase
the security of transactions carried out using it. Transmission between all parties to a transaction is carried out using a
secure connection based on the TLS protocol with a 2048-bit key.

In addition, the communication is secured by a hash function calculated from the values of the

message fields and the shared key (the shared key itself is stored in the System in an encrypted form using the AES-ECB algorithm).

The SHA256 or SHA512 algorithm is used as the hash function (method determined at the stage of configuring the respective Partner Service in the online payment system). The default function is SHA256.

Calculation of the value of a hash function

Description of how to calculate the value of the hash function and examples of calculations for basic messages.

> **NOTE:** The examples do not take into account all possible optional fields, so if such fields are present in a particular message, they should be included in the abbreviated function in an order consistent with the number next to the field.

Calculation of hash function value - Hash field

The value of the hash function, used to authenticate the message, is calculated from a string containing the concatenated fields of the message (concatenation of fields). Field values are concatenated, without parameter names, and a separator (in the form of the | character) is inserted between
consecutive (non-empty) values. The order in which the fields are glued together follows the order of their occurrence in the list of parameters in the documentation.

> **IMPORTANT!** If there is no optional parameter in the message or in the case of an empty parameter value, do not use the separator!

To the string created in this way, a key is appended at its end, shared between the Partner Service and the online payment system. From the string created in this way, the value of the hash function is calculated and constitutes
the value of the message Hash field.

Hash = function(field_value_1_message + "|" + *field_value_2_message* + "|" + ... + "|" + *field_value_n_message* + "|" + *shared_key);*

Example calculation of the hash function value at the start of a transaction

Partner Service Data:
ServiceID = 2

shared_key = 2test2

Gateway address https://{gate_host}/sciezka

a. Start of transaction

POST call without basket, with parameters:

ServiceID=2
OrderID=100
Amount=1.50

```
Hash=2ab52e6918c6ad3b69a8228a2ab815f11ad58533eeed963dd990df8d8c3709d1
```

where

```
Hash=SHA256(“2|100|1.50|2test2”)
```

b. Start of transaction. POST call with the shopping cart.

TIP: Option discussed in detail in section Product basket.

ServiceID = 2

OrderID = 100

Amount = 1.50

Product (described below)

shared_key = 2test2

Product basket (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <productList>
       <product>
              <subAmount>1.00</subAmount>
              <params>
                     <param name="productName" value="Nazwa produktu 1" />
              </params>
       </product>
       <product>
              <subAmount>0.50</subAmount>
              <params>
                     <param name="productType" value="ABCD" />
                     <param name="ID" value="EFGH" />
              </params>
       </product>
    </productList>
```

After encoding with the base64 function, we get the value of the Product parameter:

```
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48cHJvZHVjdExpc3Q+PHByb2R1Y3Q+PHN1YkFt
b3VudD4xLjAwPC9zdWJBbW91bnQ+PHBhcmFtcz48cGFyYW0gbmFtZT0icHJvZHVjdE5hbWUiIHZhbHVlPSJOYXp3
YSBwcm9kdWt0dSAxIiAvPjwvcGFyYW1zPjwvcHJvZHVjdD48cHJvZHVjdD48c3ViQW1vdW50PjAuNTA8L3N1YkFt
b3VudD48cGFyYW1zPjxwYXJhbSBuYW1lPSJwcm9kdWN0VHlwZSIgdmFsdWU9IkFCQ0QiIC8+PHBhcmFtIG5hbWU9
IklEIiB2YWx1ZT0iRUZHSCIgLz48L3BhcmFtcz48L3Byb2R1Y3Q+PC9wcm9kdWN0TGlzdD4=
```

The Hash value is calculated as follows:

```
Hash=SHA256(“2|100|1.50|PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48cHJvZHVjdExp
c3Q+PHByb2R1Y3Q+PHN1YkFtb3VudD4xLjAwPC9zdWJBbW91bnQ+PHBhcmFtcz48cGFyYW0gbmFtZT0icHJvZHVj
dE5hbWUiIHZhbHVlPSJOYXp3YSBwcm9kdWt0dSAxIiAvPjwvcGFyYW1zPjwvcHJvZHVjdD48cHJvZHVjdD48c3Vi
QW1vdW50PjAuNTA8L3N1YkFtb3VudD48cGFyYW1zPjxwYXJhbSBuYW1lPSJwcm9kdWN0VHlwZSIgdmFsdWU9IkFC
Q0QiIC8+PHBhcmFtIG5hbWU9IklEIiB2YWx1ZT0iRUZHSCIgLz48L3BhcmFtcz48L3Byb2R1Y3Q+PC9wcm9kdWN0
TGlzdD4=|2test2”)
```

Example calculation of the value of a hash function when returning a customer to the Partner Site

Partner Service Data:

ServiceID = 2

shared_key = 2test2

```
<https://shop_name/strona_powrotu?ServiceID=2>&OrderID=100&Hash=254eac9980db56f425acf8a9
df715cbd6f56de3c410b05f05016630f7d30a4ed
```

gdzie

Hash=SHA256("2|100|2test2")

Example calculation of the value of a hash function in an ITN message

Partner Service Data:

serviceID = 1

shared_key = 1test1

ITN (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
        <transactionList>
           <serviceID>1</serviceID>
           <transactions>
                  <transaction>
                         <orderID>11</orderID>
                         <remoteID>91</remoteID>
                         <amount>11.11</amount>
                         <currency>PLN</currency>
                         <gatewayID>1</gatewayID>
                         <paymentDate>20010101111111</paymentDate>
                         <paymentStatus>SUCCESS</paymentStatus>
                         <paymentStatusDetails>AUTHORIZED</paymentStatusDetails>
                  </transaction>
           </transactions>
           <hash>a103bfe581a938e9ad78238cfc674ffafdd6ec70cb6825e7ed5c41787671efe4</hash>
        </transactionList>
```

where

```
Hash=SHA256(“1|11|91|11.11|PLN|1|20010101111111|SUCCESS|AUTHORIZED|1test1”)
```

Example response (XML)

```
    <?xml version="1.0" encoding="UTF-8"?>
        <confirmationList>
           <serviceID>1</serviceID>
           <transactionsConfirmations>
                  <transactionConfirmed>
                         <orderID>11</orderID>
                         <confirmation>CONFIRMED</confirmation>
                  </transactionConfirmed>
           </transactionsConfirmations>
           <hash>c1e9888b7d9fb988a4aae0dfbff6d8092fc9581e22e02f335367dd01058f9618</hash>
        </confirmationList>
```

where value

Hash=SHA256("1|11|CONFIRMED|1test1");

Example calculation of the hash function value in querying the list of Payment Channels

Partner Service Data:

ServiceID = 100

MessageID = 11111111111111111111111111111111

Currencies = PLN,EUR

Language = PL

shared_key = 1test1

where the value

Hash=SHA256('100|11111111111111111111111111111111|PLN,EUR|PL|1test1')

The response to the above call may be as follows (note: no hash field in the response):


```
{
    "result": "OK",
    "errorStatus": null,
    "description": null,
    "gatewayGroups": [
        {
            "type": "PBL",
            "title": "Przelew internetowy",
            "shortDescription": "Select the bank from which you want to order the
payment",
            "description": null,
            "order": 1,
            "iconUrl": null
        },
        {
            "type": "BNPL",
            "title": "Buy now, pay later",
            "shortDescription": "Buy now, pay later",
            "description": null,
            "order": 2,
            "iconUrl": null
        }
    ],
    "serviceID": "10000",
    "messageID": "2ca19ceb5258ce0aa3bc815e80240000",
    "gatewayList": [
        {
            "gatewayID": 106,
            "name": "PBL test payment",
            "groupType": "PBL",
            "bankName": "NONE",
            "iconURL": "https://testimages.autopay.eu/pomoc/grafika/106.gif",
            "state": "OK",
            "stateDate": "2023-10-03 14:35:01",
            "description": "Test payment",
            "shortDescription": null,
            "descriptionUrl": null,
            "availableFor": "BOTH",
            "requiredParams": ["Nip"],
            "mcc": {
                "allowed": [1234, 9876],
                "disallowed": [1111]
            },
            "inBalanceAllowed": true,
            "minValidityTime": null,
            "order": 1,
            "currencies": [
                {
                    "currency": "PLN",
```

```
                    "minAmount": 0.01,
                    "maxAmount": 5000.00
                }
            ],
            "buttonTitle": "Pay"
        },
                {
            "gatewayID": 701,
            "name": "Pay later with Payka",
            "groupType": "BNPL",
            "bankName": "NONE",
            "iconUrl": "https://testimages.autopay.eu/pomoc/grafika/701.png",
            "state": "OK",
            "stateDate": "2023-10-03 14:37:10",
            "description": "<div class=\"payway_desc\"><h1>Cost details</h1><p>Pay later
- one-off up to 45 days (...). Offer details at: <a href="?r="https://payka.pl\"
target=\"_blank\">Payka.pl</a></p></div>",
            "shortDescription": "Pay later - in one go up to 45 days or in several equal
instalments",
            "descriptionUrl": null,
            "availableFor": "B2C",
            "requiredParams": [],
            "mcc": null,
            "inBalanceAllowed": false,
            "minValidityTime": 60,
            "order": 2,
            "currencies": [
                {
                    "currency": "PLN",
                    "minAmount": 49.99,
                    "maxAmount": 7000.00
                }
            ],
            "buttonTitle": "Pay"
        }
    ]
}
```


Additional extensions

Alternative transaction initiation models

Card pre-authorisation

General description of the operation of the card pre-authorisation service

Card pre-authorisation support provides the functionality of blocking funds on the customer"s card for a certain (e.g. predetermined during the establishment of the blockade) period of time and then making a debit. A special case is when the block is removed without any amount being deducted (e.g. the service to the customer has not been performed).

All these operations (blocking of funds, debiting, withdrawal of blocking) should be ordered via the API of the Autopay Payment System. If there is no debit order to the card during the validity period of the transaction setting up the blockade, the System will release the funds, notifying you with a standard Transaction Status Change (ITN) message.

Other operations (successful debit, placing a block on the card and subsequent debit) also result in the sending of an ITN message. This message is the only binding information about a change in the status of a transaction and (used together with the service **transactionStatus**; see section Enquiry about the status of a transaction), helps to handle the transaction without breaking the session with the user (even in the event of various network problems). Synchronous operation acknowledgements (node **confirmation**, they serve only to present preliminary information about the order).

Steps in a card pre-authorisation transaction

Blocking at the request of the Partner

It is possible to distinguish between 3 basic ways of placing a lock on a card:

a) Establishing a block during the authorisation of a one-off payment (See Scheme A for Preauthorisation). The customer fills in the card format, after the transaction has been started, in which the Partner indicates in the start parameters:

- card payment channel (**GatewayID=1500**) and

- the desire to secure funds rather than encumber (**Hold=true**)

b) Assumption of a lock when initiating an automatic payment (card enrolment in the Service or Mobile Application) (See Scheme B for Preauthorisation)).

> **TIP:** The automatic payment initialization process in the pre-authorization model (sending the RPAN message with ClientHash is started with the **transactionClear** service call.

> **NOTE:** in this specific case, if the transaction will not be cleared in organizations - recursion cannot exist because the transaction has not been completed.

The customer fills in the card format, once the transaction has been started, in which the Partner indicates in the start parameters:

- card payment channel (**GatewayID=1503**),

- the fact of accepting the rules of the automatic payment service provided by AP (**RecurringAcceptanceState=ACCEPTED**, lub po or after the business arrangements **PROMPT/FORCE**)

- the choice to initialise an automatic payment with a potential debit to the card (**RecurringAction=INIT_WITH_PAYMENT**)

- the desire to secure funds rather than encumber (**Hold=true**)

c) Establishing a lock using a previously saved card (see Scheme C for Preauthorisation).

> **TIP:** The automatic payment initialization process in the pre-authorization model (sending the RPAN message with ClientHash is started with the **transactionClear** service call.

The customer does not fill in the card form, but a backend (without redirection) pre-transaction takes place in which the partner indicates in the start parameters:

- card payment channel (**GatewayID=1503**)

- indication of a previously added card (**ClientHash from RPAN**)

- choice of automatic payment method (**RecurringAction=MANUAL**)

- the desire to secure funds rather than encumber (**Hold=true**)

Each of these methods of establishing a blockade results in an ITN message, the status of which indicates the result of the transaction authorisation. In addition to the standard statuses, in the case of blocking of funds, the System may provide in the ITN status **paymentStatus=ON_HOLD**, which confirms the establishment of a block of funds on the customer's card. In addition, the ITN will, as standard, contain a global transaction identifier (**remoteID**), which will be required for subsequent loading of the established blockade.

Card debit at the Partner's request

Description of the card charge at the partner's request

Once the lock has been placed, a debit order can be placed by the Partner on a previously authorised card (See Scheme D for Preauthorisation).For this purpose, the dedicated service must be called up: **transactionClear** (https://{gate_host}/webapi/transactionClear) with the corresponding parameters. All parameters are passed via the POST method (`Content-Type: application/x-www-form-urlencoded`). The protocol is case-sensitive in both parameter names and values. The values of the passed parameters should be encoded in UTF-8.

Description of available parameters for card debit at the Partner's request

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.

Hash order	name	required	type	description
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. on a UID basis), the field value must be unique and indicate a specific payment order on the Partner Service.
3	RemoteID	YES	string{1,20}	The alphanumeric transaction identifier assigned by the System and transmitted to the Partner in the ITN message of the incoming transaction. Its indication will result in a debit to the card authorized in the transaction of the indicated **RemoteID**, if it is in a blocked state (**status ON_HOLD**).
4	Amount	YES	amount	Amount of the debit (must not be greater than the amount of the blockade); a dot is used as decimal separator - '.' Format: 0.00.
5	Products	YES	string{1,10000}	Information about the products included in the transaction, transmitted as Base64 transport protocol encoded XML. The structure must include all products specified in the pre-authorisation, but can be simplified (only productID and idBalancePoint will be taken into account to identify the product whose amount is to be updated, and the new amount should be specified in `subAmount`). /**Required for multiple products specified in the pre-authorisation.**
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

Confirmation of transaction for card debit at Partner's request

For correct querying, a defined HTTP header with appropriate content must be sent along with the passed parameters. The attached header should be named 'BmHeader' and have the following value 'pay-bm, in its entirety it should look as follows 'BmHeader: pay-bm'. In case of a valid message, an XML-formatted text is returned (in the same HTTP session), containing confirmation of the operation or a description of the error.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balancePayoff>
    <serviceID>ServiceID</serviceID>
    <messageID>MessageID</messageID>
    <remoteOutID>RemoteOutID</remoteOutID>
    <hash>Hash</hash>
```

```
        </balancePayoff>
```

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balancePayoff>
    <balancePointID>BalancePointID</balancePointID>
    <messageID>MessageID</messageID>
    <remoteOutID>RemoteOutID</remoteOutID>
    <hash>Hash</hash>
    </balancePayoff>
```

Description of parameters to be returned for card debit at the partner's request

Hash order	name	required	type	description
1	serviceID	YES	string{1,32}	Partner Service ID. Derived from a method request. **Required for confirmation=CONFIRMED.**
2	messageID	YES	string{1,20}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request. **Required for confirmation=CONFIRMED.**
3	confirmation	YES	string{1,100}	Order acknowledgement status. It can take two values: - CONFIRMED – the operation was successful. **NOTE:** This does not mean that the load is executed! The system will asynchronously deliver the ITN with **paymentStatus=SUCCESS**. - NOTCONFIRMED – operation failed.
4	reason	NO	string{1,1000}	Explanation of the details of the processing of the request..
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**. **Required for confirmation=CONFIRMED.**

Release of blocking at the request of the Partner

Once the lock has been established, it can be ordered by the Partner to release it without any deduction of any funds (See Scheme E for Preauthorisation)). For this operation, use the service **releaseHold** (See Cancellation of an unpaid transaction). Upon successful initiation of a lock release (correct response to a cancel
transaction), the System will asynchronously provide an ITN with the **paymentStatus=FAILURE** and

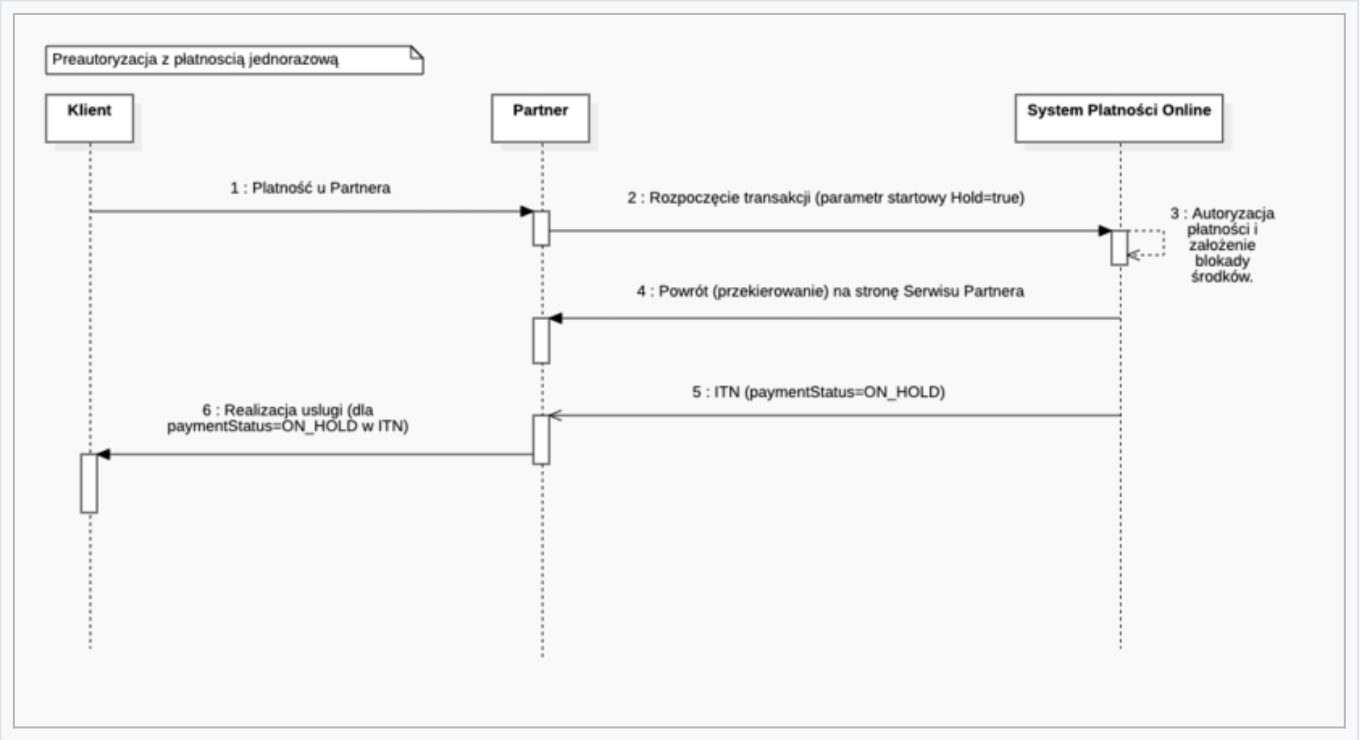
paymentStatusDetails=CANCELLED.

Release of blocking after overdue transactions

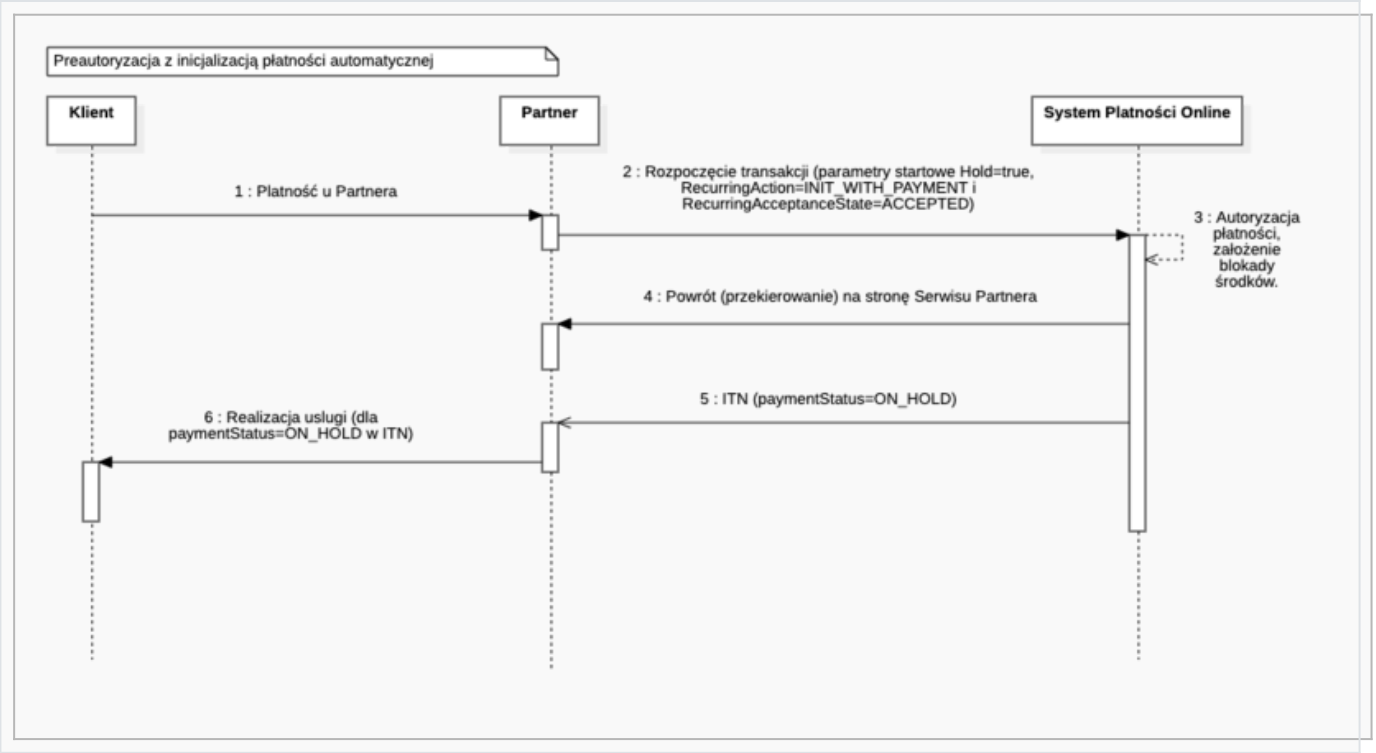
In the event of Partner's inactivity (after the lock has been set up) for a predetermined period of validity, the transaction is released by the System (without deducting any funds) (See Scheme F for Preauthorisation). The system will cancel the transaction, remove the lock and provide the ITN with the **paymentStatus=FAILURE** and **paymentStatusDetails=CANCELLED**.

Schemes for Preauthorisation

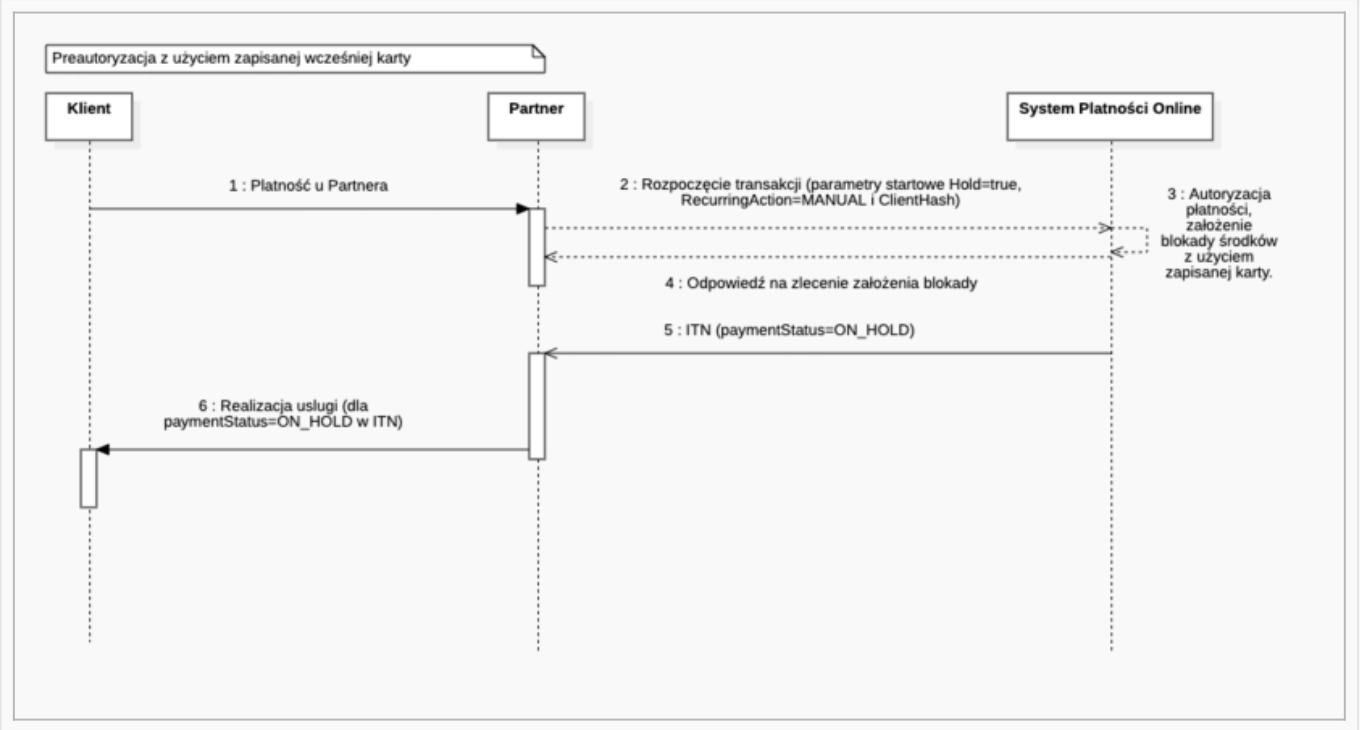
Scheme A for Preauthorisation: Setting up a block during the authorisation of a one-off payment



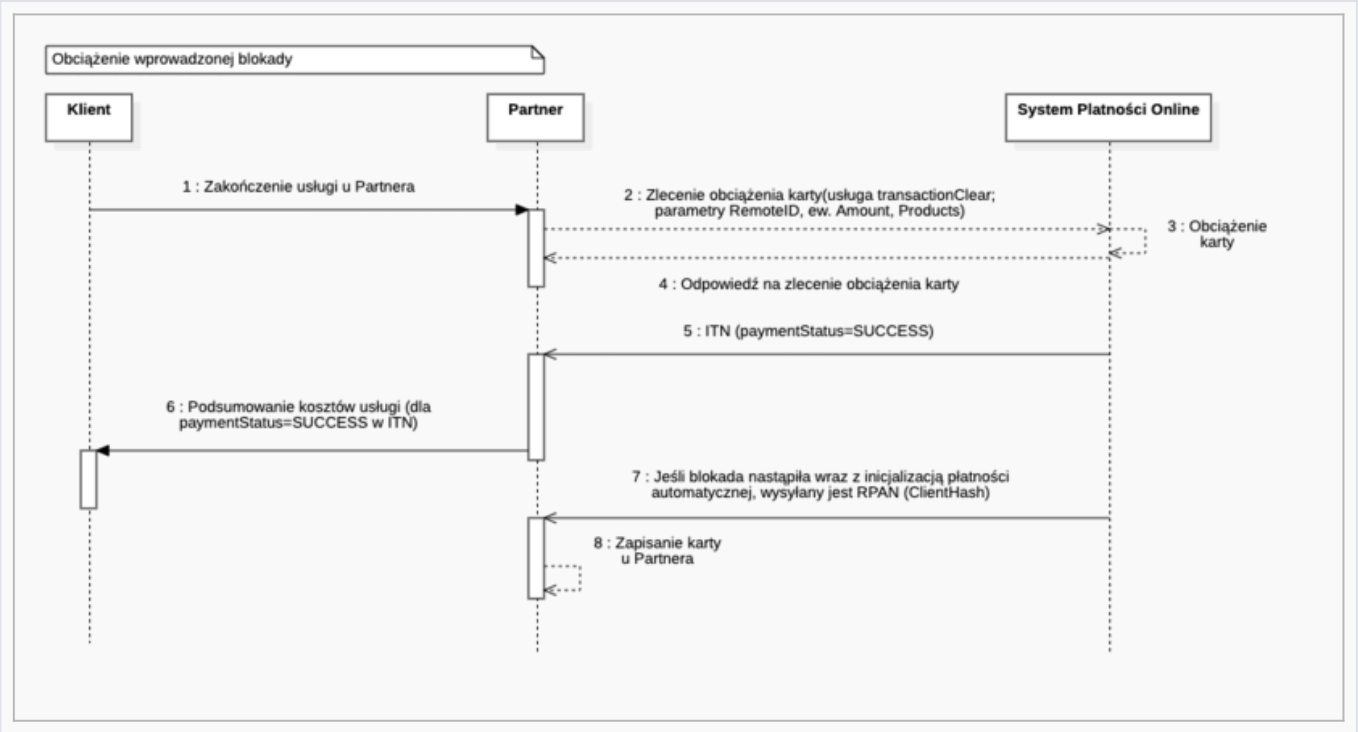
Scheme B for Preauthorisation: Assumption of a block during the initiation of an automatic payment (card enrolment)README.md



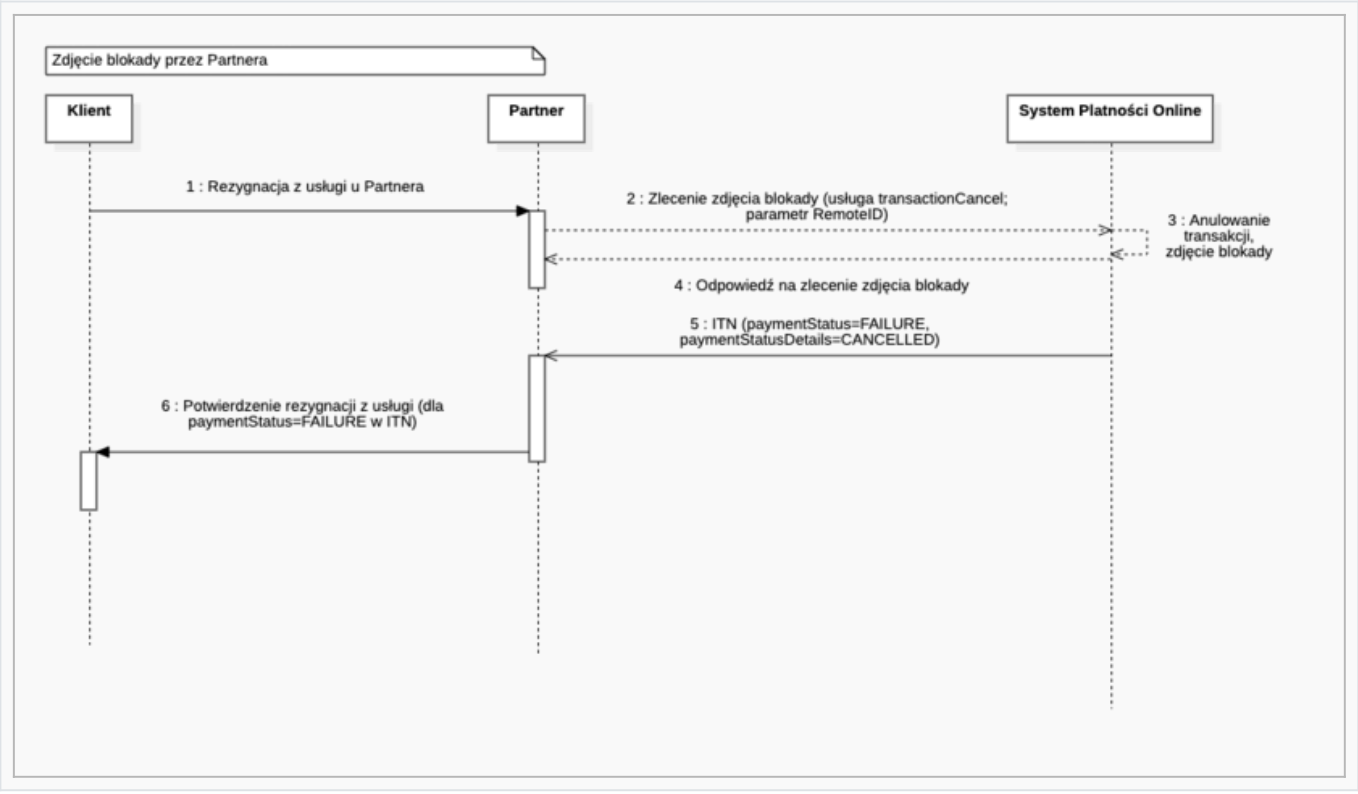
Scheme C for Preauthorisation: Setting up a lock using a previously stored card



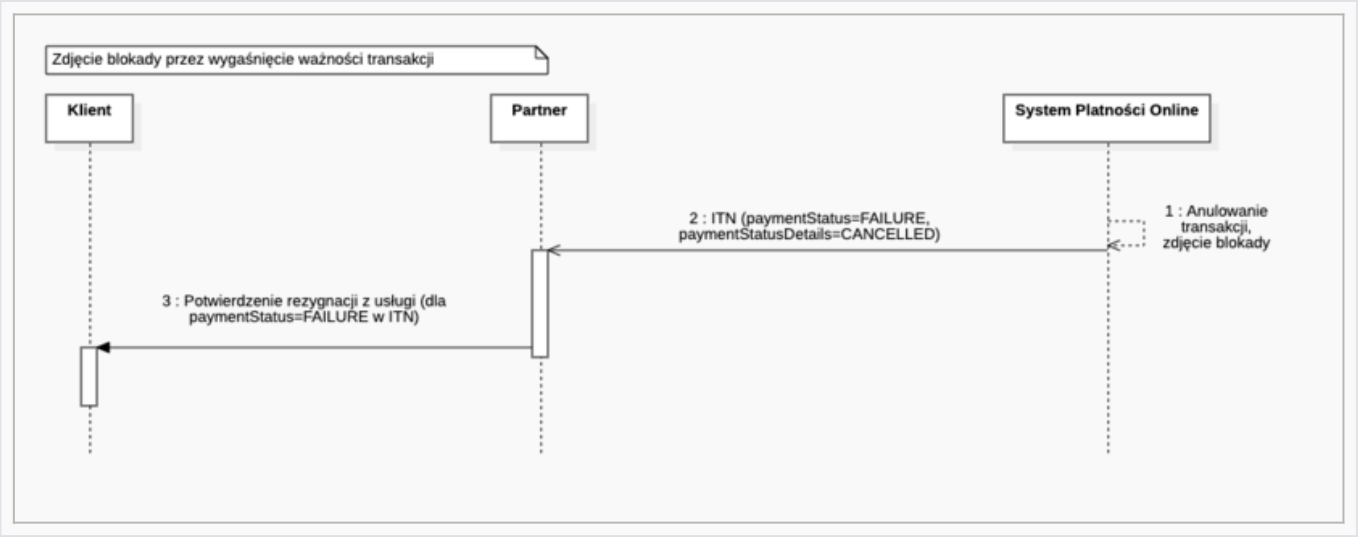
Scheme D for Preauthorisation: Partner's order to debit a previously authorised card



Scheme E for Preauthorisation: Order by Partner to release the lock (without deducting funds)



Scheme F for Preauthorisation: Release of lock by the Scheme (without deduction of funds)



Pre-transaction

Pre-transaction description

Pre-transaction extends the standard transaction initiation model by handling specific needs:

- order a payment link on the basis of the submitted parameters
- debit from the customer (if no additional authorisation is required by the Customer)
- verify the correctness of the payment link before the Customer is redirected to the System - the call results in the validation of the parameters and configuration of the System
- shorten the payment link - instead of several/several parameters, link is shortened to two identifiers
- hiding the data of sensitive parameters of the transaction link - the pre-transaction takes place in the backend, and the link to continue the transaction does not contain sensitive data, but only identifiers of the continuation of
- use of the mobile SDK in a mixed variant - the start of the transaction is performed by the mobile app backend, rather than the SDK itself using the transaction token

The specific use cases of Pre-transaction, are loads:

- BLIK 0\ In order to use this service, you must provide **GatewayID=509** and pass the transaction authorisation code in the parameter **AuthorizationCode**.

- BLIK 0 OneClick

- Charges for "Automatic payment"
  In order to use this service, you must provide one of the **GatewayID=509** i **gatewayType="Płatność automatyczna"** and the necessary parameters.

- Authorisations through Visa wallets
  In order to use this service, you must provide **GatewayID=1511** and pass the encoded token in the parameter **PaymentToken**. In the absence of a token, authorisation will take place on the System website.

- Authorisations through Google Pay wallets

NOTE: The service allows the card stored in the customer's wallet to be debited without redirection to the System. Often, additional authorisation is enforced in the form of 3DS (default behaviour of the test environment, which can be reconfigured).

In the **Whitelabel** model, integrate as described and then provide **GatewayID=1512** and the encoded token in the **PaymentToken** parameter. If there is no token (or a model other than **Whitelabel**), simply enter **GatewayID=1512** - authorisation will take place on the System website.

- Authorisations through Apple Pay wallets
  To use this service, you will need to enter **GatewayID=1513**. Authorisation will take place on the System website.

- Authorisation through the native format of the mobile SDK

NOTE: The service allows the card to be debited, the details of which are provided on the secure card format of the SDK, and the start of the transaction itself is performed by the backend of the mobile application.

In addition to the relevant GatewayID - 1500 for a one-time payment or 1503 for activation of an automatic payment (and other parameters) - the PaymentToken obtained from the SDK and the parameter `WalletType=SDK_NATIVE` (description in section Starting a transaction with additional parameters)

Calling a Pre-transaction

A required element in the case of a pre-transaction is to send backend (using e.g. cURL) the standard start message of the transaction (see Start of the transaction), with a 'BmHeader' of value: 'pay-bm-continue-transaction-url':

Example of a header

'***BmHeader: pay-bm-continue-transaction-url***')

In addition, it is recommended that the parameter **CustomerIP** (for claims, reporting purposes).

Example of Pre-transaction start-up (PHP)

```
    $data = array(
         'ServiceID' => '100047',
         'OrderID' => ‘20161017143213’,
         'Amount' => '1.00',
         'Description' => 'test bramki',
         'GatewayID' => '0',
         'Currency' => 'PLN',
         'CustomerEmail' => 'test@bramka.pl',
         'CustomerIP' => '127.0.0.0',
         'Title' => 'Test title',
        'Hash' => 0c5ca136e8833e40efbf42a4da7c148c50bf99f8af26f5c9400681702bd72056
        );

        $fields = (is_array($data)) ? http_build_query($data) : $data;

        $curl = curl_init('https://{gate_host}/test_ecommerce');
        curl_setopt($curl, CURLOPT_HTTPHEADER, array('BmHeader: pay-bm-continue-
transaction-url'));
        curl_setopt($curl, CURLOPT_POSTFIELDS, $fields);
        curl_setopt($curl, CURLOPT_POST, 1);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, true);
        $curlResponse = curl_exec($curl);
        $code = curl_getinfo($curl, CURLINFO_HTTP_CODE);
        $response = curl_getinfo($curl);
        curl_close($curl);

        echo htmlspecialchars_decode($curlResponse);
```

Response to Pre-transaction - link to follow up on transaction

In the case of correct validation of the parameters (and configuration) and the need for the customer to perform an additional action (selecting a payment channel - if specified **GatewayID=0**, execution/confirmation transfer, entering CVC/CVV code, execution of 3DS) - an XML with a link to continue the transaction will be returned.

Example of a transaction continuation link file (XML)

```
    <?xml version="1.0" encoding="UTF-8"?>
                <transaction>
                <status>PENDING</status>
                <redirecturl>
                https://{gate_host}/payment/continue/96VSD39Z6E/L6CGP5BH
                </redirecturl>
                <orderID>20180824105435</orderID>
                <remoteID>96VSD39Z6E</remoteID>
                <hash>
                1c6eae2127f0c3f81fbed3b6372f128040729a4d4e562fb696c22e0db68dbbe1
                </hash>
        </transaction>
```

Pre-transaction object

The **transaction** object represents the receipt or withdrawal of funds from an AP, such as a completed purchase or refund.

Transaction object attributes for Pre-transaction

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	status	YES	string{1,32}	Transaction status. In this case the constant `PENDING`.
2	redirecturl	YES	string{1,100}	Address to continue a transaction initiated by a pre-transaction message.
3	orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
4	remoteID	YES	string{1,20}	The unique transaction identifier assigned in the AP System.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Response to Pre-transaction - no continuation of transaction

In the event of an invalid validation or unsuccessful load no continuation link is generated. A text in XML format is returned (in the same HTTP session) indicating the processing status of the request.

Example of request processing status (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction>
        <orderID>OrderID</orderID>
        <remoteID>RemoteID</remoteID>
        <confirmation>ConfStatus</confirmation>
        <reason>Reason</reason>
        <blikAMList>
                <blikAM>
                        <blikAMKey>Klucz1</blikAMKey>
                        <blikAMLabel>Etykieta1</blikAMLabel>
                </blikAM>
                <blikAM>
                        <blikAMKey>Klucz2</blikAMKey>
                        <blikAMLabel>Etykieta2</blikAMLabel>
                </blikAM>
        </blikAMList>
        <paymentStatus>PaymentStatus</paymentStatus>
```

```
        <hash>Hash</hash>
</transaction>
```

Pre-transaction outcome

Parameters returned for the result of the Pre-transaction.

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction. **Required for confirmation=CONFIRMED.**
2	remoteID	YES	string{1,20}	The unique transaction identifier assigned in the AP System. **Required for confirmation=CONFIRMED.**
3	confirmation	YES	string{1,100}	Order acknowledgement status. It can take two values: - CONFIRMED – the operation was successful. **NOTE: Does not indicate success.** - NOTCONFIRMED – operation failed.
4	reason	NO	string{1,1000}	Explanation of the reason for rejection of the order (for confirmation=NOTCONFIRMED), if available.
5	blikAMList	NO	string{1,10000}	List of available mobile bank applications under BLIK 0 OneClick option (for confirmation=NOTCONFIRMED and reason=ALIAS_NONUNIQUE).

				Format for blikAMList:
				`<blikAM>`
				`<blikAMKey>Key1</blikAMKey>`
				`<blikAMLabel>Label1</blikAMLabel>`
				`</blikAM>`
				…
				`<blikAM>`
				`<blikAMKey>KeyN</blikAMKey>`
				`<blikAMLabel>LabelN</blikAMLabel>`
				`</blikAM>`

6	paymentStatus	NO	enum	Transaction authorisation status, takes values: - PENDING – transaction initiated - SUCCESS – correct authorisation of transactions - FAILURE – transaction not completed correctly

Hash order	name	required	type	description
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service. Required for confirmation=CONFIRMED.**

Correct validation of parameters

If the parameters (and configuration) are validated correctly and there is no need for the customer to perform an additional action - a confirmation of the load order is returned.

This is the case where the data is sufficient to make a debit for the payment channel in question, for example: BLIK 0 without
the required BLIK code (nor indication of the bank's mobile app alias), recurring payment, OneClick Card payment without the required CVC/CVV/3DS.

Result

confirmation=CONFIRMED

Incorrect parameter validation

In the event of incorrect parameter (and configuration) validation - an error is returned.

Result

confirmation=NOTCONFIRMED

An error may also be returned in the event of a synchronous response from the Payment Channel (e.g. an error specific to an attempt to initialise a BLIK automatic payment, i.e. reason= RECURRENCY_NOT_SUPPORTED).

> **NOTE:** An error may also be returned in the event of a synchronous response from a Payment Channel (e.g. an error specific to an attempt to initialise a BLIK automatic payment, i.e. *reason=RECURRENCY_NOT_SUPPORTED*). Another known case is also the validation error of the address given in the start parameter CustomerEmail (INVALID_EMAIL).

Handling of responses for Transactions

Confirmation status (confirmation)	Payment status (paymentStatus)	Description of the Partner's behaviour
CONFIRMED	SUCCESS	Transaction accepted for processing, status correct. Do not retry debit The payment confirmation can be displayed, but business processes should be paused until the confirmation in the ITN (this will be sent once the AP has received the correct transaction status from the Payment Channel).
CONFIRMED	FAILURE	Transaction accepted for processing, status invalid. You can retry debit with the same **OrderID**. Once the AP has received the status of the transaction from the Payment Channel, an ITN message will be sent. **NOTE:** It is not possible to retry transaction with the same **OrderID** if, during integration, a model is agreed for the System to block transaction starts with the same **OrderID**. By default, the Partner's preservation of the uniqueness of the **OrderID** is only a recommendation and is not subject to verification in transaction starts.
CONFIRMED	PENDING	A transaction has been accepted for processing, but its status is not yet known. Do not retry the load. Further handling as in the case of timeout.
NOTCONFIRMED	-	Transaction not ordered (reason indicated in reason node). You can retry the load with the same OrderID.
Timeout (or other response such as invalid structure, missing required fields, other confirmation status)	-	Wait for the ITN until the expiry date of the transaction (a short expiry time, e.g. 15 min, is recommended for this purpose), informing the customer of the result in a separate process (email/sms). After this time, it is recommended to query the transaction status (**transactionStatus**). If the method returns no registered transaction (or **FAILURE** payment statuses alone), the debit order can be retried with the same **OrderID**. Alternatively, you can try to cancel the transaction, thus speeding up the process of obtaining the final transaction status and possibly the process of renewing the transaction start message. To do so, use the transaction cancellation service (**transactionCancel**) and confirm its operation by querying the transaction status (as described above).

Requesting transfer details for a Fast Transfer transaction

Description of ordering transfer data in a Fast Transfer transaction

Fast Transfer is a form of payment that requires the Customer to independently rewrite the transfer data provided by the System. What type a given Payment Channel is, is told by the gatewayType

parameter in response to calling the service "Querying the list of currently available Payment Channels". The transfer data can be displayed to the Customer:

- on the AP website (execution of the transaction based on the standard transaction start model described in part Start of the transaction)
- on the Partner's site (transaction processing without redirecting the customer to the AP site is described below)

Calling

For the correct transmission of the message, a standard transaction start message must be sent backend (e.g.
cURL), with a header 'BmHeader' of value: '*pay-bm*' (In its entirety, the header should look as follows '*BmHeader: pay-bm*'). If the header is incorrectly defined or missing, the message will be misread. In addition, it is recommended to pass the CustomerIP parameter as described under User IP (needed for complaint, reporting processes) and **required** to pass a non-zero **GatewayID** parameter (with **gatewayType "Fast Transfer"`"**).

Implementation of background transaction start (PHP)

```
$data = array(
     'ServiceID' => '100047',
     'OrderID' => '20150723144517',
     'Amount' => '1.00',
     'Description' => 'test bramki',
     'GatewayID' => '71',
     'Currency' => 'PLN',
     'CustomerEmail' => 'test@bramka.pl',
     'CustomerIP' => '127.0.0.0',
     'Title' => 'Test title',
     'ValidityTime' => '2016-12-19 09:40:32',
     'LinkValidityTime' => '2016-07-20 10:43:50',
     'Hash' => 'e627d0b17a14d2faee669cad64e3ef11a6da77332cb022bb4b8e4a376076daaa'
    );

    $fields = (is_array($data)) ? http_build_query($data) : $data;

    $curl = curl_init('https://{gate_host}/test_ecommerce');
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('BmHeader: pay-bm'));
    curl_setopt($curl, CURLOPT_POSTFIELDS, $fields);
    curl_setopt($curl, CURLOPT_POST, 1);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, true);
    $curlResponse = curl_exec($curl);
    $code = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    $response = curl_getinfo($curl);
    curl_close($curl);

    echo htmlspecialchars_decode($curlResponse);
```

Answer - transfer details

In the case of payments of this type, the System generates a set of data needed to make an intra-bank (and therefore fast) transfer to the AP bank account. This data is placed in the response to the start of the transaction, in an xml document.

Payment system response to the start of the transaction (XML)

```
    <?xml version="1.0" encoding="UTF-8"?>
        <transaction>
                <receiverNRB>47 1050 1764 1000 0023 2741 0516</receiverNRB>
                <receiverName>Autopay</receiverName>
                <receiverAddress>81-718 Sopot, ul. Powstancow Warszawy
6</receiverAddress>
                <orderID>9IMYEH2AV3</orderID>
                <amount>1.00</amount>
                <currency>PLN</currency>
                <title>9IMYEH2AV3  - weryfikacja rachunku</title>
                <remoteID>9IMYEH2AV3</remoteID>
                <bankHref>https://ssl.bsk.com.pl/bskonl/login.html</bankHref>
                <hash> fe685d5e1ce904d059eb9b7532f9e06a64c34c1ea9fcf29b62afefdb7aad7b75
</hash>
        </transaction>
```

List of returned parameters for the response

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	receiverNRB	YES	string{32}	Account number of the recipient of the transfer (AP).
2	receiverName	YES	string{1,100}	Name of the recipient of the transfer (AP).
3	receiverAddress	YES	string{1,100}	Address details of the recipient of the transfer (AP).
5	orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
6	amount	YES	amount	Transaction amount. A full stop - '.' - is used as decimal separator. Format: 0.00; maximum length: 14 digits before the decimal point and 2 after the decimal point. **NOTE:** The permissible value of a single Transaction in the Production System is min. 0.01 PLN, max. 100000.00 PLN (or up to the Bank's individual single Transaction limit for an intra-bank transfer).
7	currency	YES	string{1,3}	Transaction currency.

Hash order	name	required	type	description
8	title	YES	string{1,140}	The full title of the transfer (ID together with the Description field from the start of the transaction).
9	remoteID	YES	string{1,20}	The unique transfer identifier assigned in the AP System.
10	bankHref	YES	string{1,100}	The login address in the online banking system, which can be used to create a 'Go to bank' button.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

NOTE: The above information should be used to display transfer data and redirect the user to the bank's login page.

BLIK 0 OneClick payment

Description of BLIK 0 OneClick payments

This is a dedicated solution for BLIK payments, allowing you to payment without entering your BLIK code (and without having to leaving the Website). Its successful initiation in the System causes automatic activation/awakening of the bank's mobile application and presenting the transaction to the User for confirmation.

Potential benefits:

- making available the first convenient and secure payment method in mCommerce that does not require a card number opens this segment to new customers,
- better customer shopping experience - pays faster and more conveniently,
- shopping frequency and customer value over time - customers are more willing to and more often buy from those shops where the shopping process is more convenient,
- conversion rate - the service has greater control over the process of purchase and payment process (the customer does not abandon it), the risk of basket loss,
- fast transaction decision - in a short time the transaction is subject to authorisation, refusal or cancellation,

- The service has the possibility to analyse the very stage of making the payment.

The condition for BLIK 0 OneClick to be made available to the Customer is to have been authorised on the Service (having an account and having previously logged into it). If, during a previously executed BLIK payment, together with other payment information, the Service has sent a dedicated UID Alias (description of the parameters **BlikUIDKey** and **BlikUIDLabel** in another part of the document), and the Customer, while confirming the payment in the mobile application indicated that he or she wished to remember the shop, this resulted in a permanent association (typically for a period of 2 years) of the Service Customer with his/her application, i.e. Alias UID registration. Its subsequent use will result in authorisation of the transaction without entering the code.

Calling BLIK 0 OneClick payments

When selecting a BLIK Payment Channel, it is recommended not to force the user to enter a BLIK code. Instead, it is advisable to display the 'I want to enter my BLIK code BLIK' hyperlink under the "Buy and pay" button to enable entering the code in the first attempt (in case the Customer would like to make a BLIK payment from a different mobile application than the one in which he/she has previously saved a given Service).

The Service should perform the Pre-Transaction, paying particular attention to the following on:

- specifying the parameter **GatewayID** = 509 - indicating the payment channel BLIK,
- providing **BlikUIDKey** and **BlikUIDLabel** parameters - indicating BLIK 0 OneClick Alias UID (User ID) required by BLIK 0 OneClick. user)
- providing the **AuthorizationCode** parameter - if the customer provided the code BLIK,
- providing **BlikAMKey** parameter - if the Customer specified a label of the of the bank's mobile application from the list presented on the Website,
- handling possible responses to pre-transaction, including handling 'Response - no continuation' and BLIK-specific errors 0 OneClick:

a) error of many mobile applications of the bank (**confirmation=NOTCONFIRMED** and **reason=ALIAS_NONUNIQUE**) - displaying the list of labels returned in the pre-transaction aliases list (key + label pairs contained in the **BlikAMList** structure), in order to retrieve the selected key and provide it in the **BlikAMKey** parameter of the next pre-transaction attempt

b) authorisation errors (**confirmation=NOTCONFIRMED** and **reason** with one of the values: **ALIAS_DECLINED, ALIAS_NOT_FOUND, WRONG_TICKET, TICKET_EXPIRED, TICKET_USED**) - display the Blik Code field, in order to retrieve it and provide it in the **AuthorizationCode** parameter of the next pre-transaction attempt

Google Pay

Description

Google Pay is an instant and intuitive payment system from Google. It allows the user to complete the payment process without completing a card form, as the card details are stored securely on the company's servers.

Google Pay is a product that allows encrypted data of the customer's payment card to be obtained allowing it to be debited.

In order to pay via Google Pay, you need to save your payment card to your Google account, using any Google platform (e.g. buying apps on Google Play) or directly on the Google Pay.

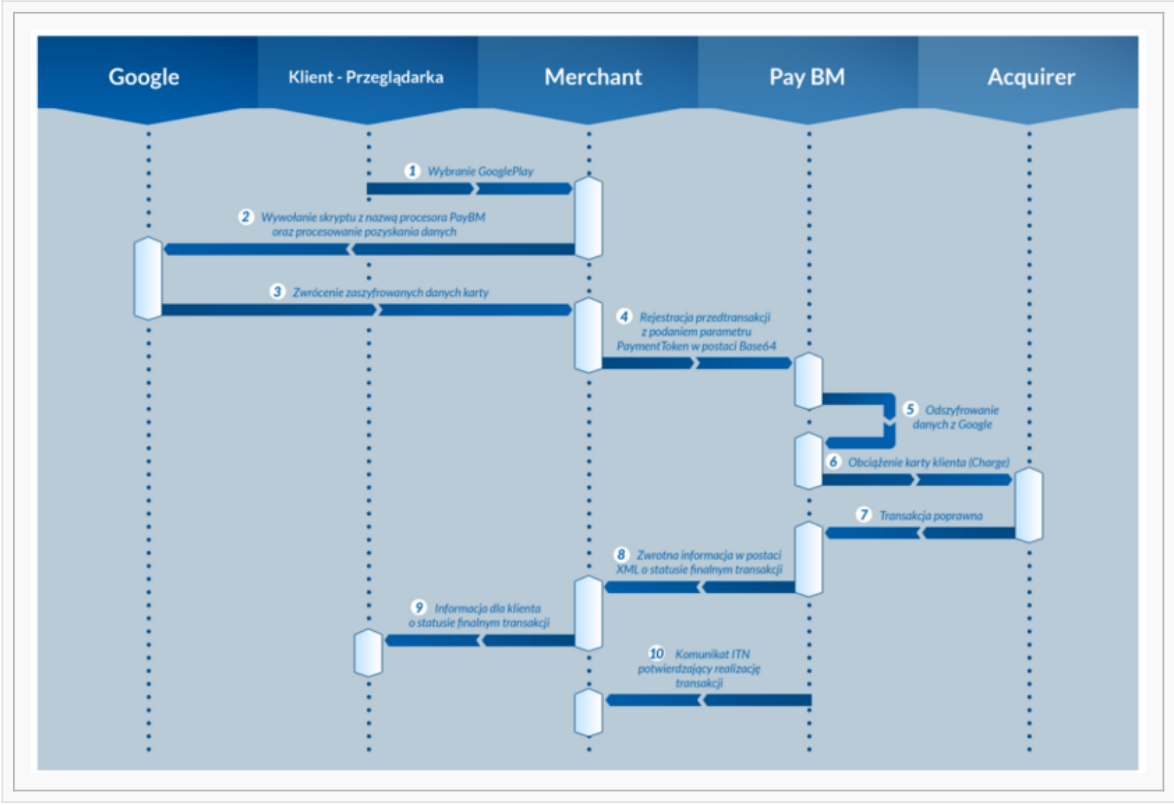
> **NOTE:** The service requires the prior signing of a contract with the card operator. Please contact the Autopay Business Department for details.

Communication scheme

After clicking on 'Pay with Google Pay', a Google Pay form appears on the shop page. In it, the customer confirms his Google account and the card he intends to pay with (he can also change to another card or add a new one at this stage). The script transmits the encoded card data in the background via the **postMessage** function, then the shop has to accept and encode it via the base64 function and finally send it in the **PaymentToken** parameter along with the other parameters (transaction data).

On its website, the shop must call up the script provided by Google with the payment processor's details changed.

> **TIP:** Details in Google developer documentation.



Detailed scheme of communication and data exchange

Google Pay transaction registration

1. The shop on its website must send a request to the AP Online Payment System to retrieve the data needed to process Google Pay (**paybmApiResponse**).

> **TIP:** An example of how to send a request is available at GitHubie Autopay.

2. The shop must then call the script provided in the parts of the Google Developer Documentation Tutorial, containing:

a) Payment processor details altered:

```
const tokenizationSpecification = {
        type: 'PAYMENT_GATEWAY',
        parameters: {
                'gateway': 'bluemedia',
                'gatewayMerchantId': paybmApiResponse.acceptorId
        }
};
```

b) Data returned by the AP Online Payment System transferred in the object **PaymentDataRequest.merchantInfo**:

```
PaymentDataRequest.merchantInfo = {
        merchantId: paybmApiResponse.merchantId,
        merchantOrigin: paybmApiResponse.merchantOrigin,
        merchantName: paybmApiResponse.merchantName,
        authJwt: paybmApiResponse.authJwt,
};
```

3. After clicking on 'Pay with Google Pay' on the shop page, a Google Pay form appears on the shop page. In it, the customer confirms his Google account and the card he intends to pay with (he can also change to another card or add a new one at this stage). A script in the background transmits the encoded card data, which the shop has to accept and then encode with the Base64 function and send in the **PaymentToken** parameter together with the rest of the transaction start parameters (i.e. the transaction data of the AP Online Payment System - details are described in section Starting a transaction with additional parameters).

> **TIP:** A complete example of integration with Google Pay is available on GitHubie Autopay.

Additional information

In order to maintain the aesthetic integrity of the design used on the website and mobile app, please use the guidance provided in the parts of the Brand Guidelines of the development documentation

Google for style descriptions and buttons for web pages, and for parts Developer documentation tutorial Google, where you will find the information needed for the development of the mobile application.

Apple Pay

Apple Pay implementation on the shop's website.

> Request to contact payment infrastructure product - IT support needed.

1. Creation of an account by the Partner and obtaining a payment processing certificate in accordance with the document **Configure Apple Pay (iOS, watchOS)**

- **communication certificate** – for so-called presentation – *merchant identifier*

- **debit certificate** – *payment processing certificate*

2. Web implementation in accordance with document **Apple Pay on the Web**

3. Preparation of 2 endpoints on the Partner's side, on a domain registered with Apple (using 2 certificates from Apple):

- for session start

- to charge the customer based on the token from Apple

> TIP: The Safari (the client's browser) request the session to the endpoint (mentioned above) they then go to us - we return the session.

4. Apple Pay processing

- As part of your service registration with Apple, generate your certificate *merchant identity.*
- Generate a *payment processing* certificate based on the certificate provided by the AP CSR

> NOTE: AP CSR certificates for acceptance and for production differ).

- After using it in the Apple registration process, provide the AP with a certificate signed by Apple and send it via formularz Autopay

  TIP: The client should provide the country, city, website domain, email of the contact person.

- As part of the payment processing on the Partner site, start an Apple API session.

- Then return the Autopay token in the PaymentToken start parameter.

NOTE: Decryption of the token is the responsibility of the AP.

Payment token format: a slice of an object in json format that the ApplePay api returns:

```
EncryptedPaymentData {
        String version;
        String data;
        String signature;
        Header header;
}
Header {
        String ephemeralPublicKey;
        String publicKeyHash;
        String transactionId;
        String applicationData;
}
```

NOTE: When sending an ApplePayPaymentRequest, you need to populate the applicationData field with the Base64-encoded orderId value, as described in the document applicationData.

Autopay widget (WhiteLabel model)

Partners who would like to embed some of the transaction starts directly on their site / in their shopping cart (in the so-called WhiteLabel model) can do so by integrating the Autopay Widget. Currently, the Autopay Widget supports the collection of card data (within the PaywayId 1500/1503) or Visa Mobile starts (PaywayId PaywayId 1523)

IMPORTANT! The Partner is not entitled to store Card data (in particular card number, CVC security code, CVV2), with the exception of the parameters transferred when processing automatic payments by the AP, as described in this section.

IMPORTANT! The Partner website in which the Autopay widget functionality is used must be encrypted and the HTML IFRAME with the widget must be embedded in an HTTPS address with the use of TLS.

IMPORTANT! The partner undertakes to submit to AP, in electronic form,
the following documents:
a) on a one-off basis (prior to the conclusion of the Contract): a completed SAQ-A PCI questionnaire (Section 2); The document will be provided by AP or is available for download on the website: https://www.pcisecuritystandards.org
b) On a quarterly basis: the result of the quarterly PCI ASV audit including a scan of external (public) IP addresses/networks/domains - IPv4 and/or IPv6. Such audit must be conducted by one of the authorised contractors listed at:
https://www.pcisecuritystandards.org/assessors_and_solutions/approved_scanning_vendors

Autopay WidgetJS SDK

You will need to use the Autopay WidgetJS SDK to embed and communicate with the Autopay widget. In a nutshell, it will come down to embedding the HTML IFRAME with the widget and configuring the JS SDK to handle the messages (events) produced when the Cardholder interacts with the widget. The final message is an event with a status of `FORM_SUCCESS` including `paymentToken` necessary for the backend start of transactions on the API of the AP Online Payment System.

Embedding the SDK

Below are examples of how to easily embed and syndicate a widget, using the Autopay WidgetJS SDK, for both card and VisaMobile channels.

The Autopay WidgetJS SDK is available at `widget-new/widget-communication.min.js` once it has been placed in the `<head>`

```
<script
src="?r=quot;https://testcards.autopay.eu/widget-new/widget-communication.min.js"></scri
pt>
```

You gain access to the object `WidgetConnection`

```
var widgetConfigObject = { ... };
var widget = new WidgetConnection(widgetConfigObject)
```

which, when supplemented with a configuration in the form of a JSON object, will enable full communication with the Autopay API and, as a result, ensure that you receive an event with a status of `FORM_SUCCESS` from `paymentToken'em`.

Examples of configurations

Example of configuration for card data

Configuration:

```
{ language: 'pl', amount: 1.23, currency: 'PLN', serviceId: 123456 }
```

PaymentToken returned:

```
{status: 'FORM_SUCCESS', message: 'ey...9', id:
'OGFlZTYyYTMtN2U2OS00MTU1LTgyNDctNmMwMGI2NjE5ZDQy'}
```

Example of configuration for VisaMobile data

Configuration:

```
{ language: 'pl', amount: 1.23, currency: 'PLN', serviceId: 123456, merchantName:
'ShopName' }
```

PaymentToken returned:

```
{status: 'FORM_SUCCESS', message: 'ey...9', prefix: '48', phoneNumber: '666666666', id:
'OGFlZTYyYTMtN2U2OS00MTU1LTgyNDctNmMwMGI2NjE5ZDQy'}
```

Detailed discussion of the configuration of the WidgetConnection object

Language

Determines the language version of the widget in which it will be presented.

Field name: `language` Format `string` Values: default `pl`, the following languages are currently supported: `cs`, `de`, `el`, `en`, `es`, `fr`, `hr`, `hu`, `it`, `pl`, `ro`, `se`, `sk`, `sl`, `uk`

Transaction amount

Transaction amount

Field name: `amount` Format `float` Values: an amount written in float format, e.g: "1,23 PLN" should be `1.23`

amount: 1.23, currency: 'PLN', serviceId: 123456, merchantName: 'ShopName'

Currency of transaction

Currency of transaction

Field name: `currency` Format `string` Values: default PLN, other currencies in a format compatible with the current service configuration

Service number

Service number received from Autopay (dependent on the development environment)

Field name: `serviceId` Format `integer` Values: usually six digits

Recurrence type (for cards only)

Indication of type of recurrence initiation (Only for channel 1503 related to recurrence initiation)

Field name: `recurringAction` Format `string` Values: 'INIT_WITH_REFUND', 'INIT_WITH_PAYMENT'

Shop name (only for VisaMobile)

Name displayed to the user in the VisaMobile notification in the bank's mobile application (Only for VisaMobile channel 1523)

Field name: `merchantName` Format `string` Values: Shop name

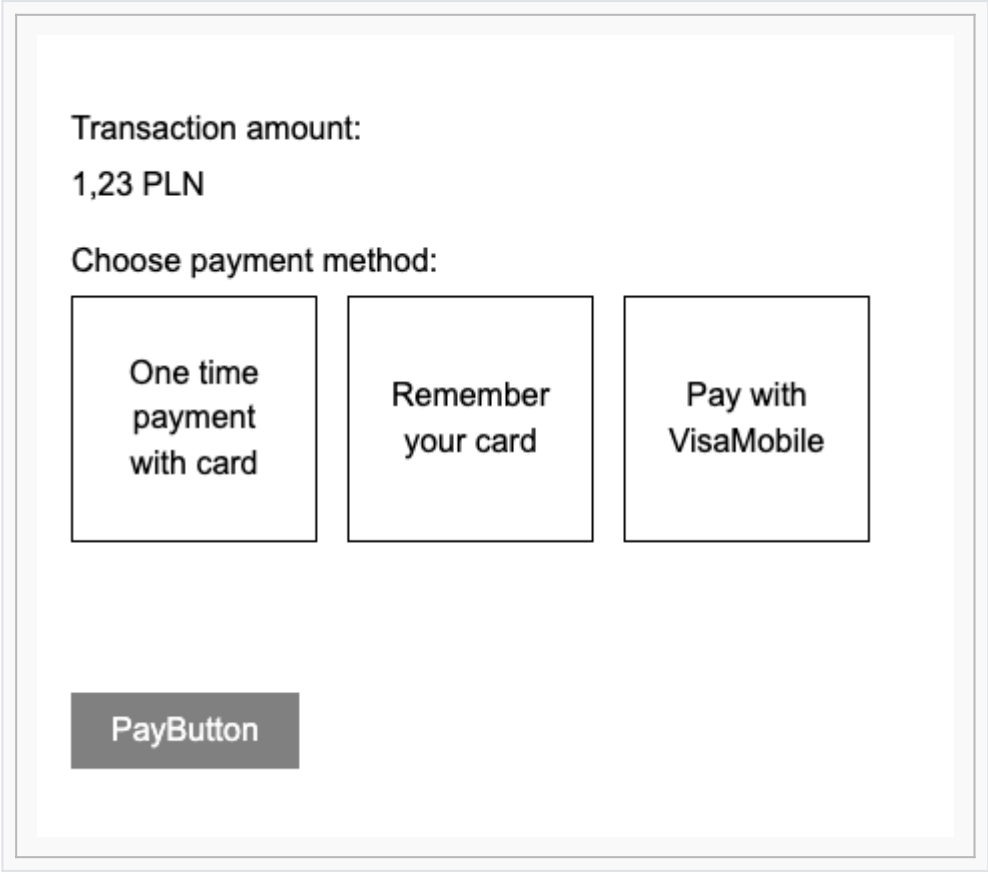
Card Widget - Example of implementation on a partner website

IMPORTANT! The following example of HTML code was created for illustrative purposes. In order to actually run it on your local computer, the following HTML file must be placed under some local domain (any, it can be `test.local`).
This HTML cannot be fired in the browser as a local file because theJS events exchanged between the IFRAME and the page are verified for domain consistency (and so some domain must be present).

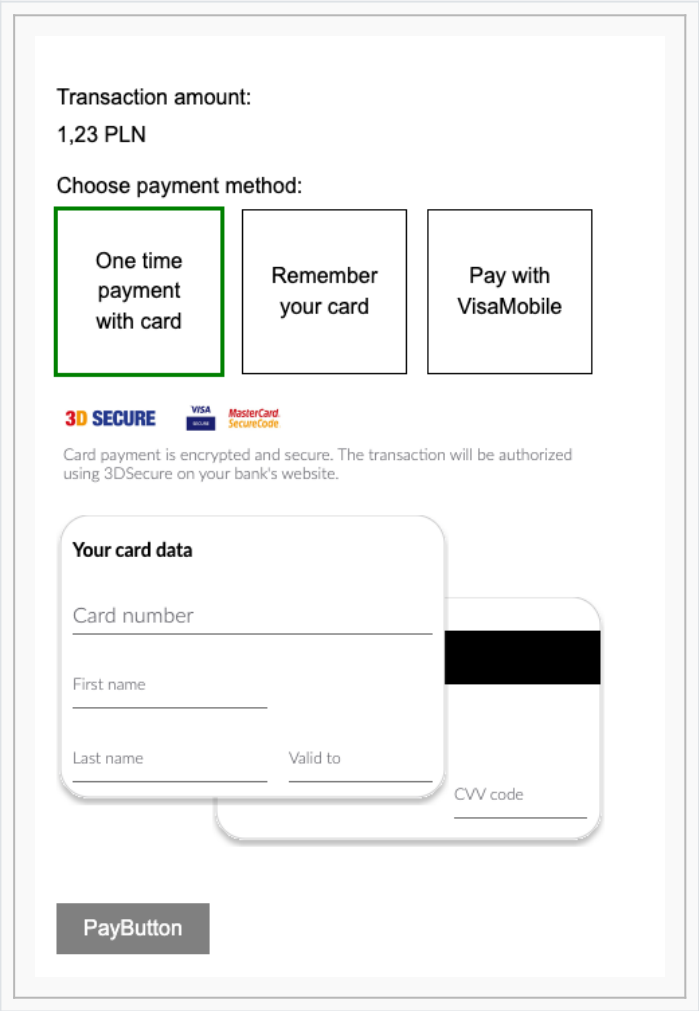
The following page is intended to mimic Front Merchant, showing what elements need to be implemented in order to integrate with the Autopay Widget.

In the browser, the following sample page consists of three sections:

- the top section contains a selection of specific payment channels
- the middle section contains the place where the HTML IFRAME will be embedded, to which the address of the widget (visamobile or standard card if necessary) will be placed
- the bottom section contains a button (inactive by default) `PayButon` bundled with the JS SDK to control the start of the process in the widget (in this example, the button only activates when it receives a message that the validation is correct and the data needed to start the process is complete)



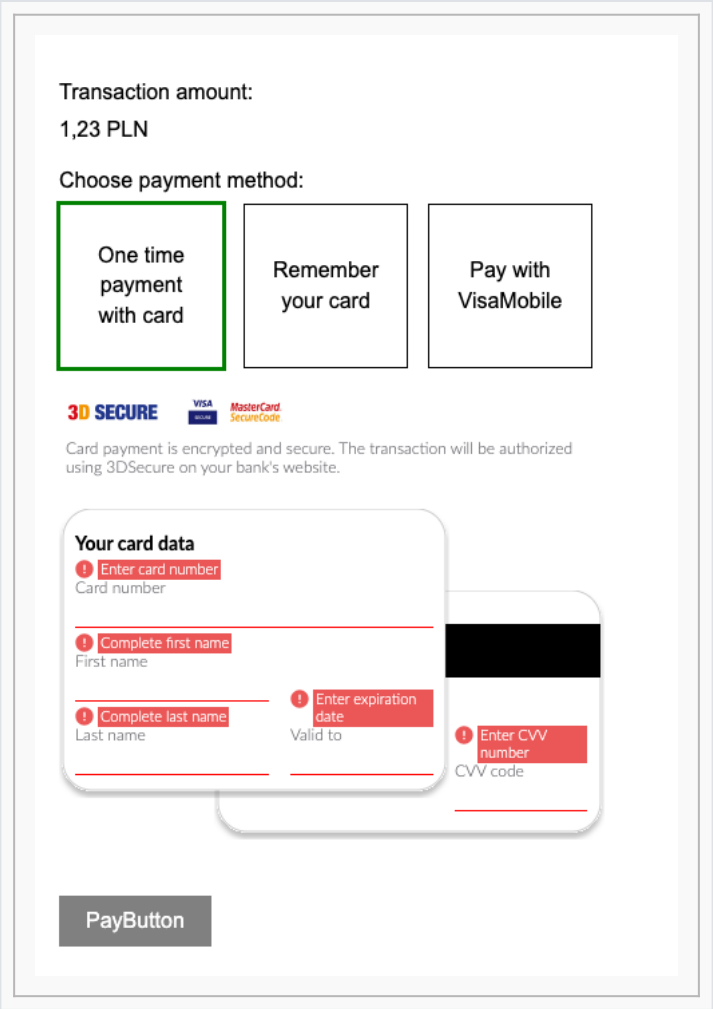
When a card channel is selected (payway: 1500 or 1503), a dedicated card form view (based on HTML IFRAME) will load. When full, valid card details are entered in the widget, (thanks to validation events) the "Pay" button on the Merchant front-end will be activated.



TIP: As you can see in the example, it is also possible to support the VisaMobile channel in the WhiteLabel model. The implementation/layout is analogous to the card widget therefore the code below already contains both cases.

Validation and completion of data

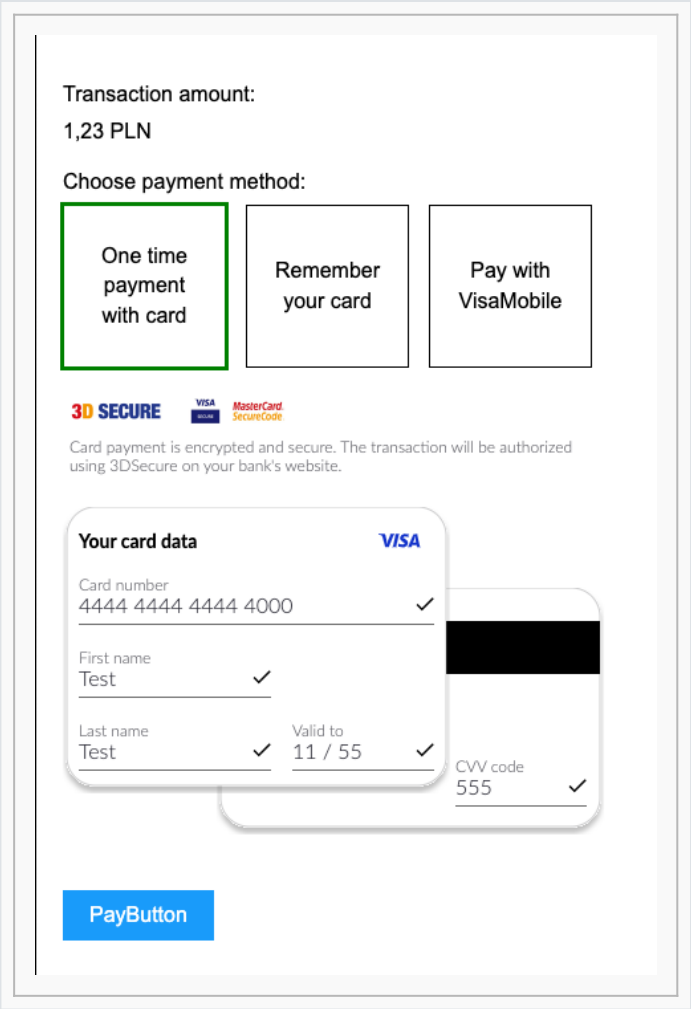
The Autopay Widget JS SDK receives events from the widget when it enters the card data `VALIDITY_STATUS` with value `valid: false`



Once we have the full card details, the last event will be `VALIDITY_STATUS` with value `valid: true`

```
{status: 'VALIDITY_STATUS', message: null, valid: true, id: 'M2Zl...mU2'}
```

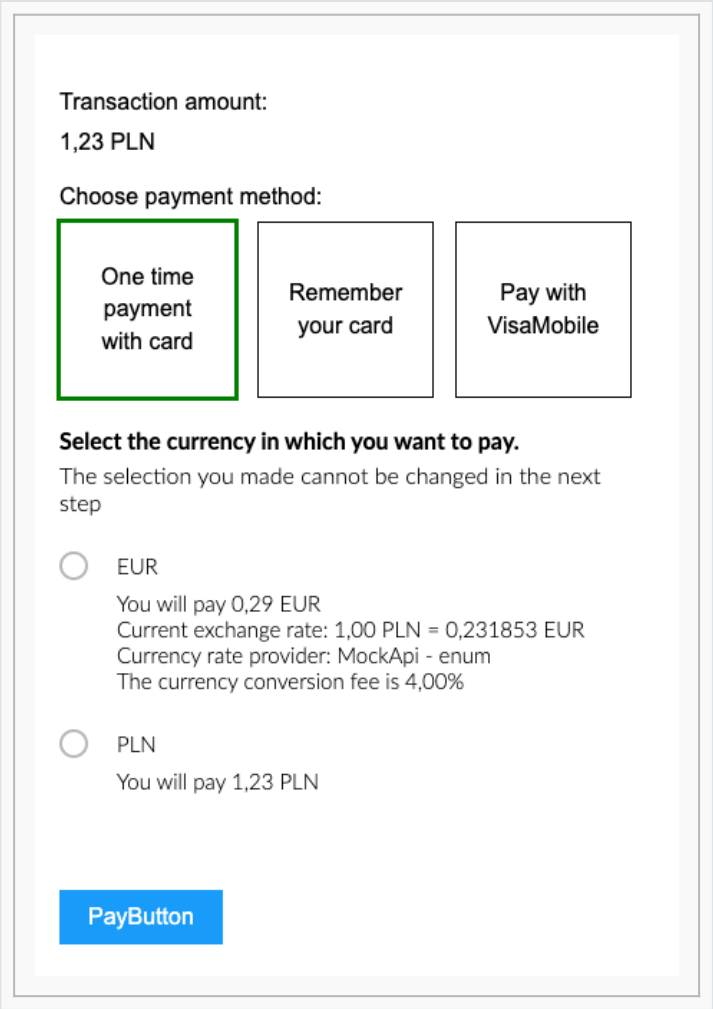
The activation of the button can be based on this event `PayButton`



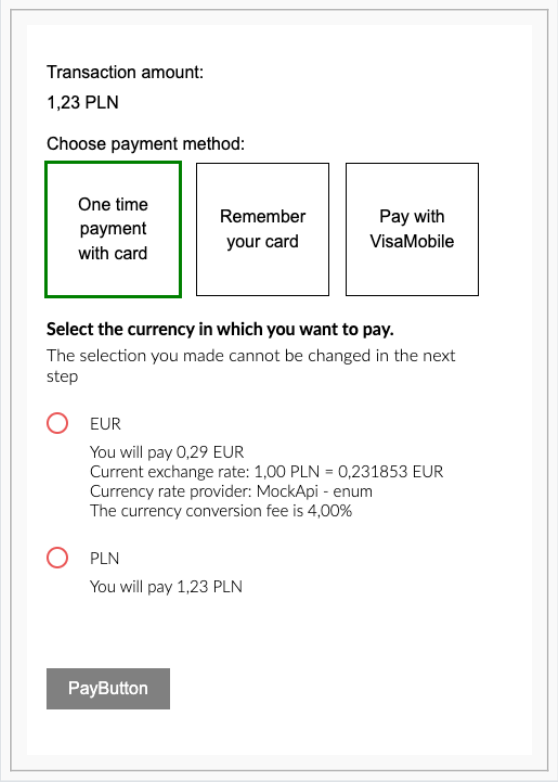
TIP: On the test environment, card payments are based on a 3ds mock and an authorisation mock. Dedicated test card numbers correspond to the different scenarios. A full list of test cases can be found in a separate appendix.

DCC screen

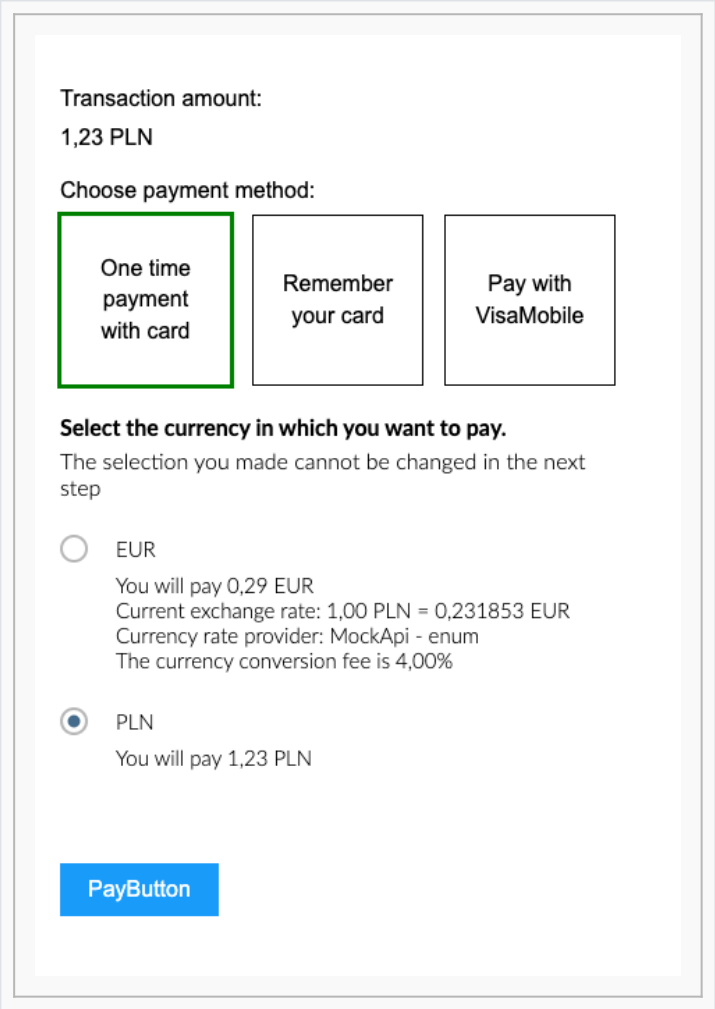
If the scenario and card meet the conditions for obtaining a DCC offer, an additional screen will appear with a currency conversion proposal for the cardholder



The cardholder can choose to use the card debit in its native currency or leave the original currency. Validation also occurs on this screen.



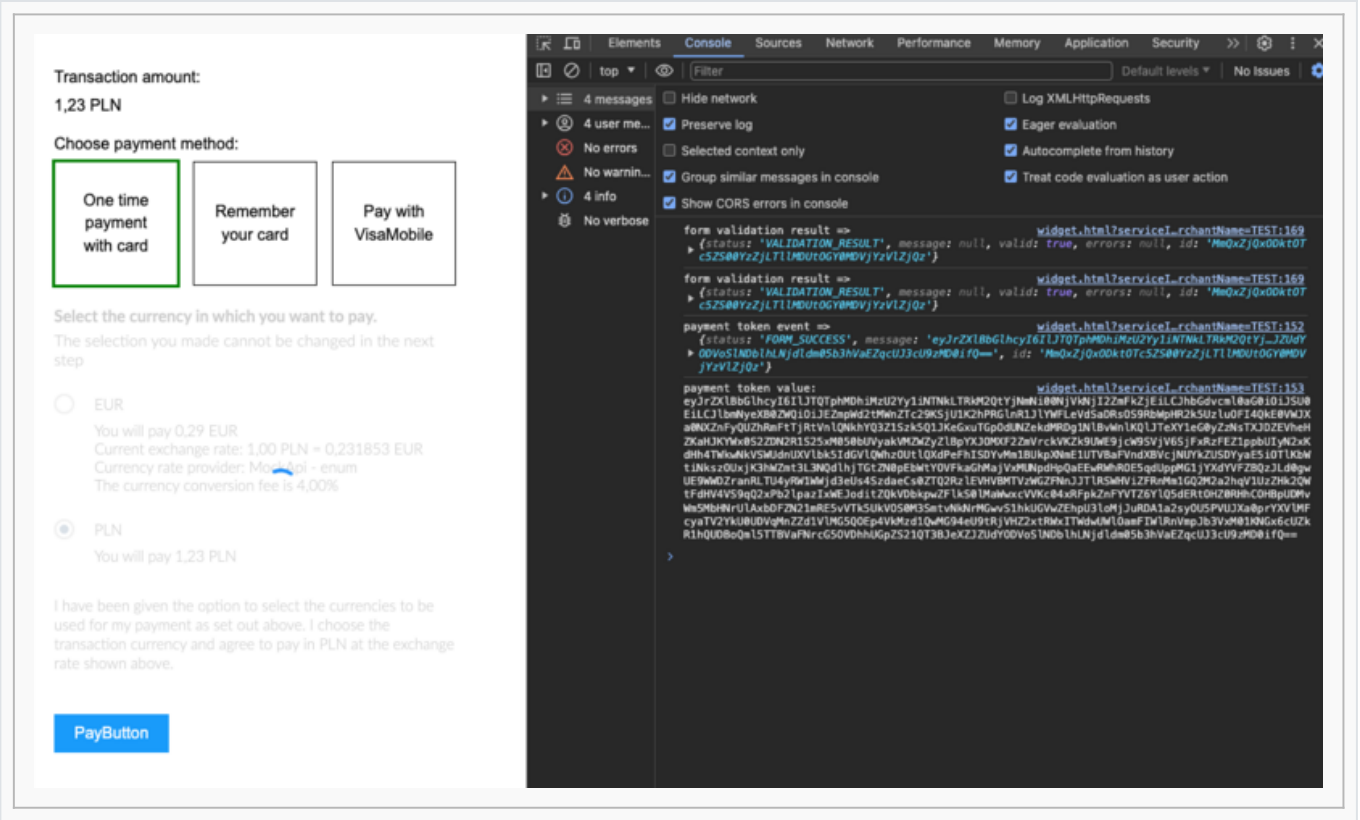
The selection of the Cardholder's currency will not affect the Merchant and the original amount of the transaction itself, but will affect the amount the card will be charged. If the Cardholder does not wish to take advantage of the DCC currency conversion offer, he selects the original currency (which in this case is PLN).



Obtaining a token

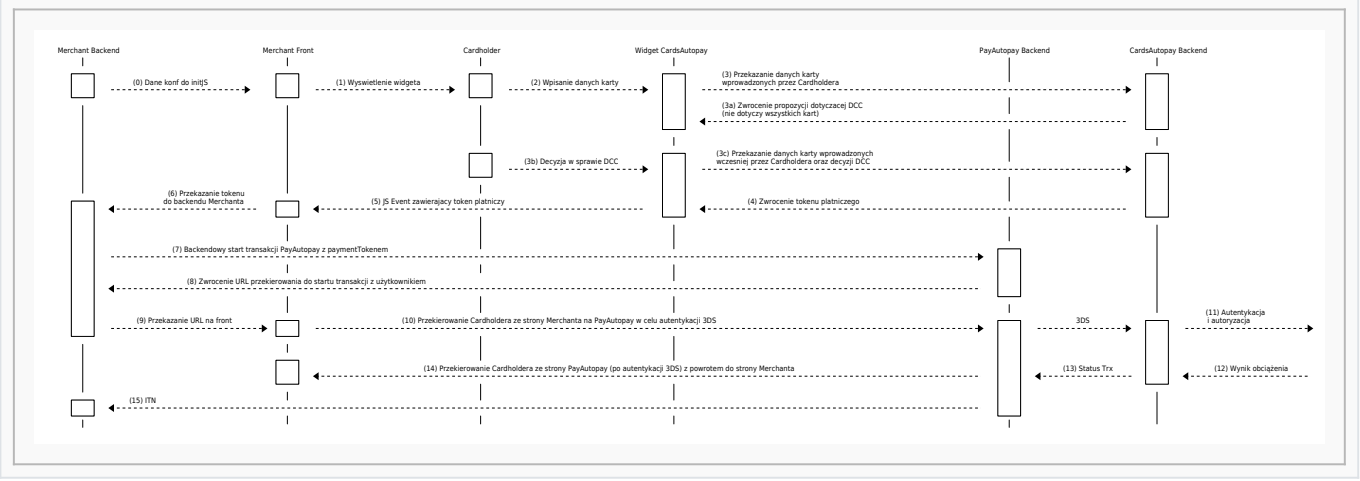
The button should be linked to the Autopay Widget JS SDK so that clicking it triggers a call to the `widget.sendForm();` method in the `WidgetConnection` object Which, ultimately, will result in a `FORM_SUCCESS` event, i.e. getting the paymentToken value (in the `message` field).

```
{status: 'FORM_SUCCESS', message: 'eyJ...n19', id: 'M2Z...ZmU2'}
```



Card Widget - Detailed scheme of communication and data exchange

The following is a detailed diagram of the communication between Merchant, Cardholder and Autopay payment systems in the case of so-called WhiteLabel integration (i.e. using a card widget)



Secure transfer of card data to the Autopay system and full transaction flow:

- (0) Transfer of configuration data initiating the embedding of the Widget to the frontend.
- (1) Display of the Widget's cardformat embedded on the Merchant frontend (card data is not provided on the Merchant frontend but on the Autopay widget frontend)
- (2) Start by the cardholder entering the card data.

- (3) Transmission of card data using a TLS connection secured with an Extended Validation certificate to the CardsAutopay backend. ** Only applicable if DCC can be offered ** (3a) Returning the details of the DCC conversion proposal ** (3b) Cardholder decides whether to use DCC ** (3c) Transmission of card details previously entered by Cardholder and DCC decision follows
- (4) WidgetJS receives the `paymentToken' value from CardsAutopay and sends it to the Merchant Front (via JS).
- (5) The Merchant Front receives the payment token from the Widget via a JavaScript event.
- (6) The Merchant Front passes the payment token to the Merchant Backend.
- (7) There is a backend start of Pre-transaction with paymentToken received earlier from the frontend.
- (8) Autopay API returns the continue-transaction URL which will be used to redirect to the start of the transaction with the user
- (9) Merchant's backend forwards the redirect URL to the Merchant frontend.
- (10) Cardholder redirection from Merchant's website to PayAutopay for 3DS authentication.
- 3DS verification follows (depending on the bank's decision, this may be full or simplified verification)
- (11) When the Cardholder ‘returns’ from the 3DS, the authentication result is completed/collected and authorised.
- (12) Autopay receives the debit result.
- (13) The transaction status is transmitted to the Online Payment System (in the background).
- (14) Cardholder redirection from the PayAutopay website (after 3DS authentication) back to Merchant's website occurs.
- (15) Asynchronously to the Merchant's backend comes the message ITN with the status of the transaction ** (in the case of a transaction initiating a recurrence, the Merchant's Backend will also receive an additional message RPAN)

Notification of the launch of an automatic payment (RPAN)

Below is an example of a simple HTML/JS implementation using the VisaMobile widget (and card widget)

```
{ 'status': 'FORM_SUCCESS', 'message': 'eyJrZ...', ... }
```

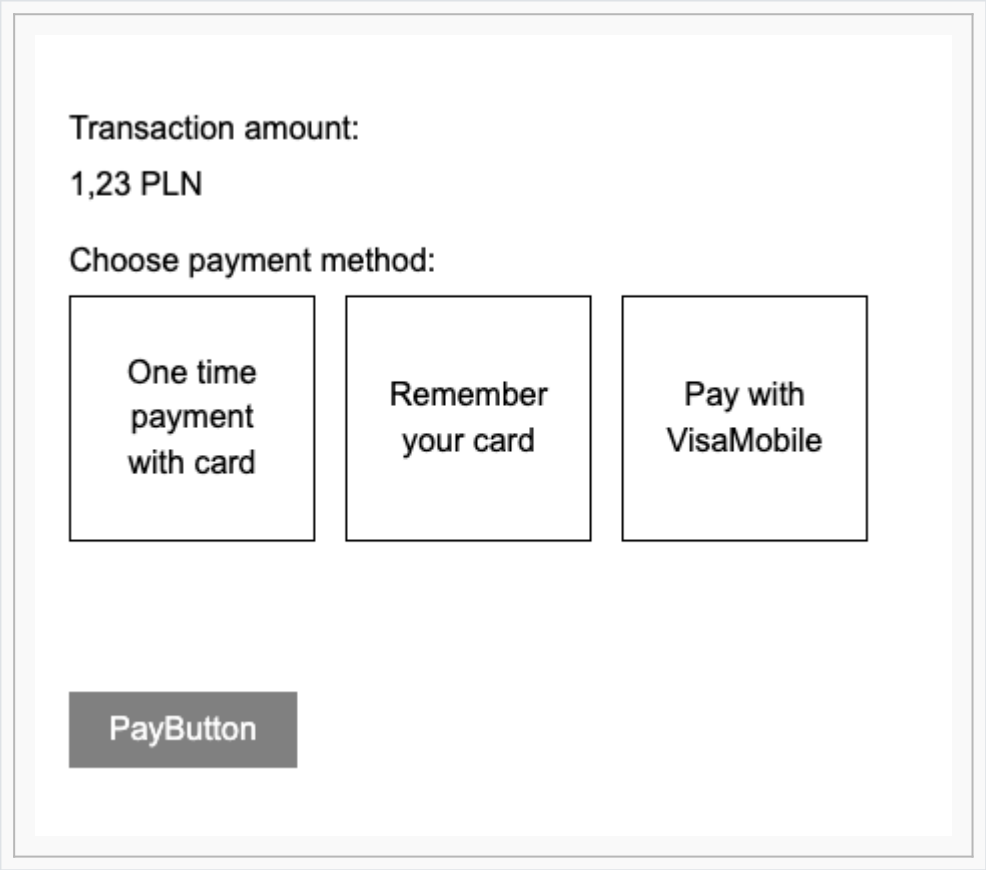
Crucial to this integration is the place in the JS code that is responsible for receiving events, especially the event with status `FORM_SUCCESS`, as it contains in the `message` field the value of the paymentToken that the merchant needs to pass to its backend in order to complete the parameters for the Autopay API to enable the payment to start in Autopay.

Example page

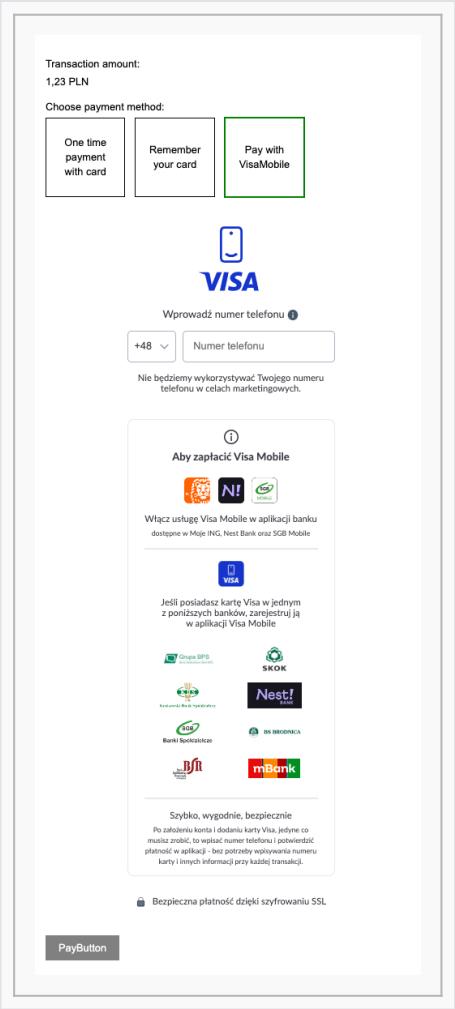
In the browser, the following sample page consists of three sections:

- the top section contains icons/buttons for specific payment channels (using graphical representations from Autopay)
- the middle section contains the placeholder for the HTML IFRAME, into which the widget address (visamobile or standard card as required) will be inserted if necessary

- the bottom section contains the (inactive by default) `PayButon` button, bundled with the JS SDK, which controls the start of the process in the widget (in this example, the button only becomes active when it receives a message about correct validation and obtaining all the data needed to start the process)



When a channel dedicated to VisaMobile is selected, a dedicated view (based on HTML IFRAME) is displayed, where entering the full phone number (thanks to validation messages) results in the activation of the 'Pay' button.

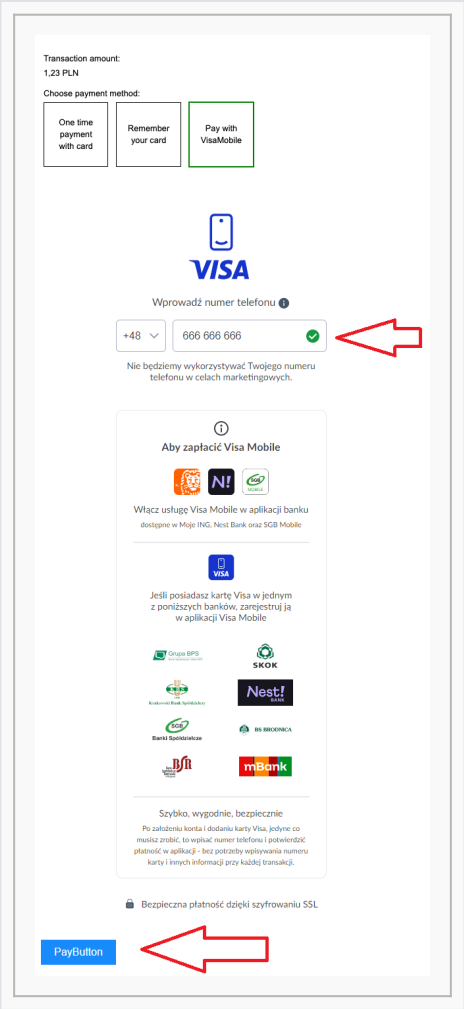


Validation and completion of data

When entering a phone number, the Autopay Widget JS SDK receives a `VALIDITY_STATUS` event from the widget with the value `valid: false` When the full phone number is obtained, the last event will be `VALIDITY_STATUS` with the value `valid: true`.

```
{status: 'VALIDITY_STATUS', message: null, valid: true, id: 'M2Zl...mU2'}
```

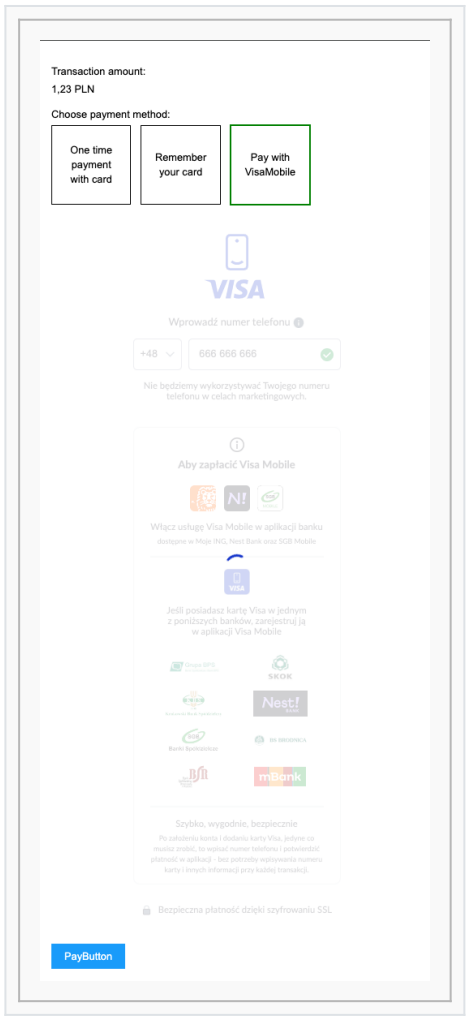
The activation of the button can be based on this event `PayButton`



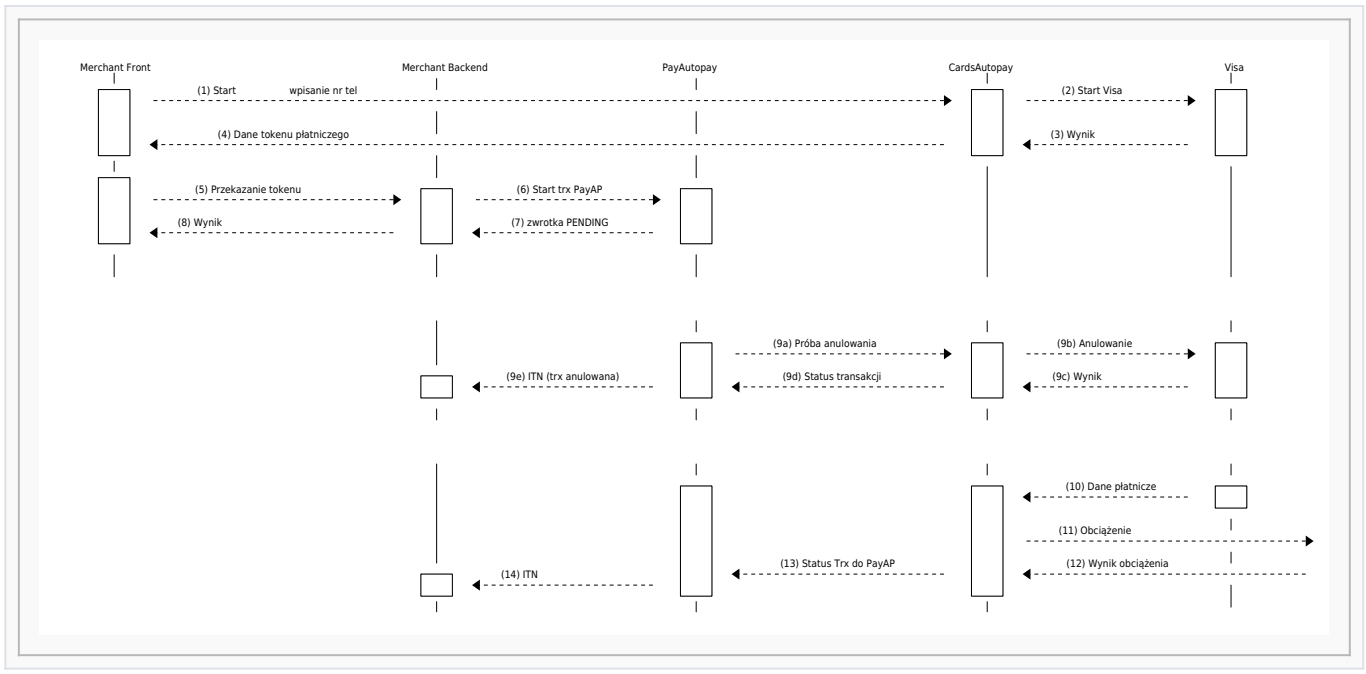
Obtaining a token

The button should be linked to the Autopay Widget JS SDK so that clicking it triggers a call to the `widget.sendForm();` method in the `WidgetConnection` object Which, ultimately, will result in a `FORM_SUCCESS` event, i.e. obtaining the paymentToken value.

```
{status: 'FORM_SUCCESS', message: 'eyJ...n19', prefix: '48', phoneNumber: '666666666',
id: 'M2Z...ZmU2'}
```



Visa Mobile widget - Detailed scheme of communication and data exchange



Start transaction:

- (1) Start by entering your phone number in the VisaMobile widget.
- (2) Start transaction in VisaMobile
- (3) Return from VisaMobile
- (4) Return from the widget to Merchant (using JS) the generated `paymentToken`.
- (5) The Merchant front end passes the payment token to the Merchant backend.
- (6) Backend starts Pre-transaction with paymentToken with **paymentToken** received earlier.
- (7) The return gets an XML PENDING response right away (because it is processed in the background).
- (8) The Merchant front end presents a screen waiting for the result.

Possible cancellation of the transaction (from the PayAutopay paywall):

- (9a) If the user does not get the notification in the mobile app or changes his/her mind he/she has the option to cancel the transaction from within the paywall
- (9b) A cancellation request is sent from the CardsAutopay system to Visa.
- (9c) The CardsAutopay system receives the cancellation result from Visa.
- (9d) The cancellation result is received by the PayAutopay system and presented to the user on the PayAutopay paywall.
- (9e) In parallel, an ITN is sent (with negative status) to Merchant

Load and return the result:

- (10) Waiting for payment token data from Visa (if the VisaMobile customer confirms in the mobile app that they want to pay with a specific card for a particular order)
- (11) A debit order follows.
- (12) A positive or negative debit result is received
- (13) Transaction status is sent to the PayAutopay gateway.
- (14) The Autopay gateway sends the transaction result to Merchant in the form of ITN

Description of the sample HTML JS code (Card Widget and VisaMobile)

The following code was used to generate the example integrations mentioned above in the sections with examples of the implementation of the Card Widget as well as the Visa Mobile Widget

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Autopay Widget Integration Example</title>
    <script
src="?r=quot;https://testcards.autopay.eu/widget-new/widget-communication.min.js"></scri
pt>
    <style>/* omitted from the example */</style>
</head>
<body>
<div>
  <form onsubmit="submitForm(event)" novalidate>
   <div class="form-group"><p>Transaction amount:</p><span>1,23 PLN</span></div>

    <!-- example implementation of a merchant-side payment channel selection mechanism -
->
    <p>Choose payment method:</p>
    <ul>
```

```
      <li onclick="setPayway(event, 1500)">One time payment with card</li>
      <li onclick="setPayway(event, 1503)">Remember your card</li>
      <li onclick="setPayway(event, 1523)">Pay with VisaMobile</li>
    </ul>

    <!-- the place where the HTML IFRAME with the widget will be injected -->
    <div class="form-group" id="iframe-wrapper">
      <iframe id="iframe"></iframe>
    </div>

    <!-- call to action button in the widget -->
    <button type="submit" id="button" disabled="disabled">PayButton</button>
  </form>
</div>
<script type="text/javascript">
window.addEventListener("load', () => {
    // auxiliary variables (only for the purpose of the example)
    var currentPayway = null;
    var widget = null;

    // example configurations depending on the devlopers' environment (only for the
purpose of the example)
        var AUTOPAY_CARDS_DOMAIN_ENV_PROD = 'https://cards.autopay.eu';
        var AUTOPAY_CARDS_DOMAIN_ENV_TEST = 'https://testcards.autopay.eu';

        var MERCHANTS_SERVICE_ID_ENV_PROD = 903555;
        var MERCHANTS_SERVICE_ID_ENV_TEST = 903555;

    // auxiliary method to support payment channel selection and widget embedding
    function setPayway (event, paywayId) {
        if (currentPayway === paywayId) {
            return;
        }
        currentPayway = paywayId
        removeWidget();
        disableSubmitButton();
        markActiveIcon(event);

        if (paywayId === 1500) {
            startWidget('/widget-new/partner'   , { language: 'en', amount: 1.23,
currency: 'PLN', serviceId: MERCHANTS_SERVICE_ID_ENV_TEST });
            return
        }
        if (paywayId === 1503) {
            startWidget('/widget-new/partner'   , { language: 'en', amount: 1.23,
currency: 'PLN', serviceId: MERCHANTS_SERVICE_ID_ENV_TEST, recurringAction:
'INIT_WITH_REFUND' });
            return
        }
        if (paywayId === 1523) {
            startWidget('/widget-new/visamobile', { language: 'en', amount: 1.23,
currency: 'PLN', serviceId: MERCHANTS_SERVICE_ID_ENV_TEST,  merchantName: 'ShopName' });
                        return
        }
    }

    // auxiliary method (only for the purpose of the example) setting the border on the
selected payment channel
    function markActiveIcon (event) {
        var currentActive = document.querySelector('ul li.active');
        if (currentActive) {
            currentActive.classList.remove('active');
```

```
        }
        var newActive = event.target;
        if (newActive.nodeName.toLowerCase() === 'img') {
            newActive = newActive.parentNode;
        }
        newActive.classList.add('active');
    }

    // the main method responsible for embedding the IFRAME with the widget and setting
up communication between it and its object representation WidgetConnection
    function startWidget (widgetVariantUrl, widgetConfig) {
        if (!widgetEvents || !WidgetConnection) {
            return;
        }
        var iframeEl = document.getElementById('iframe');
        iframeEl.src = AUTOPAY_CARDS_DOMAIN_ENV_TEST + widgetVariantUrl;
        widget = new WidgetConnection(widgetConfig)

        widget.startConnection(iframeEl).then(() => {

            // handling the main, final event containing the PaymentToken value
            widget.on(widgetEvents.formSuccess, function (message, eventData) {
                console.log('payment token event =>', eventData);
                console.log('payment token value:', message);
                // here should be a merchant API call to pass the paymentToken (message)
to the merchant backend;    <<<<<<<<<<<<<<<<
            })

            // handling validation events during user/cardholder input in the widget
            widget.on(widgetEvents.validityStatus, function (message, eventData) {
                console.log('form validation status =>', eventData);
                if (eventData.valid) {
                    enableSubmitButton();
                } else {
                    disableSubmitButton();
                }
            })

            // handling the validation event when the user/cardholder enters data in the
widget
            widget.on(widgetEvents.validationResult, function (message, eventData) {
                console.log('form validation result =>', eventData);
                if (eventData.valid) {
                    enableSubmitButton();
                } else {
                    disableSubmitButton();
                }
            })

            // handling the showModal event
            widget.on(widgetEvents.showModal, function () {
                console.log('show modal');
            })
        })
    }

    // an auxiliary method (only for the purpose of the example) setting the removal of
the widget for payment channels other than cards (in the example, there is a 106 PBL
channel)
    function removeWidget () {
        if (!widget) {
            return;
```

```
        }
        widget.stopConnection();
    }

    // auxiliary method (only for the purpose of the example)
    function enableSubmitButton () {
        document.getElementById('button').removeAttribute('disabled');
    }

    // auxiliary method (only for the purpose of the example)
    function disableSubmitButton () {
        document.getElementById('button').setAttribute('disabled', 'disabled');
    }

    // an auxiliary method (for the purpose of the example only) that binds the call of
the active pay button to the sendForm() call in the widget object
    function submitForm (event) {
        event.preventDefault();
        if (!widget || widget.invalid) {
            return;
        }
        disableSubmitButton();
        widget.sendForm();
    }

    window.setPayway = setPayway;
    window.submitForm = submitForm
});
</script>
</body>
</html>
```

Automatic payment

Description of automatic payment

Automatic payments are an extremely convenient and secure way of making recurring transactions. It involves the automatic collection of receivables from the customer on its payment dates. The service must first be activated. In the case of cards, this is done by redirecting the customer to the service activation form. In the case of BLIK while by accepting an automatic payment in the Mobile Application. Upon successful authorisation of such an activation transaction, the AP transmits to the Partner a standard message about the change of status of the transaction (ITN) and a message about the activation of the automatic payment service (RPAN). The RPAN message contains the field **clientHash**with which the partner will identify the specific automatic payment during subsequent debits and deactivation of the service.

NOTE: Before initiating recurring transactions, the Partner is required to familiarize themselves with the requirements of the VISA and Mastercard payment organizations, as well as Autopay standards. These requirements are intended to reduce the risk of chargebacks and ensure compliance with industry regulations. Detailed information is available in the article: Requirements for Automatic Payments.

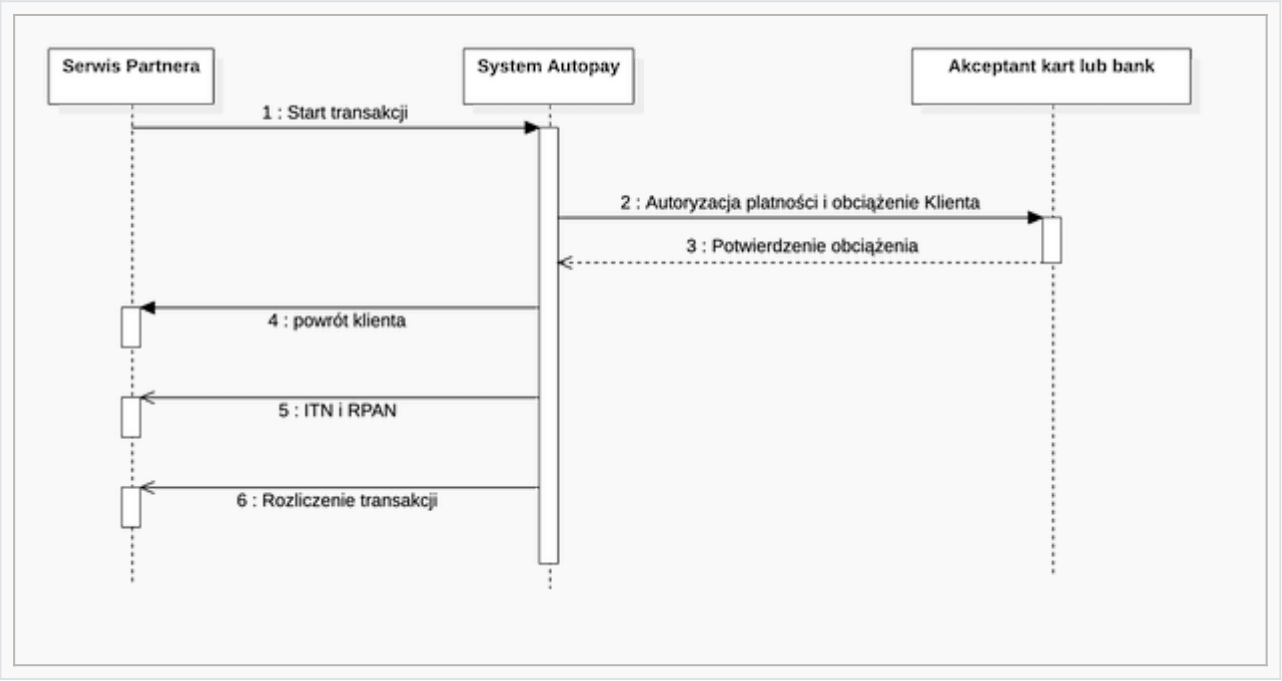
All transactions within the lifecycle of an automated payment

(activation and debit) are carried out within dedicated
Payment Channels (BLIK - **GatewayID=522**, Payment cards – **GatewayID = 1503**) i
gatewayType="Płatność automatyczna". In the case of integrating
BLIK automatic payments, it is possible to specify (in the data provided before
integration) the lifespan of activated automatic payments
(unlimited by default) or to specify it in the initialisation transaction
(parameter **RecurringValidityTime**).

Activation of automatic payment

Activation of the automatic payment consists of an authorisation activation transaction, ITN communication and RPAN. Upon receipt of the RPAN, the Partner is ready to perform recurring debits (or one click).

It is a case of activation of the automatic payment service during
payment for a service/good (thus **RecurringAction=INIT_WITH_PAYMENT** and settlement of the transaction to the Partner).



Process for activating the automatic payment service

The ITN message sent after an automatic payment is similar to those received after one-time payments (it is only extended by the **RecurringData** node, and - for card payments - **CardData**). The other two elements of the service activation process are transaction start and RPAN.

Automated transaction start message

The service activation process is initiated from the Partner Site by starting the transaction with the parameter **RecurringAction** allows you to control the behaviour of the System:

a) value **INIT_WITH_PAYMENT** - corresponds to the activation of the automatic payment service

when paying for a service/goods (card or account is debited with the amount due, and funds from the payment are transferred to the Partner); on the list of available payment channels, the System presents only automatic payments (unless a payment channel has been selected payment channel in the Service),

b) value **INIT_WITH_REFUND** - corresponds to the activation of the automatic payment service outside the payment process for the service/goods (the card or account are debited with the amount of PLN 1, followed by automatic refund of funds to the Customer's account); on the list of available payment channels available, the System presents only automatic payments (unless a payment channel has been payment channel has been selected on the Site),

c) no parameter (or empty) - unless a payment channel is selected on the Site, the System will display all payment channels available for the Site (including automatic ones) and leave it to the Customer to decide: one-time payment or initiation of automatic payment. If the Customer chooses automatic payment, the transaction will be billed to the Partner in the standard way (and the **RecurringAction=INIT_WITH_PAYMENT** parameter will return in RPAN).

> **NOTE:** It is not permitted to initiate activation transactions with the selected automatic payment channel but without the selected RecurringAction.

In some cases (if it follows from the response of the method *legalData*) it is also required to specify the parameters *RecurringAcceptanceState* (with the value **ACCEPTED**, which means that the customer has read and accepted the terms and conditions of the automatic payment on the Partner Service) and *RecurringAcceptanceID*.

Activation of the automatic payment card service is carried out on formats provided by AP. The customer is required to enter card details: first name, surname, card number, expiry date and CVV code. In the case of automatic payment from a bank account (BLIK), authorisation is carried out without entering card data: e.g. with BLIK code (transported in the start-up parameters in AuthorisationCode), or via BLIK OneClick Alias (transported in the start-up parameters in BlikUIDKey/BlikUIDLabel).

Once the transaction has been authorised, the AP System transmits to the Partner Service a transaction status change (ITN) message and a message about activation of the automatic payment service (RPAN). The RPAN message is dedicated to automatic payment activation events and contains its identifier (ClientHash), which the Partner will use during subsequent debits and deactivation of the service. The RPAN also contains information about an action in the automatic payment process (RecurringAction, described above).

Notification of the launch of an automatic payment (RPAN)

Upon receipt of a positive payment status for activation of automatic payment, a dedicated message is sent to the Service. This notification consists of sending by the AP System an XML document containing data about the activated automatic payment. The document is sent via HTTPS protocol (port 443 by default), using the POST method, as an HTTP parameter named recurring. This parameter is stored using the Base64 transport encryption mechanism.

Document format (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
        <recurringActivation>
                <serviceID>ServiceID</serviceID>
                <transaction>
                        <orderID>OrderID</orderID>
                        <remoteID>RemoteID</remoteID>
                        <amount>999999.99</amount>
                        <currency>PLN</currency>
                        <gatewayID>GatewayID</gatewayID>
                        <paymentDate>YYYYMMDDhhmmss</paymentDate>
                        <paymentStatus>PaymentStatus</paymentStatus>
<paymentStatusDetails>PaymentStatusDetails</paymentStatusDetails>
                        <startAmount>999998.99</startAmount>
                        <invoiceNumber>InvoiceNumber</invoiceNumber>
                        <customerNumber>CustomerNumber</customerNumber>
                        <customerEmail>CustomerEmail</customerEmail>
                        <customerPhone>CustomerPhone</customerPhone>
                </transaction>
                <recurringData>
                        <recurringAction>RecurringAction</recurringAction>
                        <clientHash>ClientHash</clientHash>
                        <expirationDate>YYYYMMDDhhmmss</expirationDate>
                </recurringData>
                <cardData>
                        <index>Index</index>
                        <validityYear>ValidityYear</validityYear>
                        <validityMonth>ValidityMonth</validityMonth>
                        <issuer>Issuer</issuer>
                        <bin>BIN</bin>
                        <mask>Mask</mask>
                </cardData>
                <hash>Hash</hash>
        </recurringActivation>
```

Element values: **orderID, serviceID, amount** pertaining to each activated automatic payment are identical to the values of the corresponding fields provided by the Service at the initiation of the respective initialization payment.

Description of the parameters returned for triggering the automatic payment

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1.	serviceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System.
2.	transaction -> orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.

Hash order	name	required	type	description
3.	transaction -> remoteID	YES	string{1,20}	An alphanumeric transaction identifier assigned by the Online Payment System.
5.	transaction -> amount	YES	amount	Transaction amount. A dot - '.' - is used as decimal separator. Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot. **NOTE:** The permissible value of a single Transaction in the Production System is respectively: - for PBL - min. 0.01 PLN, max. 100000.00 PLN (or up to the amount set by the Bank issuing the payment instrument) - for Payment Cards – min. 0.10 PLN, max. 100000.00 PLN (or up to the amount set by the Bank issuing the payment instrument) - for Fast Transfers - min. 0.01 PLN, max. 100000.00 PLN (or up to the Bank's individual limit for a single transaction for an intra-bank transfer) - for BLIK - min. 0.01 PLN, max. 75000.00 PLN (or up to the Bank's individual limit for a single transaction for an intra-bank transfer) - dla OTP – min. 100.00 PLN, max. 2000.00 PLN - for Alior Installments – min. 50.00 PLN, max. 7750.00 PLN
6.	transaction -> currency	YES	string{1,3}	Transaction currency.
7.	transaction -> gatewayID	YES	string{1,5}	Identifier of the Payment Channel through which the customer settled the payment.
8.	transaction -> paymentDate	YES	string{14}	The time when the transaction was authorised, transmitted in the format YYYYMMDDhhmmss. (CET time).

Hash order	name	required	type	description
9.	transaction -> paymentStatus	YES	enum	Transaction authorisation status. Takes values (status transitions identical to the corresponding field in ITN): **PENDING** – transaction initiated **SUCCESS** – correct authorisation of transactions, the Service receives the funds for the transaction **FAILURE** – transaction not completed correctly
10.	transaction -> paymentStatusDetails	YES	enum	Detailed status of the transaction, the value can be ignored by the Service.
11.	transaction -> startAmount	NO	amount	The amount of the transaction stated in the Payment Link (does not include the amount of the commission charged to the Customer, if any.) The sum of the Customer's commission and startAmount is in the amount field, as this is the resulting value of the transaction). A dot - '.' - is used as the decimal separator. Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot.
12.	transaction -> invoiceNumber	NO	string{1,100}	The number of the financial document in the service.
13.	transaction -> customerNumber	NO	string{1,35}	Customer number in the service.
14.	transaction -> customerEmail	NO	string{1,60}	Customer email address.
15.	transaction -> customerPhone	NO	string{9-15}	User telephone number.
16.	recurringData -> recurringAction	NO	string{1,100}	Action in the automatic payment process.
17.	recurringData -> clientHash	YES	string{1,64}	Automatic payment identifier.
18.	recurringData -> expirationDate	NO	string{14}	Expiry time of the automatic payment, transmitted in the format YYYYMMDDhhmmss. (CET time)
19.	cardData -> index	NO	string{1, 64}	Index of the payment card used in the automatic payment (if a card is used).

Hash order	name	required	type	description
20.	cardData -> validityYear	NO	string{4}	Card validity in YYYY format (if a card was used).
21.	cardData -> validityMonth	NO	string{2}	Card validity in mm format (if card used).
22.	cardData -> issuer	NO	string{64}	Card issuer, possible values: - VISA - MASTERCARD - MAESTRO - AMERICAN EXPRESS (currently not supported) - DISCOVER (currently not supported) - DINERS (currently not supported) - UNCATEGORIZED (unrecognized issuer)
23.	cardData -> bin	NO	string{6}	First 6 digits of the card number.
24.	cardData -> mask	NO	string{4}	The last 4 digits of the card number.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

TIP: Element **hash** (message) is used to authenticate the document. For a description of how the hash is calculated, see section Security of transactions.

Response to notification

In response to the notification, an HTTP status of 200 (OK) is expected and a text in XML format (not Base64 encoded), returned by the Partner Service in the same HTTP session, containing an acknowledgement of receipt of the message.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <confirmationList>
            <serviceID>ServiceID</serviceID>
            <recurringConfirmations>
                    <recurringConfirmed>
                            <clientHash>ClientHash</clientHash>
                            <confirmation>Confirmation</confirmation>
                    </recurringConfirmed>
            </recurringConfirmations>
            <hash>Hash</hash>
```

```
        </confirmationList>
```

Confirmation element of the automatic payment

The **confirmation** element is used to convey the status of verification of the authenticity of the transaction by the Partner Service. The value of the element is determined by checking the correctness of the value of the **serviceID** parameter, comparing the values of the **orderID** and **amount** fields in the notification message and in the message initiating the transaction, and verifying the consistency of the calculated hash from the message parameters with the value passed in the message hash field.

Two values are provided for the element **confirmation**:

a) **CONFIRMED** - the parameter values in both messages and the hash parameter match - the transaction is genuine;

b) **NOTCONFIRMED** - the values in the two messages are different or a hash mismatch - transaction not authentic;

> **TIP:** The **hash** element (in the message response) is used to authenticate the response and is calculated from the values of the response parameters. For a description of how the hash is calculated, see the section Security of transactions.

If there is no correct response to the sent notifications, the online payment system will make another attempt to transmit it after a specified time has elapsed. The Partner's service should perform its own business logic (e.g. launching an automatic payment service, mailing etc.) only after the first message of a given **ClientHash**.

ITN/ISTN/IPN/RPAN/RPDN message retry scheme

Below is a diagram describing the scheduled renewal of messages (we reserve however, the possibility of renewing any of them at any time).

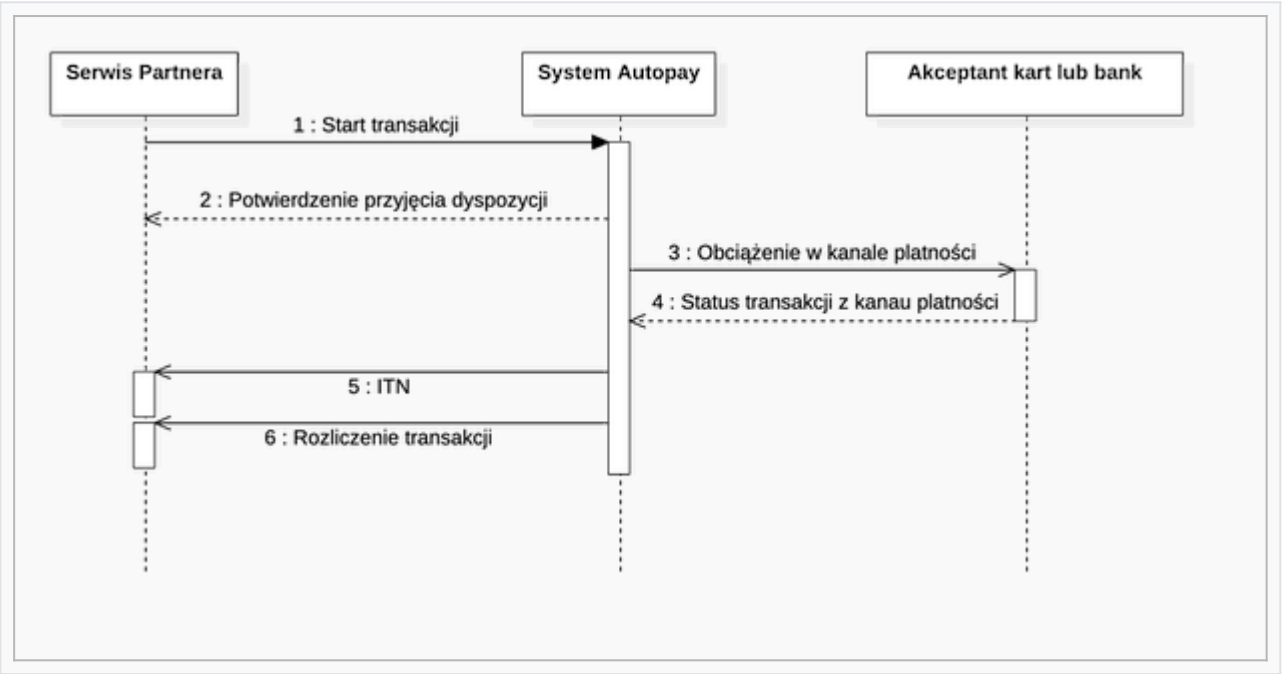
Retry number	Interval until next renewal
1-12	3 min
13-156	10 min
157-204	1 hour
205-209	1 day

> **NOTE:** Continuous repetition of an identical message by the System indicates a missing or incorrect response to it from the Service, and requires the Partner to urgently diagnose the cause.

Charge for automatic payment

Correct receipt of the service identifier (**ClientHash**), makes the Partner ready to automatically charge the Customer for goods/services purchased from the Service. The process consists of transactions and ITN communication.

Below is the process of automatically charging the customer for the service/goods (thus **RecurringAction=MANUAL/AUTO** and settlement of the transaction to the Partner).



The ITN message sent after an automatic payment is similar to those received after one-off payments. It is only extended by the RecurringData node and (for card payments) CardData.

Automated payment transaction start message

In order to execute an automatic debit, the Partner Service should execute a Pre-transaction with a **ClientHash** parameter, consistent with the previously activated automatic payment service (originating from RPAN), with a **RecurringAcceptanceState** parameter of **NOT_APPLICABLE** and the corresponding value of the **RecurringAction** parameter:

a) **AUTO** - recurring payment (debit without customer involvement),

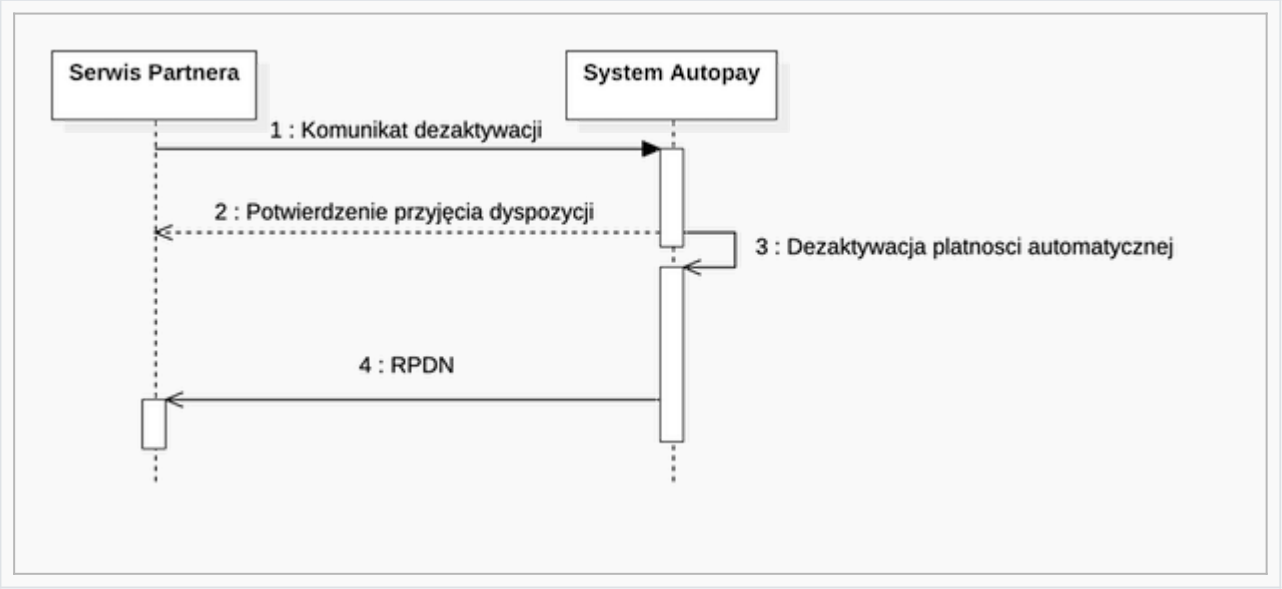
b) **MANUAL** - one-click payment (debit ordered by the customer, called OneClick).

NOTE: The customer's participation in the MANUAL option, is usually limited to calling a message (selecting in the Service the option to pay with the stored card). In the vast majority of cases, an additional authorisation at the bank (in the form of a 3DS or CVC code) is required. Then, instead of a debit (and the status of the order in response to the pre-transaction), the System will return a link to continue - this is the default behaviour of the system on the test environment. In order to test the load scenario without the need for additional authorisation, the need to change the configuration of the System for the duration of the test must be declared.

NOTE: Option not available for BLIK automatic payments (BLIK OneClick).

Deactivation of service

The partner can deactivate the automatic payment service at any time. The process may consist of a message ordering the deactivation and an RPDN message (dedicated to events of cancellation of the automatic payment service).



It may also happen that the cancellation of the service is initiated from the AP side (e.g. at the request of the Customer, bank or card organisation). In such a situation, the System will also provide an RPDN message.

Automatic payment deactivation message

The service can disable the service via a dedicated message. All parameters are transmitted via the POST method (to the address https://{gate_host}/deactivate_recurring). The protocol is case sensitive in both parameter names and values. The values of passed parameters should be encoded in UTF-8.

List of parameters for deactivating automatic payment

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. based on UID), the field value must be unique for the Partner Site.

Hash order	name	required	type	description
3	ClientHash	YES	string{1,64}	Automatic payment identifier.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

NOTE: The Hash (message) element is used to authenticate the document. The value of this element is calculated as the value of a hash function from a string containing the concatenated values of all document fields and the attached shared key.

Response

In response to the notification, an XML-formatted text is returned in the same HTTP session, containing an acknowledgement.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <confirmationList>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <recurringConfirmations>
                    <recurringConfirmed>
                            <clientHash>ClientHash</clientHash>
                            <confirmation>Confirmation</confirmation>
                            <reason>Reason</reason>
                    </recurringConfirmed>
            </recurringConfirmations>
            <hash>Hash</hash>
    </confirmationList>
```

Element confirmation

The **confirmation** element is used to convey the status of verification of the authenticity of the operation by the Service. The value of the element is determined by checking the correctness of the values of the **serviceID** and **clientHash** parameters with those provided in the RPAN message at the start of a given activation payment, as well as verifying the consistency of the calculated hash from the message parameters with the value provided in the Hash field.

Two values are provided for the element **confirmation**:

a) **CONFIRMED** - the parameter values are correct and the Hash parameter are consistent - the operation is genuine;

b) **NOTCONFIRMED** - values in both messages are incorrect or Hash mismatch - operation not authentic;

NOTE: The hash element (in the message response) is used to authenticate the response and is calculated from the value of the response parameters. The value of this element is calculated as the value of a hash function from a string containing the concatenated values of all document fields (without tags) and the attached shared key. For a description of the Security of transactions.

Notification of deactivation of automatic payment (RPDN)

When the automatic payment is deactivated for a given **ClientHash**, a dedicated message in the form of an XML document is sent, is via HTTPS protocol (default port 443), using the POST method, with a parameter named **recurring**. This parameter is stored using the Base64 transport encoding mechanism.

Document format (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <recurringDeactivation>
            <serviceID>ServiceID</serviceID>
            <recurringData>
                    <recurringAction>RecurringAction</recurringAction>
                    <clientHash>ClientHash</clientHash>
                    <deactivationSource>DeactivationSource</deactivationSource>
                    <deactivationDate>DeactivationDate</deactivationDate>
            <recurringData>
            <hash>Hash</hash>
    </recurringDeactivation>
```

The values of the elements: serviceID, clientHash relating to each deactivated cyclic payment, are identical to the values of the corresponding fields, given in the RPAN message at the start of the respective initialisation payment.

Description of the returned parameters

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System.
2	recurringData -> recurringAction	YES	string{1,100}	Action in the automatic payment process (in this case, the value DEACTIVATE).
3	recurringData -> clientHash	YES	string{1,64}	Automatic payment identifier.

Hash order	name	required	type	description
4	recurringData -> deactivationSource	YES	string{1,64}	Reason for deactivation of automatic payment. This description should be treated as informative, the list of its allowed values is constantly growing and the appearance of new values may not entail the non-acceptance of the RPDN message. Below are the current values: - **SERVICE**: requested by Partner - **ACQUIRER**: commissioned by the AP (e.g. upon receipt of information on fraudu) - **BM_PL**ordered by the customer on bills.autopay.eu - **PAYBM**: resulting from card expiry.
5	recurringData -> deactivationDate	YES	string{14}	The time when the automatic payment is switched off, transmitted in the format YYYYMMDDhhmmss. (CET time)
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

NOTE: The hash (message) element is used to authenticate the document. The value of this element is calculated as the value of a hash function from a string containing the concatenated values of all document fields and an attached shared key.

Acknowledgement of receipt of the message

In response to the notification, an HTTP 200 (OK) status is expected and a text in XML format (not Base64 encoded), returned by the Service in the same HTTP session, containing an acknowledgement of receipt of the message.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <confirmationList>
            <serviceID>ServiceID</serviceID>
            <recurringConfirmations>
                    <recurringConfirmed>
                            <clientHash>ClientHash</clientHash>
                            <confirmation>Confirmation</confirmation>
                    </recurringConfirmed>
            </recurringConfirmations>
            <hash>Hash</hash>
    </confirmationList>
```

Element Confirmation

The **confirmation** element is used to convey the status of verification of the authenticity of the operation by the Service. The value of the element is determined by checking the correctness of the values of the **serviceID** and **clientHash** parameters with those provided in the RPAN message at the start of a given initialization payment, and verifying the consistency of the calculated hash from the message parameters with the value provided in the hash field.

Two values are provided for the element **confirmation**:

a) **CONFIRMED** - parameter values are correct and the hash parameter are consistent - authentic operation

b) **NOTCONFIRMED** - values in both messages are invalid or hash mismatch - operation not authentic

> **NOTE:** The hash element (in the message response) is used to authenticate the response and is calculated from the values of the response parameters. For a description of how the hash is calculated, see the section Security of transactions.

In the absence of a correct response to the sent notifications, the System will make further attempts to communicate the latest status of the transaction after the specified time has elapsed. The Partner Service should perform its own business logic (e.g. confirmation email), only after the first message about a given payment status.

> **TIP:** We recommend that you also read the sections *Communication monitoring ITN/ISTN/IPN/RPAN/RPDN* and *Message retry scheme ITN/ISTN/IPN/RPAN/RPDN*.

Payer Identity Verification

Description of Payer Identity Verification

Payer identity verification can be performed by comparing the data provided for verification in the transaction initiation parameters with the data obtained from the payment registered in the System. If, during integration, the option for data verification is agreed upon and the transaction initiation includes the data declared by the client for verification, the System will compare two sets of data and include the result in the ITN message next to the sender's transfer details. For most PBL-type transactions, the first ITN message will not contain the client's address data (name, address). This data is supplemented in the System after scanning the AP account history of the given payway. Once this data is completed, another ITN message is sent to the Partner, now including this information.

> **TIP:** In the case of detecting joint account ownership (in case of the presence of personal data of two individuals), the System places the data of the first person (in order of appearance) in the `fName` and `lName` fields. The full, unprocessed sender data, including all detected first and last names, is available in the `senderData` field.

Fields in the transaction start message related to the service (described in the section Starting a Transaction with Additional Parameters):

- **VerificationFName**
- **VerificationLName**
- **VerificationStreet**
- **VerificationStreetHouseNo**
- **VerificationStreetStaircaseNo**
- **VerificationStreetPremiseNo**
- **VerificationPostalCode**
- **VerificationCity**
- **VerificationNRB**

Fields in the ITN message related to the service (described in the section Additional Fields in the ITN/IPN Message of an Incoming Transaction):

- Node **customerData**
- verificationStatus
- Node **verificationStatusReasons**

TIP: Detailed instructions on how to interpret payment and verification statuses in this process are provided in the section Additional Fields in the ITN/IPN message of an incoming transaction.

Starting a transaction with additional parameters

Transaction starts can be carried out with the additional parameters outlined in the following sections.

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	type	description
8	Language	string{1,2}	Selection of the language in which content will be presented in the Sytem. Acceptable values are: PL, EN, DE, CS, ES, FR, IT. The use of values other than PL should be confirmed during integration and should depend on the Customer's actual choice of language on the Website.
9	CustomerNRB	string{26}	Customer account number, a parameter intended only for Partner Services generating dedicated account numbers for the order or the Customer (see Settlement model for transactions after each payment). Only digits allowed. If, during integration, the use of accounts outside Poland was established, then the field transfers IBAN and the expected field data range changes to: alphanumeric Latin characters (min. 15, max. 32 characters).

Hash order	name	type	description
10	SwiftCode	string{8,11}	The swift code corresponding to the account number given. Only digits allowed. Parameter to be provided if the use of non-Polish accounts was established during integration.
11	ForeignTransferMode	string{4,5}	The system by which the foreign settlement transfer is to be made: SEPA (Single Euro Payments Area) - possible transfers in Euro currency within the European Union Member States, as well as other countries within the Old Continent, e.g. Iceland, Liechtenstein, Norway, Switzerland, Monaco or Andorra, SWIFT - foreign transfers not feasible with SEPA (e.g. different currency than Euro), involves higher transfer costs than with SEPA. Acceptable values: SEPA and SWIFT. Parameter to be provided if the use of accounts outside Poland has been established during integration.
12	TaxCountry	string{1,64}	Country of residence of the payer.
13	CustomerIP	string{1,15}	User's IP address, a parameter intended only for Partner Services running the System in the background (see Pre-transaction and Ordering transfer data in a Fast Transfer transaction).
14	Title	string{1,95}	Title of the transfer clearing the transaction, a parameter intended only for Partner Services cleared by transfer after each payment (see Transaction settlement model after each payment). In some cases, independent of AP, the title of the settlement transfer may be modified independently by the Bank from which the settlement took place. Acceptable alphanumeric Latin characters and characters in the range: `ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!()\"`, where the "/" sign will be replaced by a "-" for outgoing transactions.
15	ReceiverName	string{1,35}	Name of the recipient of the transfer clearing the transaction, parameter intended only for Partner Services cleared by transfer after each deposit (see Transaction settlement model after each payment). Acceptable alphanumeric Latin characters and characters in the range: `ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!()=\[\]{};:?`
16	Products	string{1,10000}	Information about the products included in the transaction, transmitted as Base64 transport protocol encoded XML. The description of the structure in part Product basket.
17	CustomerPhone	string{9-15}	User telephone number. _ Only digits allowed._

Hash order	name	type	description
18	CustomerPesel	string{11}	User's PESEL number. Only digits allowed.
20	CustomerNumber	string{1,35}	Customer number in the Service.
21	InvoiceNumber	string{1,100}	The number of the financial document in the Service.
22	CompanyName	string{1,150}	Company name for automatic verification of the contributor, e.g. Cool Company.
23	Nip	string{1,10}	The VAT identification number of the verified company, e.g. 5851351185. Only digits allowed.
24	Regon	string{9,14}	The REGON identification number of the verified company, e.g. 191781561. Only digits allowed.
25	VerificationFName	string{1,32}	The first name provided in the Service for automatic verification of the contributor, e.g. Jan. Only letters of the Polish alphabet are acceptable.
26	VerificationLName	string{1,64}	Name given in the Service for automatic verification of the contributor, e.g. Smith. Only letters of the Polish alphabet are acceptable.
27	VerificationStreet	string{1,64}	Street provided on the Service for automatic verification, e.g. Long. Only letters of the Polish alphabet and numbers are permitted.
28	VerificationStreetHouseNo	string{1,64}	The house number provided on the Service for automatic verification of the contributor. Only letters of the Polish alphabet and numbers are permitted.
29	VerificationStreetStaircaseNo	string{1,64}	The staircase number provided on the Service for automatic verification of the contributor. Only letters of the Polish alphabet and numbers are permitted.
30	VerificationStreetPremiseNo	string{1,64}	Flat number provided on the Service for automatic verification of the contributor. Only letters of the Polish alphabet and numbers are permitted.
31	VerificationPostalCode	string{1,64}	Postal code provided in the Service for automatic verification of the contributor, format XX-XXX, e.g. 80-180. Only numbers and the "-" sign are permitted.
32	VerificationCity	string{1,64}	City specified in the Service for automatic verification of the contributor, e.g. Warsaw. Only letters of the Polish alphabet and numbers are permitted.
33	VerificationNRB	string{1,26}	The bank account number provided on the Service for automatic verification of the contributor, e.g. 88154010982001554242710005. Only digits allowed.

Hash order	name	type	description
35	RecurringAcceptanceState	string{1,100}	Automatic payment terms and conditions acceptance information indicating whether the Customer has accepted the automatic payment terms and conditions or whether acceptance must be enforced on the part of the System. Field required for automatic payments in the WhiteLabel model of the payment service provided by AP to the Customer (the Customer pays a commission). The availability of the terms and conditions can be checked by calling the legalData method. Allowed values: **NOT_APPLICABLE** - acceptance of terms and conditions not required (single payment or debit action, i.e. recurringAction with AUTO or MANUAL value) **ACCEPTED** - declaration of acceptance of the terms and conditions in the contractor's service (to be provided with the **RecurringAcceptanceID**) **PROMPT** - on the card form the card enrolment consent is proposed, its ticking initiates the automatic payment **FORCE** - consent to save the card is required on the card form, otherwise payment is not possible. **NOTE:** The availability of the **ACCEPTED/PROMPT/FORCE** option depends on business arrangements (in particular, determination of the place of display of consent/automatic payment service rules).
36	RecurringAction	string{1,100}	Required field for automatic payments, specifying the possible actions on an automatic payment. Allowed values: INIT_WITH_PAYMENT - activation of automatic payment with payment for goods/services INIT_WITH_REFUND - activation of automatic payment followed by return of payment AUTO - cyclical payment (debit without customer involvement) MANUAL - one-click payment (debit initiated by the customer) **NOTE:** Option not available for BLIK automatic payments (BLIK OneClick). DEACTIVATE - deactivate automatic payment
37	ClientHash	string{1,64}	Automatic payment identifier. This parameter allows a payment instrument (e.g. Card, BLIK) to be assigned to a Customer in an anonymised manner. Based on it, the Partner can trigger subsequent debits in the automatic payment model.
38	OperatorName	string{1,35}	The name of the operator of the telephone number given. Allowed values: Plus, Play, Orange, T-Mobile.
39	ICCID	string{12,19}	SIM card number of the phone number specified. Allowed values (only digits allowed): For Plus: 12 or 13 digits For Play, Orange, T-Mobile: 19 digits

Hash order	name	type	description
40	AuthorizationCode	string{6}	A payment authorisation code entered on the Service/System side (currently supported in BLIK). Its use means that there is no need to redirect the Customer to the Payment Channel page. It should therefore only be entered via Pre-transaction. Format dependent on the Payment Channel. For BLIK carried out in the background (BLIK 0, possibly BLIK OneClick): 6 digits.
41	ScreenType	string{4,6}	Type of view of the payment authorisation form. Acceptable values: IFRAME - not supported FULL.
42	BlikUIDKey	string{1,64}	Alias UID key (used in BLIK). This is the unique identifier of the user on the Service. _Permissible alphanumeric Latin characters and characters: .
43	BlikUIDLabel	string{1,20}	Alias UID label (used in BLIK), which will be presented to the Customer in the banking application to distinguish accounts with the Partner. It is recommended to use the login, nickname or email address assigned to the Customer's authorised account. If there is a possibility of personal data (e.g. e-mail address jan.kowalski@poczta.pl) make the data confidential (by replacing 3 dots with some characters, e.g. ja...ki@po...pl). Acceptable alphanumeric Latin characters and characters in the range: . : @ - , space.
44	BlikAMKey	string{1,64}	Alias key of the bank's mobile application (used in BLIK). This is the unique identifier of your BLIK account. Acceptable digits.
45	ReturnURL	string{1,1000}	Dynamic payment return address starting with http/https. Acceptable valid URLs. May contain IP, port, subdomain, Polish characters, and (after domain) parameters and special characters: `,'+&;%$#_!=.`
46	TransactionSettlementMode	string{2,10}	Possibility to change the settlement of transactions. Lack of parameter (backward compatibility) treated as sending COMMON values. The NONE parameter results in the transaction being treated as a prepaid balance top-up and no settlement. Acceptable values: COMMON NONE

Hash order	name	type	description
47	PaymentToken	string{1,100000}	Token used in Visa and Google Pay wallets placed directly on the Partner's website (authorisation without redirection to the System). In this case, the Website integrates directly with the Visa and/or Google API to retrieve the card handle. The token obtained is transmitted to the Online Payment System in a form encoded with the Base64 transport protocol. **NOTE:** The parameter is unnecessary if the selection of Payment Channels (and logging into the wallet) is done directly on the Online Payment System page..
48	DocNumber	string{1,150}	Financial document number.
49	RecurringAcceptanceID	string{1,10}	The identifier of the automatic payment service terms and conditions displayed on the Service and accepted by the Customer. Required field for automatic payments in the **WhiteLabel** model of the payment service provided by AP to the Customer (the Customer pays a commission). The ID of the terms and conditions appropriate for the selected language (and payment channel) should be retrieved using the **legalData** method.
50	RecurringAcceptanceTime	string{1,19}	Optional field. The moment of acceptance of the Terms and Conditions by the Customer, this value will be verified by the System with the time of the Terms and Conditions with the given **RecurringAcceptanceID**. Example value: 2014-10-30 07:54:50 (Time in CET).
51	DefaultRegulationAcceptanceState	string{1,100}	Information about acceptance of the payment service terms and conditions. Field required in the **WhiteLabel** model of the payment service provided by AP to the Customer (Customer pays the commission). Its absence may be associated with an error or display of the System's transition page with the requirement to accept the terms and conditions. The availability of the regulations can be checked by calling the **legalData** method. Allowed values: **ACCEPTED** - acceptance of rules and regulations made in the counterparty service (to be specified together with **DefaultRegulationAcceptanceID**).
52	DefaultRegulationAcceptanceID	string{1,10}	Identifier of the terms and conditions of the payment service provided by AP to the Customer displayed on the Website and accepted by the Customer. Field required in the WhiteLabel model of the payment service provided by AP to the Customer (Customer pays the commission). The ID of the rules and regulations relevant for the selected language (and payment channel) must be retrieved using the legalData method.
53	DefaultRegulationAcceptanceTime	string{1,19}	Optional field. The time when the rules and regulations were accepted by the customer, this value will be verified by the System with the time of the rules and regulations with the specified DefaultRegulationAcceptanceID; example value: 2014-10-30 07:54:50. (Time in CET)

Hash order	name	type	description
54	WalletType	string{1,32}	The payment wallet type, determines the source of the PaymentToken parameter (if it has been sent). _Available values are: SDK_NATIVE - native card format (mobile SDK) WIDGET - card widget (card format before the start of the transaction) _
55	RecurringValidityTime	string{10}	Expiry date of activated BLIK automatic payment, format YYYY-MM-DD (example value: 2024-01-30). If the parameter is missing, a default configuration date set during integration (customarily indefinite) will be proposed.
56	ServiceURL	string{1,1000}	Parameter specifying the web address of the shop from which the payment was launched, starting with http/https. Acceptable valid URL. May contain IP, port, subdomain, Polish characters, as well as (after the domain) parameters and special characters: `,'+&;%$#_!=`
57	BlikPPLabel	string{1,35}	Label of the BLIK recurring payment, displayed in the mobile application when it is accepted. If the parameter is missing, its default value will be used (set during integration).
58	ReceiverNameForFront	string{1,35}	Name of the BLIK payment recipient, displayed in the mobile application when accepting a payment. If the parameter is missing, its default value will be used (customarily the Service URL). **NOTE:** The service must be agreed with the business mentor. Acceptable alphanumeric Latin characters, characters in the range: `ĘęÓóĄąŚśŁłŻżŹźĆćŃń-/,!()=[]{};:.?` and space.
59	AccountHolderName	string{1,100}	Name of the owner of the means of payment.

Product basket

Description of the product basket

The product basket is sent as a parameter (POST method) named Products. Its value is encoded with the Base64 transport protocol.

Format before encoding (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <productList>
            <product>
                    <subAmount>SubAmount1</subAmount>
                    <params>Params1</params>
            </product>
            <product>
                    <subAmount>SubAmount2</subAmount>
                    <params>Params2</params>
            </product>
            …
            <product>
                    <subAmount>SubAmountN</subAmount>
```

```
                    <params>ParamsN</params>
                </product>
            </productList>
```

The **productList** node must contain at least 1 **product** element, each **product** node must contain one **subAmount** and **params** element each.

The **subAmount** element must contain a positive product amount (the decimal separator is a dot followed by two penny digits). The sum of the amounts of the subsequent products must be equal to the amount specified in the **Amount** parameter (transaction amount).

If the above conditions are not met, the System will return an error.

Element Params

The **params** element can be used to convey product-specific information. The names of the parameters and their meaning are subject to agreement in working form each time during integration.

Examples of product parameters and their meaning below.

- In this case, the node contains the product name:

```
            <params>
                        <param name="productName" value="Product name 1" />
                </params>
```

- In this case, the node contains two values assigned to the product, which can denote, for example, the type of product and its name:

```
            <params>
                        <param name="productType" value="ABCD" />
                        <param name="productName" value="Product name 1" />
                </params>
```

- In the event that the Service has a balance in the System, and plans to make refunds to the Customer of all or part of the amount paid for the designated product, it is obliged to transfer in the product its **unique** identifier (parameter named productID with type string{1,36} (Latin alphanumeric characters and characters allowed: _ and -)):

```
            <params>
                        <param name="productID" value="12456" />
                </params>
```

- If the Partner uses an extended structure (multiple settlement points), it is obliged to transmit in each product the identifier of the settlement point (parameter named idBalancePoint with type integer{1,10}):

```
            <params>
                        <param name="idBalancePoint" value="12456" />
```

```
            </params>
```

- In the event that the Partner uses billing **MASS_TRANSFER**, i.e:
    - each transaction is settled immediately upon deposit and
    - settlements are made at the product/billing point level (i.e. as many settlement transfers are made after the deposit as products in the basket), and
    - no fixed billing data has been set (or not all billing data is set rigidly in the billing configuration),

The partner must provide in each product the missing data for the billing of that product. The available parameters that constitute the data are:

- **customerNRB** - target account number for billing.
  NRB format (26 digits with checksum). If, during integration, the use of non-Polish accounts is established, then the field transfers IBAN and the expected data range of the field changes to: alphanumeric Latin characters (min. 15, max. 32 characters). Specifying values in the IBAN format will result in the need to specify in the product also the parameters swiftCode, foreignTransferMode.

- **title** - - The title of the product settlement transfer. In certain cases, independent of AP, the title of the settlement transfer may be independently modified by the Bank from which the settlement occurred.
  Acceptable alphanumeric Latin characters and characters in the range: `ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-,!()\"`

- **receiverName** - the name of the recipient of the transfer clearing the product.
  Acceptable alphanumeric Latin characters and characters in the range: `ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!()=\[\]{};:?`

Example of application of parameters to a product:

```
<params>
            <param name="customerNRB" value="831090109800000000107285707" />
            <param name="title" value="Settlement of product X" />
            <param name="receiverName" value="John Smith" />
    </params>
```

- In the Marketplace model, the Partner is obliged to provide a billing point identifier (parameter named **idBalancePoint** of type integer{1,10}). This also applies to the balance crediting of the settlement point.

Example of basket parameters:

```
<params>
            <param name="idBalancePoint" value="12456" />
            <param name="productName" value="Balance transfer for Autopay" />
    </params>
```

Product basket display on the Payment Channel selection screen

If it has been established during the discussions concerning the product basket that its summary is to be displayed on the System page (Payment Channel selection screen), the labels of each parameter used in the basket can be specified. The System can use the default label of the parameter or can adopt it at the start of the transaction.

The value of the **title** attribute will be displayed before the value of the product parameter.

Example of the title attribute

```
<params>
            <param name="productName" value="Product name 1" title="Name"/>
            <param name="productType" value="ABCD" title="Type"/>
    </params>
```

Additional online communication options to the partner

Additional fields in the ITN/IPN message of the incoming transaction

Description

Instant notifications of a change in the status of a transaction may include additional fields (see Schemes for Preauthorisation). Their occurrence is a configuration issue, determined during integration (by default only the node is sent **customerData**).

Whether it is an ITN or IPN message is determined solely by the presence of a node **product**.

Full list of additional fields in the ITN/IPN message

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	type	description
11	addressIP	string{1,15}	The Customer's IP address as registered by the System front-end, or the address transmitted to the System in the CustomerIP parameter, or the IP from which the transaction was started in the System.
13	customerNumber	string{1,35}	Customer number in the service.

Hash order	name	type	description
21	title	string{1,140}	Title of the payment. In certain cases, beyond AP's control, the title of the transfer may be modified independently by the Bank where the customer's payment took place.
22	customerData-> fName	string{1,128}	Name of payer.
23	customerData-> lName	string{1,128}	Last name of payer.
24	customerData-> streetName	string{1,128}	Payer street name.
25	customerData-> streetHouseNo	string{1,10}	Payer's house number.
26	customerData-> streetStaircaseNo	string{1,10}	Payer staircase number.
27	customerData-> streetPremiseNo	string{1,10}	Payer premises number.
28	customerData-> postalCode	string{1,6}	Postcode of the payer's address.
29	customerData-> city	string{1,128}	Payer city.
30	customerData-> nrb	string{1,26}	Payer's bank account.
31	customerData-> senderData	string{1,600}	Payer data in undivided form.
32	verificationStatus	enum	Element containing payer verification status. It is an enum allowing values: PENDING, POSITIVE and NEGATIVE.
nd.	verificationStatusReasons	list	A list of reasons for negative or pending verification. There can be many reasons.

Hash order	name	type	description
33	verificationStatus	enum	Detailed reason in case of negative or pending verification. Allowed values for negative verification: - NAME - name or surname does not match - NRB - account number does not match - TITLE - the title does not match - STREET - the street name does not match - HOUSE_NUMBER - house number not correct - STAIRCASE - staircase number not correct - PREMISE_NUMBER - the premises number is not correct - POSTAL_CODE - postal code does not match - CITY - city disagrees - BLACKLISTED - the account from which the payment was made is blacklisted - SHOP_FORMAL_REQUIREMENTS - service verified did not meet formal conditions - JOINT_OWNERSHIP - Co-ownership has been detected on the bank account, which – according to the service configuration – is not permitted Allowed values for pending verification: - NEED_FEEDBACK - waiting for the service to meet the formal conditions is in progress. **NOTE:** To count the Hash values, the values of the following nodes are taken: **verificationStatusReasons**, **verificationStatusReason**.
60	startAmount	amount	The amount of the transaction stated in the Payment Link (does not include the amount of the commission charged to the Customer, if any). The sum of the customer's commission and **startAmount** is in the field **amount**as this is the resulting transaction value). A full stop is used as decimal separator - '.' Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot.
70	recurringData-> recurringAction	string{1,100}	Action in the automatic payment process (meaning and allowed values are described in section Definitions).
71	recurringData-> clientHash	string{1,64}	Automatic payment identifier generated by the AP and transmitted to the Partner upon successful activation of the automatic payment.

Hash order	name	type	description
72	recurringData-> expirationDate	string{14}	Expiry time of the automatic payment, transmitted in the format YYYYMMDDhhmmss. (CET time)
73	cardData-> index	string{1,64}	Card index (if a card is used). Index identifies a card with a given expiry date (changing the date or card number changes the value of this parameter).
74	cardData-> validityYear	string{4}	Card validity in YYYY format (if a card was used).
75	cardData-> validityMonth	string{4}	Card validity in mm format (if card used).
76	cardData-> issuer	string{1,64}	Card type (if a card is used). Possible values: - VISA - MASTERCARD - MAESTRO - AMERICAN EXPRESS (currently not supported) - DISCOVER (currently not supported) - DINERS (currently not supported) - UNCATEGORIZED (unrecognized issuer)
77	cardData-> bin	string{6}	First 6 digits of the card number (if a card is used). Passed if the cardData-> mask parameter is not passe
78	cardData-> mask	string{4}	The last 4 digits of the card number (if a card is used). Passed if the cardData->bin parameter is not passed.
90	product-> subAmount	amount	The product amount uses a full stop as decimal separator - '.' Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot. Node only available in IPN communications.
91	product-> params	list	Subsequent product parameters according to the basket format in the transaction start. Node only available in IPN messages.

NOTE: For counting the Hash values, the **value** attributes of the subsequent **product.params** nodes are taken.

Example of ITN/IPN message with additional parameters (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <transactionList>
      <serviceID>ServiceID</serviceID>
      <transactions>
      <transaction>
```

```
                <orderID>OrderID</orderID>
                <remoteID>RemoteID</remoteID>
                <amount>999999.99</amount>
                <currency>PLN</currency>
                <gatewayID>GatewayID</gatewayID>
                <paymentDate>YYYYMMDDhhmmss</paymentDate>
                <paymentStatus>PaymentStatus</paymentStatus>
                <paymentStatusDetails>PaymentStatusDetails</paymentStatusDetails>
                <addressIP>127.0.0.1</addressIP>
                <customerNumber>1111111</customerNumber>
                <title>title</title>
                <customerData>
                  <fName>fName</fName>
                  <lName>lName</lName>
                  <streetName>streetName</streetName>
                  <streetHouseNo>streetHouseNo</streetHouseNo>
                  <streetStaircaseNo>streetStaircaseNo</streetStaircaseNo>
                  <streetPremiseNo>streetPremiseNo</streetPremiseNo>
                  <postalCode>postalCode</postalCode>
                  <city>city</city>
                  <nrb>nrb</nrb>
                  <senderData>senderData</senderData>
                </customerData>
                <verificationStatus>verificationStatus</verificationStatus>
                <verificationStatusReasons>
                   <verificationStatusReason>reason1</verificationStatusReason>
                   <verificationStatusReason>reason2</verificationStatusReason>
                   <verificationStatusReason>reason3</verificationStatusReason>
                </verificationStatusReasons>
                <startAmount>999998.99</startAmount>
                <recurringData>
                        <recurringAction>RecurringAction</recurringAction>
                        <clientHash>ClientHash</clientHash>
                        <expirationDate>YYYYMMDDhhmmss</expirationDate>
                </recurringData>
                <cardData>
                        <index>Index</index>
                        <validityYear>ValidityYear</validityYear>
                        <validityMonth>ValidityMonth</validityMonth>
                        <issuer>Issuer</issuer>
                        <bin>BIN</bin>
                </cardData>
                <product>
                        <subAmount>SubAmount</subAmount>
                        <params>
                                <param name="idBalancePoint" value="idBalancePoint"/>
                                <param name="invoiceNumber" value="invoiceNumber"/>
                                <param name="customerNumber" value="customerNumber"/>
                                <param name="subAmount" value="SubAmount"/>
                        </params>
                </product>
          </transaction>
          </transactions>
          <hash>Hash</hash>
        </transactionList>
```

Detailed description of the change in the verification status - for a transaction successfully completed (positive or negative result)

Payment status (paymentStatus)	Verification status (verificationStatus)	Verification details (verificationStatusReasons)	Details
PENDING	PENDING	Empty	The customer selected the payment method.
SUCCESS	PENDING	Empty	The transaction has been paid, the System is waiting to retrieve the contributor's details from the account.
SUCCESS	PENDING	NEED_FEEDBACK	Autopay is waiting for the Partner to meet the formal conditions.
SUCCESS	POSITIVE	Empty	The verification was successful.
SUCCESS	NEGATIVE	List of reasons for NEGATIVE	Negative verification.

Detailed description of verification status change - for a transaction not completed correctly

Payment status (paymentStatus)	Verification status (verificationStatus)	Verification details (verificationStatusReasons)	Details
PENDING	PENDING	Empty	The customer selected the payment method.
FAILURE	PENDING	Empty	The transaction has not been completed correctly. Verification status will not be provided.

Detailed transaction statuses

The ITN message for an incoming transaction contains, in addition to the payment status (field **paymentStatus**), a detailed description of this status (field **paymentStatusDetails**). This description is to be treated as informative, the list of its allowed values is constantly growing and the appearance of new values may not imply non-acceptance of the ITN message.

Transaction status values - General statuses (independent of the payment channel)

Field value	Meaning of the field
AUTHORIZED	transaction authorised by a Payment Channel
ACCEPTED	transaction approved by the Call Centre (e.g. as a result of a successful complaint)
REJECTED	Transaction interrupted by a Payment Channel (bank/clearing agent)
REJECTED_BY_USER	transaction terminated by Customer
INCORRECT_AMOUNT	an amount different from that indicated at the start of the transaction was paid
EXPIRED	past due transaction
CANCELLED	a transaction cancelled by the Partner Service or the Call Centre (e.g. at the request of the Customer). It is not possible to start a new transaction or to continue a previously started transaction with the same **OrderID**
RECURSION_INACTIVE	cyclical payment activity error
ANOTHER_ERROR	another error occurred while processing the transaction

Transaction status values - Card statuses

Field value	Meaning of the field	Optional card organisation error code
CONNECTION_ERROR	error with connection to payment card issuer's bank	✔
CARD_LIMIT_EXCEEDED	error in payment card limits	✔
SECURITY_ERROR	security error (e.g. incorrect cvv)	✔
DO_NOT_HONOR	Refusal of authorisation at bank; suggested customer contact with card issuer	✔
THREEDS_NEGATIVE	transaction unsuccessful in 3DS	✔
CARD_EXPIRED	card not valid	✔
INCORRECT_CARD_NUMBER	incorrect card number	✔
FRAUD_SUSPECT	suspected fraud (e.g. lost card, etc.)	✔
STOP_RECURRING	recurrence impossible due to cancellation of customer instructions	✔
VOID	transaction abandoned or communication error	✖
UNCLASSIFIED	other errors	✔

Transaction status values - BLIK transaction specific statuses

Field value	Meaning of the field
INSUFFICIENT_FUNDS	Lack of funds. Recommended message to be displayed to the customer stating: *Payment unsuccessful - bank refusal. Check the reason for refusal in the bank application.* *If the reason is that you are over the limit, you can increase it by contacting your bank.*
LIMIT_EXCEEDED	Limits error (e.g. amounts). Recommended message to be displayed to the customer: *Payment unsuccessful - bank refusal. Check the reason for refusal in the bank application.* *If the reason is that you are over the limit, you can increase it by contacting your bank.*
BAD_PIN	an incorrect PIN was entered when confirming the transaction
ISSUER_DECLINED, USER_DECLINED, SEC_DECLINED	transaction terminated by customer
TIMEOUT and **AM_TIMEOUT**	timeout in communication with the bank's mobile application
USER_TIMEOUT	timeout waiting for the customer to confirm the transaction

Immediate notification of a change in product status (IPN)

In the case of a Partner using extended-structure (Product basket with multiple billing points), the System provides independent notification of status changes for each product. Such a service makes sense if individual billing points should receive their own notifications. In this case, the IPN configuration (address for notifications, fields to be included in the message, etc.) is stored precisely at the billing point configuration level. The structure of the IPN is similar to that of the ITN (extended only by a **product** node similar to the one described in section Additional fields in ITN message/IPN input transaction, completed only by the repetition of the **subAmount** in the params). IPN example in subsection Additional fields in the ITN/IPN message of an input transaction).

Immediate notification of a change in the status of a settlement transaction (ISTN)

It is possible to provide messages about all disbursements (settlements, balance disbursements and refunds) made by the System as part of the payment service. As the service is not enabled by default, the need for this service, together with the ISTN mailing address, must be requested by the Partner during the requirements determination.

In the event of successful initiation of ISTN communication, the System immediately transmits notifications of the fact of the ordering of the settlement transaction (withdrawals/refunds, if any) and the change of its status. The confirmations are sent, to the address on the Partner's service server established during the addition configuration of the Partner Service:

```
https://shop_name/receipt_of_settlement_information
```

This notification consists of the System sending an XML document containing the new transaction statuses. The document is sent via the HTTPS protocol (default port 443). The document is sent by the POST method, as an HTTP parameter named transactions. This parameter is stored using the Base64 transport encoding mechanism.

Document format (XML)

```
    <?xml version="1.0" encoding="UTF-8"?>
        <transactionList>
                <serviceID>ServiceID</serviceID>
                <transactions>
                        <transaction>
                                <isRefund>true/false</isRefund>
                                <productID>ProductID</productID>
                                <orderID>OrderID</orderID>
                                <orderOutID>OrderOutID</orderOutID>
                                <remoteID>RemoteID</remoteID>
                                <remoteOutID>RemoteOutID</remoteOutID>
                                <amount>999999.99</amount>
                                <currency>PLN</currency>
                                <transferDate>YYYYMMDDhhmmss</transferDate>
                                <transferStatus>TransferStatus</transferStatus>
<transferStatusDetails>TranasferStatusDetails</transferStatusDetails>
                                <title>Title</title>
                                <receiverBank>ReceiverBank</receiverBank>
                                <receiverNRB>ReceiverNRB</receiverNRB>
                                <receiverName>ReceiverName</receiverName>
                                <receiverAddress>ReceiverAddress</receiverAddress>
                                <senderBank>SenderBank</senderBank>
                                <senderNRB>SenderNRB</senderNRB>
                        </transaction>
                </transactions>
                <hash>Hash</hash>
        </transactionList>
```

Returned parameters

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	NO	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System.
2	isRefund	NO	Boolean	Information on whether the ISTN relates to the return of a transaction (true) or normal settlement (false).
3	productID	NO	string{1,36}	The identifier of the billed product from the product basket of the incoming transaction, the field value must be unique for the Partner Service.
4	orderID	NO	string{1,32}	Input transaction identifier of up to 32 Latin alphanumeric characters, the field value must be unique for the Partner Service.

Hash order	name	required	type	description
5	orderOutID	NO	string{1,32}	Output transaction identifier of up to 32 alphanumeric Latin characters. The field can be assigned by the Service (in the case of a billing order) or by the Online Payment System.
6	remoteID	NO	string{1,20}	Alphanumeric identifier of the incoming transaction assigned by the Online Payment System (given if one payment is linked to the settlement).
7	remoteOutID	NO	string{1,20}	The alphanumeric identifier of the settlement transaction assigned by the Online Payment System.
8	amount	YES	amount	Transaction amount. A dot - '.' - is used as decimal separator. Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot.
9	currency	YES	string{1,3}	Transaction currency.
40	transferDate	NO	string{14}	The time when the transaction was authorised, transmitted in the format YYYYMMDDhhmmss. (CET time). Only occurs for transferStatus=SUCCESS.
41	transferStatus	YES	enum	Authorisation status of the settlement transaction. It adopts the following values: - PENDING - transfer pending - SUCCESS - transfer ordered to bank - FAILURE - transfer cannot be made, e.g. wrong account number
42	transferStatusDetails	NO	enum	Detailed transaction status, value can be ignored by the Partner Service. It accepts the following values (the list can be extended): - AUTHORIZED - transaction submitted for execution at bank - CONFIRMED - transaction confirmed at the bank (money physically sent) - CANCELLED - transaction cancelled by the Partner Service or Call Centre (e.g. at the request of the Service) - ANOTHER_ERROR - another transaction processing error has occurred
43	title	NO	string{1,140}	Title of the settlement transfer. In certain cases, beyond AP's control, the title of the settlement transfer may be modified independently by the Bank from which the settlement took place. Acceptable alphanumeric Latin characters and characters in the range: ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!@#%\^*()_=+\[\]{};:?, where the "/" sign will be replaced by a "-" for outgoing transactions.
44	receiverBank	NO	string{1,64}	The name of the bank to which the System has made the transfer.
45	receiverNRB	NO	string{26}	The bank account number of the recipient of the transfer.
46	receiverName	NO	string{1,140}	Name of the recipient of the transfer. Acceptable alphanumeric Latin characters and characters in the range: ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!@#%\^*()_=+\[\]{};:?
47	receiverAddress	NO	string{1,140}	Transfer recipient address. Acceptable alphanumeric Latin characters and characters in the range: ĘęÓóĄąŚśŁłŻżŹźĆćŃń\\s.-/,!@#%\^*()_=+\[\]{};:?
48	senderBank	NO	string{1,64}	The name of the bank through which the System made the transfer.
49	senderNRB	NO	string{26}	Bank account number of the sender of the transfer.

Hash order	name	required	type	description
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Response to notification

In response to the notification, a text in XML format (not Base64 encoded) is expected, returned by the Partner Service in the same HTTP session, containing an acknowledgement of the receipt of the transaction status.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <confirmationList>
            <serviceID>ServiceID</serviceID>
            <transactionsConfirmations>
                    <transactionConfirmed>
                            <remoteOutID>RemoteOutID</remoteOutID>
                            <confirmation>Confirmation</confirmation>
                    </transactionConfirmed>
            </transactionsConfirmations>
            <hash>Hash</hash>
    </confirmationList>
```

The **confirmation** element is used to convey the status of verification of the authenticity of the transaction by the Partner Service. The value of the element is determined by checking the correctness of the value of the **serviceID** parameter, as well as verifying that the calculated hash matches the value passed in the hash field.

Two values are provided for this element:

a) **CONFIRMED** - the hash parameter is consistent - the transaction authentic;

b) **NOTCONFIRMED** - the hash parameter is inconsistent - transaction not authentic;

If there is no correct response to the notifications sent, the System will make further attempts to communicate a new status after a specified time. The Partner Service should perform its own business logic, only after the first message about a given payment status.

> **TIP:** It is worth taking a look at ITN/ISTN/IPN/RPAN/RPDN message retry scheme.

Detailed description of the behaviour and change of settlement statuses (transferStatus)

In the basic model, the System will only provide a status of **SUCCESS**, however, more accurate notification is possible. The full option should be notified during integration and involves the following pattern of transitions statuses.

ZAn order for a settlement transaction sends a status of **PENDING**. Later, the system will deliver

SUCCESS or **FAILURE**. For a transaction for which **SUCCESS** status has occurred, there should no longer be a change of status to **FAILURE**. There may, however, be a change in detail status (subsequent detail status change messages are informational only and should not entail the re-execution of any business logic).

In special cases (e.g. an error at the bank), a transaction that was originally confirmed may be passed on for re-execution and therefore change its status to **PENDING** and back to **SUCCESS**.

Another special case could be a **FAILURE** status (e.g. after an internal System error), then replaced by a **SUCCESS** status.

Additional services

Querying the list of currently available Payment Channels

Description

In order to build a payment method selection view on the Website, the System allows the current list of payment channels to be queried remotely. For this purpose, call the method **gatewayList** (https://{gate_host}/gatewayList/v3) with the relevant parameters (in JSON format). All parameters are transmitted using REST technology. The protocol is case sensitive in both parameter names and values. The values of the passed parameters should be encoded in UTF-8.

List of available parameters

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	integer	Partner Service ID.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. based on UID). The field value must be unique for the Partner Service.
3	Currencies	YES	string{0,1000}	List of currencies whose list of available channels is to be returned. The list should be a minimum of one element. Acceptable values are: PLN, EUR, GBP, USD.
4	Language	YES	string{2}	Language in which descriptions of payment methods will be returned. Acceptable values: PL,EN,DE,FR,IT,ES,CS,RO,SK,HU,UK,EL,HR,SL,TR,BG.
5	Hash	YES	string{64}	Value of message digest function calculated as described in section Security of transactions

Example message

```
{
```

```
        "ServiceID": 47498,
        "MessageID": "11111111111111111111111111111111",
        "Currencies":"PLN,EUR",
        "Language": "PL",
        "Hash": "306519f632e53a5e662de0125da7ac3f8135c7e4080900f2b145d4b25ff1b55d"
}
```

Response to request

In response to a request, 2 lists of definitions are returned in the same HTTP session:

a) Payment channels (node `gatewayList`)

b) Payment groups (node `gatewayGroups`)

Below is a detailed description of the message returned:

	name	required	type	description
1	result	YES	string{1,5}	Response status. Acceptable values: - OK - ERROR
2	errorStatus	YES	string{1,100}	Error status, to be filled in in the event of an error (otherwise null).
3	description	YES	string{1,500}	Description of the error, to be filled in in the event of an error (otherwise null).
4	gatewayGroups	YES	list	A list containing payment groups.
4.1	type	YES	string{1,20}	Payment group type. Each payment definition is assigned to one of the types.
4.2	title	YES	string{1,50}	Name of payment group.
4.3	shortDescription	NO	string{1,200}	Brief description of the payment group.
4.4	description	NO	string{1,1000}	Detailed description of the payment group.
4.5	order	YES	integer	Recommended order of display of payment groups.
4.6	iconUrl	NO	string{1,100}	Address from which the payment group logo can be downloaded.
5	serviceID	YES	string{1,10}	Partner Service Identifier; derived from the method request.
6	messageID	YES	string{32}	Message identifier derived from the method request.
7	gatewayList	NO	list	List containing further payment channels (empty if no payment channels are configured).

	name	required	type	description
7.1	gatewayID	YES	integer{1,5}	Identifier of the Payment Channel with which the Customer can settle the payment.
7.2	name	YES	string{1,200}	The name of the Payment Channel that can be displayed in the list of available banks.
7.3	groupType	NO	string{1,30}	Type, used to group Payment Channels in their list. The parameter takes values from the node `gatewayGroups`.
7.4	bankName	NO	string{1,32}	Bank name.
7.5	iconUrl	NO	string{1,100}	Address from which the Payment Channel logo can be downloaded.
7.6	state	YES	string{1,64}	Information on the status of channel availability. Accepts values: - OK - channel available - TEMPORARY_DISABLED - channel temporarily unavailable (e.g. due to bank work) - DISABLED - channel unavailable (service suspended for an extended period)
7.7	stateDate	NO	string{1,19}	Time of last update of the status of the Payment Channel; example value: 2023-08-28 00:00:01. (CET time)
7.8	shortDescription	NO	string{1,200}	Optional field containing a brief description of the payment channel. Can be displayed when selected.
7.9	description	NO	string{1,1000}	Optional field describing the payment channel in detail (can be with HTML tags).
7.10	descriptionUrl	NO	string{1,200}	Optional field containing a link to an external page detailing the payment channel.
7.11	availableFor	YES	string{2,10}	The value of this field indicates for which customer the payment channel is intended: B2C - payment method for individuals B2B - payment method for companies B0TH - payment method for all customers. Based on this parameter, it should be decided whether a payment channel should be presented to the customer.

	name	required	type	description
7.12	requiredParams	NO	lista	A list of parameters required when selecting a payment method. For example, the start of a transaction for a payment method from the B2B group should include the parameter `Nip`. The required parameters are described in section: Starting a transaction with additional parameters. Currently, such parameters can be: `Nip` and `AccountHolderName`.
7.13	mcc	NO	object	Merchant Category Code. Optional node, additionally configurable. In special cases, for sites containing products from different categories, we can return a list of allowed and forbidden MCC codes so that the Merchant on his side can decide whether the payment method can be presented or not.
7.13.1	allowed	NO	list	List of permitted MCC codes.
7.13.2	disallowed	NO	list	List of prohibited MCC codes.
7.14	inBalanceAllowed	NO	boolean	Information on whether the channel can be used (after business arrangements) for prepaid balances (start of transaction with `TransactionSettlementMode=NONE` parameter).
7.15	minValidityTime	NO	integer	Minimum transaction validity time in minutes. Appears for channels where it takes longer than usual to establish payment status.
7.16	order	YES	integer	Recommended order of display of payment method.
7.17	currencies	YES	lista	A list containing the currencies available for the payment channel, with limits on the amounts.
7.17.1	currency	YES	string{3}	The currency that can be paid by this channel. If multiple currencies are available for a payment channel, the list will contain more than one item. Acceptable values only: PLN, EUR, GBP and USD. For counting the Hash values, the values of the following nodes are taken **currencies**.

	name	required	type	description
7.17.2	minAmount	NO	amount	The minimum amount of a transaction that can be paid through this channel. A full stop is used as decimal separator - '.' Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot. The field is present only for some channels, the value is expressed in the currency of the field **currency**. For counting the Hash values, the values of the following nodes are taken **currencies**.
7.17.3	maxAmount	NO	amount	The maximum amount of a transaction that can be paid through this channel. A full stop is used as decimal separator - '.' Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot. The field is present only for some channels, the value is expressed in the currency of the field **currency**. For counting the Hash values, the values of the following nodes are taken **currencies**.
7.18	buttonTitle	YES	string	Suggested message that should present on the 'pay' button after selecting a payment channel.

Example response


```
{
    "result": "OK",
    "errorStatus": null,
    "description": null,
    "gatewayGroups": [
        {
            "type": "PBL",
            "title": "Internet transfer",
            "shortDescription": "Select the bank from which you want to order the
payment".,
            "description": null,
            "order": 1,
            "iconUrl": null
        },
        {
            "type": "FR",
            "title": "Transfer details",
            "shortDescription": "Order a transfer using the details provided",
            "description": null,
            "order": 2,
            "iconUrl": null
        },
        {
            "type": "BNPL",
            "title": "Buy now, pay later",
            "shortDescription": "Buy now, pay later",
            "description": null,
            "order": 3,
```

```
            "iconUrl": null
        }
    ],
    "serviceID": "10000",
    "messageID": "2ca19ceb5258ce0aa3bc815e80240000",
    "gatewayList": [
        {
            "gatewayID": 106,
            "name": "PBL test payment",
            "groupType": "PBL",
            "bankName": "NONE",
            "iconURL": "https://testimages.autopay.eu/pomoc/grafika/106.gif",
            "state": "OK",
            "stateDate": "2023-10-03 14:35:01",
            "description": "Test payment",
            "shortDescription": null,
            "descriptionUrl": null,
            "availableFor": "BOTH",
            "requiredParams": ["Nip"],
            "mcc": {
                "allowed": [1234, 9876],
                "disallowed": [1111]
            },
            "inBalanceAllowed": true,
            "minValidityTime": null,
            "order": 1,
            "currencies": [
                {
                    "currency": "PLN",
                    "minAmount": 0.01,
                    "maxAmount": 5000.00
                }
            ],
            "buttonTitle": "Pay"
        },
        {
            "gatewayID": 9,
            "name": "Transfer from another bank",
            "groupType": "FR",
            "bankName": "BANK TEST",
            "iconURL": "https://testimages.autopay.eu/pomoc/grafika/9.gif",
            "state": "OK",
            "stateDate": "2023-10-03 14:35:02",
            "description": "<b>Fast transfer</b>",
            "shortDescription": "Fast transfer",
            "descriptionUrl": null,
            "availableFor": "BOTH",
            "requiredParams": [],
            "mcc": null,
            "inBalanceAllowed": true,
            "minValidityTime": null,
            "order": 2,
            "currencies": [
                {
                    "currency": "PLN"
                }
            ],
            "buttonTitle": "Generate transfer details"
        },
                {
            "id": 701,
            "name": "Pay later with Payka",
```

```
            "groupType": "BNPL",
            "bankName": "NONE",
            "iconUrl": "https://testimages.autopay.eu/pomoc/grafika/701.png",
            "state": "OK",
            "stateDate": "2023-10-03 14:37:10",
            "description": "<div class=\"payway_desc\"><h1>Cost details</h1><p>Pay later
- one-off up to 45 days (...). Offer details at: <a href="?r="https://payka.pl\"
target=\"_blank\">Payka.pl</a></p></div>",
            "shortDescription": "Pay later - in one go up to 45 days or in several equal
instalments",
            "descriptionUrl": null,
            "availableFor": "B2C",
            "requiredParams": [],
            "mcc": null,
            "inBalanceAllowed": false,
            "minValidityTime": 60,
            "order": 3,
            "currencies": [
                {
                    "currency": "PLN",
                    "minAmount": 49.99,
                    "maxAmount": 7000.00
                }
            ],
            "buttonTitle": "Pay"
        }
    ]
}
```


NOTE: The result of the method query should be refreshed every minute (calling **gatewayList** too often will increase the load on both systems without bringing much benefit).If there is no or an incorrect response, the last known and correct configuration of the Payment Channels should be displayed. This is the second reason to keep a temporary copy of the **gatewayList** on the Partner Service. An invalid response should be regarded as an empty response, a timeout, or an empty list of **gatewayGroups** or **gatewayList** nodes.

Requesting a list of currently available formal consents

Description

A description of the integration that allows the use of a payment list embedded in the service (or mobile app), without transition steps. In some cases, instead of a transition step, the standard behaviour of the system provides for a blocked start of the transaction.

The relevant formal content (i.e. information clauses and, if applicable, terms and conditions) should be displayed already at the time of selecting the payment method, and then confirmation of their display and, if applicable, acceptance (in the form of identifiers) should be sent to the Online Payment System.

The system allows the current list of obligations and related formal content to be queried remotely. To do this, call the **legalData** method (*https://{gate_host}/legalData*) with the appropriate parameters (in JSON format).

TIP: All parameters are transmitted using REST technology. The protocol is case-sensitive in both names and parameter values. The transmitted parameter values should be encoded in UTF-8.

List of available parameters

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	integer	Partner Service ID.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. based on UID). The field value must be unique for the Partner Service.
3	GatewayID	YES	integer{1,5}	Identifier of the Payment Channel through which the Customer intends to settle the payment.
4	Language	YES	string{2}	The language in which the content on the Website is presented. Acceptable values PL, EN, DE, CS, ES, FR, IT, SK, RO, HU, UK. The use of values other than PL should be confirmed during integration and depends on the actual choice (by the customer) of language in the Service.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Example message

```
    {
                "ServiceID": 102422,
                "MessageID": "11111111111111111111111111111111",
                "GatewayID": 1500,
                "Language": "PL",
"Hash":"61789013d932e2bc728d6206f7e9222b93e3176f7f07f6aa8cce1ccd65afaf0d"
        }
```

List of returned parameters

In response to the request, a list is returned (in the same HTTP session), containing further formal content in the form of: ID, type and wording of the content, its location in the Service, address to the rules and other additional information.

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	result	YES	string{1,5}	Response status. Acceptable values: - **OK** - **ERROR**
2	errorStatus	YES	string{1,100}	Error status, to be filled in in the event of an error (otherwise null).
3	description	YES	string{1,500}	Description of the error, to be filled in in the event of an error (otherwise null).
4	serviceID	YES	string{1,10}	Partner Service Identifier; derived from the method request.
5	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.
6	gatewayID	YES	integer{1,5}	Identifier of the Payment Channel with which the Customer can settle the payment.
7	language	YES	string{2}	The language in which the System returns content (clauses and regulations).
8	serviceModel	YES	string{1,20}	A field to denote the model in which the service works, for possible future guidance based on these values (currently with values: MERCHANT, PAYER). It should be ignored at this point.
nd.	regulationList	YES	list	A list containing the formal content available for the payment channel.
9	regulationID	YES	integer{1,10}	Formal content identifier, which (if accepted by the customer) should be passed in the start parameter **DefaultRegulationAcceptanceID** or **RecurringAcceptanceID** (respectively for the type **DEFAULT** and **RECURRING**). The method of acceptance is determined by the fields **showCheckbox** and **isCheckboxRequired**. **NOTE:** This value may repeat for calls with different **GatewayID**as the regulations are attributed to a group of payment channels rather than individual channels.

Hash order	name	required	type	description
10	type	YES	string{1,64}	Type of formal obligation. Projected values: - **DEFAULT** - clause(s) and payment terms and conditions in the service model provided by AP to the customer - **RECURRING** - clause(s) and the terms and conditions of the automatic payment. Value only available if an automatic payment service is configured - **PSD2** - clause dedicated to PSD2-type channels (value not used at the moment) - **RODO** - information clause on processing of personal data - **PRIVACY** - information clause on privacy policy
11	url	NO	string{1,500}	AAddress to the terms and conditions file (to be embedded in the Service itself). By default, if this is a formal obligation, it should be part of one of its clauses, i.e. the field **inputLabel**. ***TIP:*** *Appears when there is a document associated with consent.*
nd.	labelList	YES	list	A list containing the clauses available for a given formal obligation. The obligation may require the display of one or more contents.
12	labelID	YES	integer{1,10}	Clause identifier, transmitted for diagnostic purposes (can be ignored by the Partner).
13	inputLabel	YES	string{1,500}	The content of the clause to be displayed on the Service in conjunction with the relevant **regulationID**. In some cases, it may include a link to the regulations.
14	placement	NO	string{1,64}	Information suggesting where to place clauses. Current values: - - TOP_OF_PAGE - at the top of the site (e.g. near the logo/ top banner) - NEAR_PAYWALL - around the list of payment channels (directly above, below or beside) - ABOVE_BUTTON - above the "Start payment" button - BOTTOM_OF_PAGE - at the very bottom of the page (usually refers to RODO, PRIVACY information clauses)
15	showCheckbox	YES	boolean	Information on whether a clause should be displayed next to a checkbox for user acceptance.

Hash order	name	required	type	description
16	checkboxRequired	YES	boolean	Information on whether the checkbox displayed must be ticked by the user to proceed to payment. **NOTE:** If the value is true, the 'Start payment' button should be blocked until the checkbox is ticked.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

Example response


```
    {
                "serviceID": "102422",
                "messageID": "11111111111111111111111111111111",
                "gatewayID": "1500",
                "language": "PL",
                "serviceModel": "PAYER",
                "regulationList": [
                        {
                                "regulationID": 6288,
                                "type": "RECURRING",
                                "url": "https://host/path?params",
                                "labelList": [
                                        {
                                                "labelID": 1,
                                                "inputLabel": "<ul><li>\r\nI have read
and accept <a id=\"regulations_pdf\" target=\"_blank\"
href=https://{gate_host}/path?params>Regulations for the payment services</a> and <a
class=\"privacy-policy\" href=\"https://{gate_host}/polityka-prywatnosci.pdf\"
target=\"_blank\">Privacy Policy</a></li><li>\r\nI want the service to be provided
without delay and, in the event of withdrawal, I know that I will not be reimbursed for
the services provided at my request until the withdrawal has taken place\r\n</li></ul>",
                                                "placement": "ABOVE_BUTTON",
                                                "showCheckbox": true,
                                                "checkboxRequired": true
                                        }
                                ]
                        },
                        {
                                "regulationID": 1,
                                "type": "PRIVACY",
                                "labelList": [
                                        {
                                                "labelID": 1,
                                                "inputLabel": "Autopay uses cookies. By
remaining on this website, you consent to the use of cookies in accordance with the <a
class=\"privacy-policy\" href="?r="https://{gate_host}/polityka-prywatnosci.pdf\"
target=\"_blank\">Autopay S.A. privacy policy.</a> You can manage cookies yourself by
changing the settings of your browser or device software accordingly."
                                                "placement": "BOTTOM_OF_PAGE",
                                                "showCheckbox": false,
                                                "checkboxRequired": false
                                        }
```

```
                                ]
                        }
                ],
                "hash":
"61789013d932e2bc728d6206f7e9222b93e3176f7f07f6aa8cce1ccd65afaf0d",
                "result": "OK",
                "errorStatus": null,
                "description": null
        }
```

Description of response handling

As the formal requirements for the content of the clauses, their distribution and the method of acceptance depend on the payment channel used, this method should be invoked each time it is selected (hence the mandatory parameter of the **GatewayID**).

Relevant content and behaviour should be dynamically adapted to responses from the System (e.g. a required checkbox with an information clause and a link to the terms and conditions should appear). Of course, for the application to run quickly, the use of a cache to remember responses of recent calls (e.g. for 1 minute) is welcome.

The consent displayed (and possibly approved) at the time of the transition to payment should be confirmed in the System by attaching to the message of the start of the transaction in the start parameter of its identifier (and thus the corresponding value of the **regulationID**).

Depending on the value of the **type** field of the regulations:

- for the displayed/accepted clause of **type=DEFAULT**:

a. to the parameter **DefaultRegulationAcceptanceID** its value of **regulationID**;

b. the **DefaultRegulationAcceptanceState** parameter should be set to **ACCEPTED** and

c. the **DefaultRegulationAcceptanceTime** parameter should be set to the value corresponding to the moment when the consent is accepted by ticking the checkbox and clicking the ‘Start payment’ button

- - for the displayed/accepted clause of **type=RECURRING**:

a. to the **RecurringAcceptanceID** parameter should be its **regulationID** value;

b. the **RecurringAcceptanceState** parameter should contain the value **ACCEPTED** and

c. the **RecurringAcceptanceTime** parameter should be set to the value corresponding to the moment when the consent is accepted by ticking the checkbox and clicking the "Start payment' button

NOTE: The fields (e.g. **serviceModel, url, labelID**) and field values (e.g. **PSD2, RODO, PRIVACY**) of the **legalData** method are not required to be handled, but the possibility of their occurrence in the request response should be foreseen.

Balance enquiry

Description

For all services, it is possible to query the current balance. For this purpose, the method **balanceGet** https://{gate_host}/webapi/balanceGet with the relevant parameters. All parameters are passed via the POST method (`Content-Type: application/x-www-form-urlencoded`). The protocol is case-sensitive in both parameter names and values. The values of passed parameters should be encoded in UTF-8.

List of available parameters for the current balance

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
1	BalancePointID	YES	string{1,10}	Settlement Point Identifier. **NOTE:** Required one of the fields **ServiceID** or **BalancePointID**. Specifying both will result in processing of the request being stopped and an http error.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. based on UID). The field value must be unique for the Partner Service.
3	PlenipotentiaryID	NO	string{8,8}	Proxy ID. If present, the proxy's shared key is used to calculate the Hash, rather than the service/billing point's primary key. It also affects the Hash in response to this message. ***IMPORTANT!*** *The use of this field requires special business arrangements.*
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. A shared key assigned to the configuration identifier used (Service or Settlement Point) is used. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Response to request for current balance

In response to the request, a text is returned (in the same HTTP session) in XML format, containing an

acknowledgement of the operation for execution or a description of the error.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balanceGet>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <balance>Balance</balance>
            <currency>Currency</currency>
            <hash>Hash</hash>
    </balanceGet>
```

lub

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balanceGet>
            <balancePointID>BalancePointID</balancePointID>
            <messageID>MessageID</messageID>
            <balance>Balance</balance>
            <currency>Currency</currency>
            <hash>Hash</hash>
    </balanceGet>
```

List of response fields

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID. Derived from a method request.
1	balancePointID	YES	string{1,10}	Settlement Point Identifier. Derived from a method request. **NOTE:** One of the fields will be returned **ServiceID** or **BalancePointID**.
2	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.
3	balance	YES	amount	Balance value; a dot is used as decimal separator - '.' Format: 0.00.
4	currency	YES	string{1,3}	Balance currency. Acceptable values only: PLN, EUR, GBP and USD.

Hash order	name	required	type	description
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. A shared key assigned to the configuration identifier used (Service or Settlement Point) is used. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

Balance supply

Description

For services that have a balance in the System, it is possible to perform a recharge operation for future refunds. To do this, start the transaction with the **TransactionSettlementMode** parameter with the value **NONE** and **Amount** indicating the **recharge amount**.

Using **Pre-Transaction** will allow the finished transaction to be simply delivered to the payer (e.g. by providing the Partner's accounting address in CustomerEmail).

NOTE: The service must be agreed with the business custodian. In the Marketplace model, it will furthermore be necessary to build a basket of products indicating which Settlement Point (BalancePointID) is to be fed.

Balance withdrawals

Description

For services that have a balance in the System, it is possible to perform an operation to withdraw all or part of the balance to a defined account for settlement. For this purpose, it is necessary to call the method *balancePayoff* (https://{gate_host}/settlementapi/balancePayoff) with the relevant parameters.

All parameters are passed via the POST method (Content-Type: application/x-www-form-urlencoded). The protocol is case-sensitive in both parameter names and values. The values of passed parameters should be encoded in UTF-8.

List of available parameters for balance withdrawals

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

NOTE: One of the fields **ServiceID** or **BalancePointID** is required. W
Specifying both will cause the request processing to stop and an http error.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
1	BalancePointID	YES	string{1,10}	Settlement Point Identifier.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 alphanumeric characters of the Latin alphabet (e.g. based on UID), the field value must be unique and indicate a specific payment order in the Partner Service. Verification of uniqueness on the part of the System allows the MessageID to be repeated in the event of communication problems (repeating this value will result in confirmation of the order, without re-execution in the System).
3	Amount	NO	amount	Amount of withdrawal from the balance (cannot be greater than the current balance of the service); failure to specify this parameter results in withdrawal of the total funds accumulated on the balance; a dot - '.' - is used as decimal separator. Format: 0.00.
4	Currency	NO	string{1,3}	Payment currency. The default currency is PLN (the use of another currency must be agreed during integration). One currency is supported within ServiceID. Acceptable values only: PLN, EUR, GBP and USD.

Hash order	name	required	type	description
5	CustomerNRB	NO	string{26}	The account number to which the balance withdrawal is to be made. By default, the disbursement configuration does not allow this value to be defined in the method request **balancePayoff**. Such a request must be made during integration. **NOTE:** In some models, the use of this field can only occur for requests from a list of trusted IPs and the use of one of 3 additional elements: - securing the communication with a client certificate or - securing the communication with an IPSec tunnel or - use of the parameter value **CustomerNRB** from the list of trusted accounts Failure to comply with them results in a CUSTOMER_NRB_NOT_AUTHORIZED error. System panel administrators (https://portal.autopay.eu/admin) can update the trusted IP and NRB lists themselves in the **Access control** service configuration. Only digits are allowed. If, during integration, the use of accounts outside Poland was established, then the field transfers IBAN and the expected field data range changes to: alphanumeric Latin characters (min. 15, max. 32 characters).
6	SwiftCode	NO	string{8,11}	The swift code corresponding to the account number given. Only digits allowed. Parameter to be provided if the use of non-Polish accounts was established during integration.

Hash order	name	required	type	description
7	ForeignTransferMode	NO	string{4,5}	The system by which the foreign settlement transfer is to be made: - SEPA (Single Euro Payments Area) - possible transfers in Euro currency within the European Union Member States, as well as other countries within the Old Continent, e.g. Iceland, Liechtenstein, Norway, Switzerland, Monaco or Andorra, - SWIFT - foreign transfers not feasible with SEPA (e.g. different currency than Euro), involves higher transfer costs than with SEPA. Acceptable values: **SEPA** and **SWIFT**. Parameter to be provided if the use of accounts outside Poland has been established during integration.
8	ReceiverName	NO	string{35}	The name of the payee of the balance withdrawal. By default, the disbursement configuration does not allow this value to be defined in the method request **balancePayoff**. Such a request must be made during integration. Acceptable alphanumeric Latin characters and characters in the range: *ĘęÓóĄąŚśŁłŻżŹźĆćŃń\s.-/,!()=[]{};:?*
9	Title	NO	string{32}	Balance disbursement title. By default, the disbursement configuration does not allow this value to be defined in the method request **balancePayoff**. Such a request must be made during integration. In certain cases, beyond the control of the AP, this title may be independently modified by the Bank. Acceptable alphanumeric Latin characters and characters in the range: *ĘęÓóĄąŚśŁłŻżŹźĆćŃń\s.-/,!()*", where the "/" sign will be replaced by a "-" for outgoing transactions.
10	RemoteRefID	NO	string{1,20}	The alphanumeric identifier of the incoming transaction assigned by the System and transmitted to the Partner in the ITN message of the incoming transaction. The value in this message is used to indicate the payment instrument (card, account, etc.) to be used to make the withdrawal.

Hash order	name	required	type	description
11	InvoiceNumber	NO	string{1,100}	The number of the financial document in the Service.In this message, the value is used to indicate the correction invoice associated with the payment.
12	PlenipotentiaryID	NO	string{8,8}	Proxy ID. If present, the proxy's shared key is used to calculate the Hash, rather than the service/billing point's primary key. It also affects the Hash in response to this message. ***IMPORTANT!*** *The use of this field requires special business arrangements.*
nd.	Hash	YES	string{1,128}	The shared key assigned to the configuration identifier used (Service or Settlement Point) is used. Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Response to request

Upon receipt of a balance withdrawal request, the System performs an initial verification against the fields and their values sent in the message and saves the order for execution. In response to the request, an XML-formatted text is returned (in the same HTTP session), containing a confirmation that the order has been saved in the queue for execution or a description of the error (the structure of the error message is described in section Error messages.

NOTE: Confirmation of receipt of an order is not equivalent to its actual execution. A balance withdrawal can be processed up to 30 minutes after the withdrawal request is sent and may not always be successful. In the event of problems encountered during the processing of a balance withdrawal (e.g. insufficient funds in the balance), a report is sent the following working day containing information about the balance withdrawals that failed.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balancePayoff>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <hash>Hash</hash>
    </balancePayoff>
```

or

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <balancePayoff>
```

```
                <balancePointID>BalancePointID</balancePointID>
                <messageID>MessageID</messageID>
                <hash>Hash</hash>
        </balancePayoff>
```

Description of the fields

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID. Derived from a method request.
1	balancePointID	YES	string{1,10}	Settlement Point Identifier. Derived from a method request. **NOTE:** One of the fields will be returned **ServiceID** or **BalancePointID**.
2	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

NOTE: All responses other than those assumed (i.e. with invalid fields, in particular an empty or invalid **hash**) may be considered an error. In the event that the System authorises the sender of a return message but the operation fails, the response will follow the pattern described in Error messages. In the event of a communication error or insufficient balance (e.g. ON_DEMAND_ERROR), the order can be retried. If the balance is blocked (BALANCE_DISABLED) or the configuration is not active (PARTNER_DISABLED), the order should not be renewed.

IMPORTANT! In the event of an error returned in response to a balance withdrawal request, the request can be retried, but please note that any withdrawal request not ending in an error can be executed. In case of connection problems, exceeding the maximum response time, the request can be repeated with the same MessageID without fear of duplicating the order.
In the case of a blocked balance (BALANCE_DISABLED) or inactive configuration (PARTNER_DISABLED), the request should not be renewed. 3 errors in a row (regardless of the cause) will block the on-demand disbursement service for the specified message sender IP for 10 minutes - calls during this time for this IP will end with a TEMPORARY_DISABLED error.

Transaction refunds

Description

For services with a balance in the System, it is possible to perform the operation of returning to the Customer the whole or part of the amount paid for the indicated transaction. A successful refund of the whole transaction can be performed once (in case of a repeated attempt to order a refund of the same transaction, the System returns a properly described error). Refunds of part of the amount of a transaction can be performed on it multiple times, as long as their sum does not exceed the amount of the deposit.

To perform a transaction refund, call the method **transactionRefund** (https://{gate_host}/settlementapi/transactionRefund) with the relevant parameters. All parameters are passed via the POST method (`Content-Type: application/x-www-form-urlencoded`). The protocol is case-sensitive in both the names and values of the parameters. The values of the passed parameters should be encoded in UTF-8.

List of available parameters

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 alphanumeric characters of the Latin alphabet (e.g. based on UID), the field value must be unique and indicate a specific payment order in the Partner Service. Verification of uniqueness on the part of the System allows the **MessageID** to be repeated in the event of communication problems (repeating this value will result in confirmation of the order, without re-execution in the System).
3	RemoteID	YES	string{1,20}	The alphanumeric identifier of the returned input transaction assigned by the System and passed to the Partner in the ITN message of the input transaction.
4	Amount	NO	amount	Amount of withdrawal from the balance (cannot be greater than the current balance of the service); failure to specify this parameter results in withdrawal of the total funds accumulated on the balance; a dot - '.' - is used as decimal separator. Format: 0.00. In the Marketplace model, the field must be blank (total return). Otherwise, it would not be possible to indicate the billing point(s) to be charged for such an operation. With a total return, the balance is deducted according to the sum of the product amounts of the respective clearing point.

Hash order	name	required	type	description
5	Currency	NO	string{1,3}	Payment currency. The default currency is PLN (the use of another currency must be agreed during integration). One currency is supported within ServiceID. One currency is supported. Acceptable values only: PLN, EUR, GBP and USD.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

In response to the request, an XML-formatted text is returned (in the same HTTP session), either confirming the operation or describing the error (described below).

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <transactionRefund>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <hash>Hash</hash>
    </transactionRefund>
```

Description of the fields

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID. Derived from a method request.
2	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

The option to make refunds to paid transactions is possible up to 12 months back, counting from the date the transaction was initiated. An exception is BLIK payments, which, due to time constraints on the part of the payment channel provider, can be refunded up to 6 months back.

The above deadlines apply to the processing of returns via the administrative panel and the **transactionRefund**e.g. via its own administrative tools. When these are exceeded, an error will be returned (**TRANSACTION_TOO_OLD_TO_REFUND**). The scheme of operation is analogous to that for balance withdrawals. That is, the System accepts the order and asynchronously processes it within a maximum of 30 minutes, and in the event of failure, information about the operations ending in an error is sent in a report the following working day.

In the event of connection problems, exceeding the maximum response time, the request can be renewed with the same **MessageID** without fear of duplicate requests. All responses other than the assumed one (i.e. with incorrect fields, in particular blank or incorrect **hash**). If the System authorises the sender of the return message, but the operation fails, an error message will be returned in response.

Product returns

Description

For sites with a balance in the System and specifying the **productID** parameter in the product basket, it is possible to perform an operation to return to the Customer the whole or part of the amount paid for the indicated product. A successful return of the whole amount of a product can be performed once (in case of a repeated attempt to order the return of the same product, the System returns a properly described error). Partial product refunds can be performed on the product multiple times, as long as their total does not exceed the amount paid for the product.

To perform a product return, call the method **productRefund**(https://{gate_host}/settlementapi/productRefund) with the appropriate parameters. All parameters are passed using the POST method (Content-Type: application/x-www-form-urlencoded). The protocol is case-sensitive in both parameter names and values. The values of the passed parameters should be encoded in UTF-8.

List of parameters

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 alphanumeric characters of the Latin alphabet (e.g. based on UID), the field value must be unique and indicate a specific payment order in the Partner Service. Verification of uniqueness on the part of the System allows the **MessageID** to be repeated in the event of communication problems (repeating this value will result in confirmation of the order, without re-execution in the System).
3	RemoteID	YES	string{1,20}	The alphanumeric identifier of the returned input transaction assigned by the System and passed to the Partner in the ITN message of the input transaction.

Hash order	name	required	type	description
4	ProductID	YES	string{1,36}	Identifier of the returned product.
5	Amount	NO	amount	Amount of the refund (cannot be greater than the amount of the product and the current balance of the service + the amount of the refund commission, if any). Failure to specify this parameter will result in the return to the customer of the entire amount paid for the returned product; a full stop - '.' - is used as a decimal separator. Format: 0.00.
6	Currency	NO	string{1,3}	Payment currency. The default currency is PLN (the use of another currency must be agreed during integration). One currency is supported within ServiceID. Acceptable values only: PLN, EUR, GBP and USD.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

Response to request

In response to the request, a text is returned (in the same HTTP session) in XML format, containing an acknowledgement of the operation for execution or a description of the error.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <productRefund>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <hash>Hash</hash>
    </productRefund>
```

Description of the fields

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID. Derived from a method request.
2	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.

Hash order	name	required	type	description
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

The option to make refunds to paid transactions is possible up to 12 months back, counting from the date the transaction was initiated. An exception is BLIK payments, which, due to time constraints on the part of the payment channel provider, can be refunded up to 6 months back.

The above deadlines apply to the processing of refunds via the administration panel and **productRefund**, e.g. via your own administration tools. If these are exceeded, an error will be returned (**TRANSACTION_TOO_OLD_TO_REFUND**).

The scheme of operation is analogous to that for balance withdrawals. That is, the System accepts the order and asynchronously processes it within a maximum of 30 minutes, and in the event of a failure, information about the operations ending in an error is sent in a report the following business day.

In the event of connection problems, exceeding the maximum response time, the request can be renewed with the same MessageID without fear of duplicate requests. All responses other than the assumed one (i.e. with incorrect fields, in particular blank or incorrect **hash**). If the System authorises the sender of the return message, but the operation fails, an error message will be returned in response.

Enquiry about the status of a refund or a withdrawal from the balance

Description

Once the refund or balance withdrawal has been made, we have the option of verifying the status of the billing transaction and
what identifier has been assigned to it by the Online Payment System. For this purpose, call the method *outDetails* (https://{gate_host}/settlementapi/outDetails) with the corresponding parameters. All parameters are passed via the POST method (Content-Type: application/x-www-form-urlencoded). The protocol is case-sensitive in both parameter names and values. The values of the passed parameters should be encoded in UTF-8.

List of available parameters for balance withdrawals

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

NOTE: One of the fields **ServiceID** or **BalancePointID** is required.
Specifying both will cause the request processing to stop and an http error.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.

Hash order	name	required	type	description
1	BalancePointID	YES	string{1,10}	Settlement Point Identifier.
2	MessageID	YES	string{32}	Enter the same message identifier that was sent previously when ordering a refund or balance withdrawal.
3	Method	YES	enum	The operation for which the settlement transaction was created: **BALANCE_PAYOFF** - withdrawal from the balance **TRANSACTION_REFUND** - refund of transactions **PRODUCT_REFUND** - product return
nd.	Hash	YES	string{1,128}	The shared key assigned to the configuration identifier used (Service or Settlement Point) is used. The value of the hash function for the message calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Response to request

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <outDetails>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
              <status>Status</status>
            <remoteOutId>RemoteOutId</remoteOutId>
            <hash>Hash</hash>
    </outDetails>
```

or

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <outDetails>
            <balancePointID>BalancePointID</balancePointID>
            <messageID>MessageID</messageID>
            <status>Status</status>
            <remoteOutId>RemoteOutId</remoteOutId>
            <hash>Hash</hash>
    </outDetails>
```

Description of the fields

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	Partner Service ID. Derived from a method request.
1	balancePointID	YES	string{1,10}	Settlement Point Identifier. Derived from a method request. **NOTE:** One of the fields will be returned **ServiceID** or **BalancePointID**.
2	messageID	YES	string{32}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request.
3	status	YES	enum	Processing status: **NEW** - Awaiting in the queue for reprocessing. **PROCESSING** - During the process **ERROR** - Processing failed e.g. no funds in the balance **DONE** - The processing was successful.
4	remoteOutId	NO	string{1,20}	The alphanumeric identifier of the settlement transaction assigned by the Online Payment System.
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

Transaction summary page

Description

The system allows a summary of the transaction to be displayed to the customer. For this purpose, the partner can build trigger links with the appropriate parameters method. **confirmation** (https://{gate_host}/web/confirmation/payment). All parameters are transferred using the GET method. The protocol is case sensitive in both parameter names and values. The values of the passed parameters should be encoded in UTF-8.

List of parameters for the transaction summary method

Redirecting a Customer with correct parameters will result in the display of a summary of the transaction (with content depending on its status) or information about its absence (if the System does not find it).

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	OrderID	YES	string{32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

NOTE: The service must be activated after agreement with the business supervisor. It is possible to change the content of the messages or adapt the graphic layout (these are subject to agreement in working form each time during integration).

Enquiry about the status of a transaction

Description

For all services, it is possible to query the current balance. For this purpose, the method **transactionStatus** (https://{gate_host}/webapi/transactionStatus) with the relevant parameters. All parameters are passed via the POST method (Content-Type: application/x-www-form-urlencoded). The protocol is case-sensitive in both parameter names and values. The values of passed parameters should be encoded in UTF-8.

List of parameters available for transaction status

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	OrderID	YES	string{32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Partner Service**.

HTTP header for transaction status request

For correct querying, a defined HTTP header with appropriate content must be sent along with the

passed parameters. The attached header should be named '*BmHeader*' and have the following value '*pay-bm*', in its entirety, should be as follows '*BmHeader: pay-bm*'. In the case of a valid message, an (in the same HTTP session) text in XML format is returned, containing all transactions with the indicated OrderID, together with basic information about them.

TIP: Such a situation may occur e.g. when the Customer changes the Payment Channel, calls up the same transaction start again from the browser history, etc. The system allows blocking such cases, but the option is not recommended (it would not be possible to pay the abandoned transaction).

IMPORTANT! If the query relates to an orderID that occurs in more than 50 transactions of a given service, a response (XML) with error code 403 is returned.

Response structure when the limit of transactions with the same orderId is reached:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<transaction>
<reason>LIMIT_REQUESTED_TRANSACTIONS_WITH_THE_SAME_ORDER_ID_AND_SERVICE_ID_EXCEEDED</rea
son>
    <description>Transaction limit 50 with the same order id {{ORDER_ID}} and service id
{{SERVICE_ID}} exceeded. Requested count {{TRANSACTION_AMOUNT}}
    </description>
</transaction>
```

List of fields for a transaction status query

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	YES	string{1,10}	The Partner Service ID, assigned during service registration, uniquely identifies the Partner Service in the Online Payment System.
2	orderID	YES	string{1,32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction.
3	remoteID	YES	string{1,20}	An alphanumeric transaction identifier assigned by the Online Payment System.
5	amount	YES	amount	The transaction amount uses a full stop as decimal separator - '.' Format: 0.00; maximum length: 14 digits before the dot and 2 after the dot.
6	currency	YES	string{1,3}	Transaction currency.

Hash order	name	required	type	description
7	gatewayID	NO	string{1,5}	Identifier of the Payment Channel through which the customer settled the payment.
8	paymentDate	YES	string{14}	The time when the transaction was authorised, transmitted in the format YYYYMMDDhhmmss. (CET time)
9	paymentStatus	YES	enum	Transaction authorisation status, takes values (description of status changes further on): **PENDING** – transaction initiated. **SUCCESS** - correct authorisation of the transaction, the Partner Service will receive the funds for the transaction - goods/services can be issued. **FAILURE** – transaction not completed correctly.
10	paymentStatusDetails	NO	string{64}	Detailed transaction status, value can be ignored by the Partner Service. ***TIP:*** *Detailed description in part Detailed transaction statuses.*
nd.	hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

NOTE: Since the method can return multiple transactions, successive transactions are downloaded into the Hash (according to the order of occurrence of the transactions in the response). Within a given transaction, the order according to the number next to the field (excluding the ServiceID parameter, level up) applies.

Example of a hash function string

```
            Hash = funkcja(<serviceID> + „|” +
        <orderID1> + „|” + <remoteID1> + „|” + <amount1> + „|” + <currency1> + „|” +
<gatewayID1> + „|” + <paymentDate1> + „|” + <paymentStatus1> + „|” +
<paymentStatusDetails1> + „|” +
        <orderID2> + „|” + <remoteID2> + „|” + <amount2> + „|” + <currency2> + „|” +
<gatewayID2> + „|” + <paymentDate2> + „|” + <paymentStatus2> + „|” +
<paymentStatusDetails2> + …
```

Handling of transaction status query responses - proposal to handle multiple transactions in response

Condition	Meaning	Proposed message to the customer
Exactly one transaction with paymentstatus=SUCCESS	Correctly paid transaction.	The system has correctly registered the payment.
More than one transaction with paymentstatus=SUCCESS	Multiple paid transactions.	The system has registered more than one payment.
There is a RemoteID with paymentstatus=PENDING and there is none with paymentstatus=SUCCESS	The transaction is pending payment.	The system is awaiting payment.
There is at least one transaction but no status other than paymentstatus=FAILURE	Transaction cancelled.	The system has registered a payment cancellation or failure to authorise payment.
No transaction or other error	Failure to find a transaction.	The transaction was not found.

Cancellation of an unpaid transaction

Description of cancellation of an unpaid transaction

For all services it is possible to cancel a started but unpaid transaction by calling the method *transactionCancel* (https://{gate_host}/webapi/transactionCancel) with the corresponding parameters. All parameters are passed via the POST method (Content-Type: application/x-www-form-urlencoded).

The protocol is case-sensitive in both names and values of parameters. Values of transmitted parameters should be encoded in UTF-8.

List of parameters available for cancelling an unpaid transaction

IMPORTANT! The order of attributes for the Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	ServiceID	YES	string{1,10}	Partner Service ID.
2	MessageID	YES	string{32}	Pseudo-random message identifier with a length of 32 Latin alphanumeric characters (e.g. based on UID). The field value must be unique for the Partner Service.
3	RemoteID	NO	string{1,20}	An alphanumeric transaction identifier assigned by the System and transmitted to the Partner in the ITN message of the incoming transaction. Its indication will result in the cancellation of only one transaction with the indicated **RemoteID** if payment is pending (status **PENDING**).

Hash order	name	required	type	description
4	OrderID	NO	string{32}	The transaction identifier assigned in the Partner Service and communicated at the start of the transaction. Its indication (no **RemoteID**) will result in the cancellation of all pending transactions (**PENDING** status) with the indicated **OrderID** (and **ServiceID**). **NOTE:** Required one of the fields **OrderID** or **RemoteID**. Specifying both will result in processing of the request being stopped and an http error.
nd.	Hash	YES	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**.

Heading for cancellation of an unpaid transaction

For correct querying, a defined HTTP header with appropriate content must be sent along with the passed parameters. The attached header should be named '*BmHeader*' and have the following value '*pay-bm*'. In its entirety, should be as follows 'BmHeader: pay-bm'.

In the case of a valid message, an XML-formatted text is returned (in the same HTTP session), containing a confirmation of the operation or a description of the error.

Confirmation structure (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
    <transaction>
            <serviceID>ServiceID</serviceID>
            <messageID>MessageID</messageID>
            <confirmation>ConfStatus</confirmation>
            <reason>Reason</reason>
            <hash>Hash</hash>
    </transaction>
```

List of parameters for cancelling an unpaid transaction

IMPORTANT! The order of attributes for Hash enumeration must follow their numbering.

Hash order	name	required	type	description
1	serviceID	NO	string{1,32}	Partner Service ID. Derived from a method request. Required for confirmation=CONFIRMED
2	messageID	NO	string{1,20}	Pseudo-random message identifier of 32 Latin alphanumeric characters in length (e.g. based on UID). Derived from the method request. Required for confirmation=CONFIRMED

Hash order	name	required	type	description
3	confirmation	YES	string{1,100}	Order acknowledgement status. It can take two values: - CONFIRMED – the operation was successful - NOTCONFIRMED – operation failed
4	reason	NO	string{1,1000}	Explanation of the details of the processing of the request.
nd.	hash	NO	string{1,128}	Value of message digest function calculated as described in section Security of transactions. **Mandatory verification of compliance of the calculated abbreviation by the Service**. Required for confirmation=CONFIRMED

Responses to requests for cancellation of transactions

If the message is syntactically correct, the System will return one of the following pairs describing the result of the processing. In addition to its interpretation, it is recommended to check the status of the transaction (method **transactionStatus**).

Please note that once at least one transaction has been successfully cancelled, it is not possible to start a new one or to continue a previously started transaction with the same **OrderID**.

cofnfirmation	reason	details
CONFIRMED	CANCELED_FULLY	For the OrderID indicated: all pending deposit transactions have been cancelled. For the indicated RemoteID: the transaction has been cancelled.
CONFIRMED	CANCELED_PARTIALLY	For the OrderID indicated: at least one transaction was cancelled, but there were transactions that could not be cancelled (e.g. they were already paid for). For the indicated RemoteID: such a response does not occur.
NOTCONFIRMED	INCORRECT_PAYMENT_STATUS	At least one indicated transaction was found, but none could be cancelled (e.g. there was no pending transaction).
NOTCONFIRMED	TRANSACTION_NOT_FOUND	The indicated transaction was not found.
NOTCONFIRMED	OTHER_ERROR	There was another error when processing the request.

Error messages

Description of error messages

All error messages will be returned as an XML document, containing the error code, its name and description. Due to the high variability of possible errors, full documentation of errors is not maintained.

> **TIP:** The **description** field, describes each error in detail (the **statusCode** and **name** fields can be ignored).

Example of error (XML)

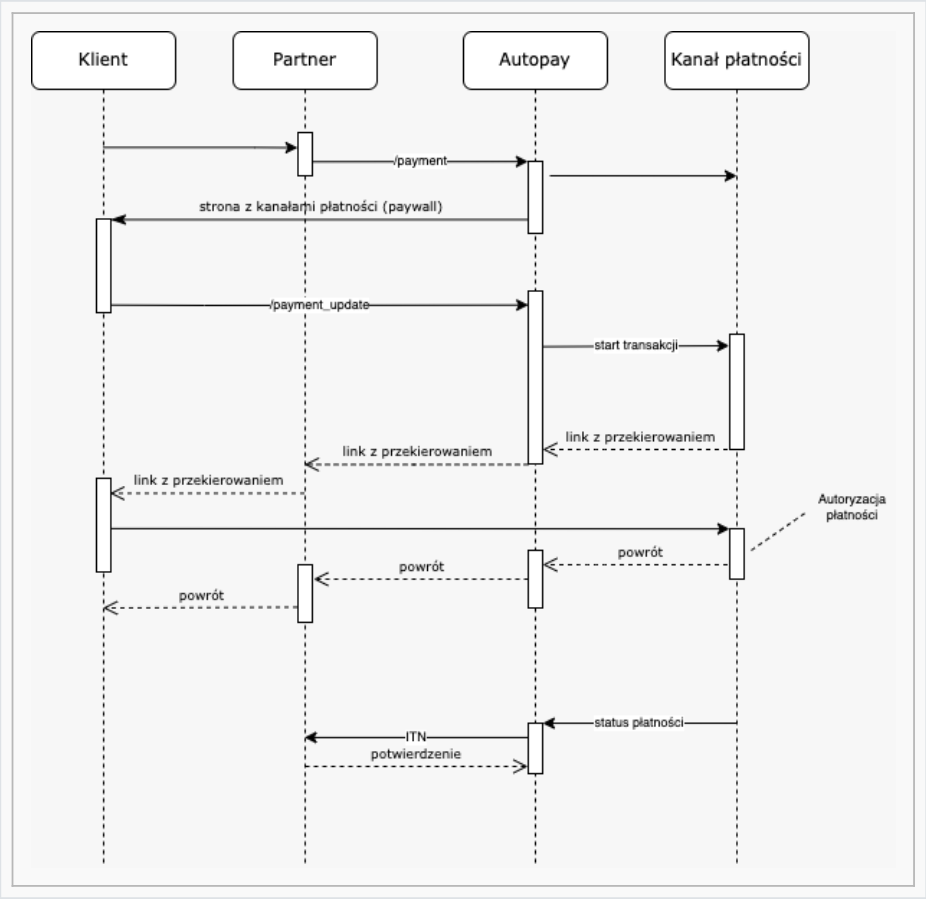
```
    <?xml version="1.0" encoding="UTF-8"?>
        <error>
                <statusCode>55</statusCode>
                <name>BALANCE_ERROR</name>
                <description>Wrong services balance! Should be 100 but is
40</description>
        </error>
```

Transaction and settlement patterns

This section presents models, event scenarios and flow information.

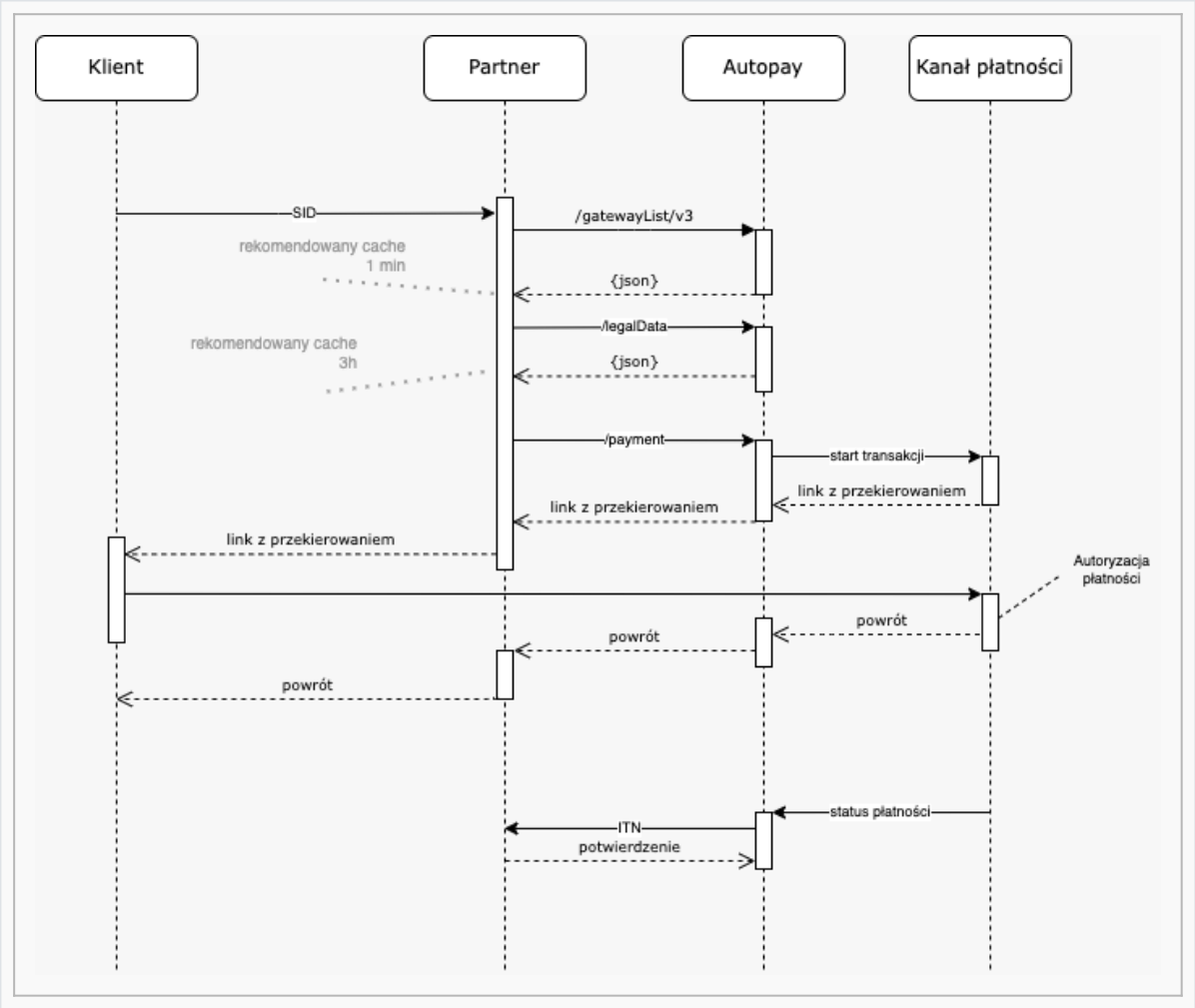
Model Paywall

The simplest model, where the choice of payment channels is on the Autopay (paywall) pages.



Model WhiteLabel

It is characterised by the presentation of payment channels on the Merchant side. To do this, obtain a list of channels with descriptions from the `gatewayList/v3` service and obtain a list of the necessary rules and regulations for each payment channel (`/legalData`). The rest of the process is the same as in the Paywall model.



Extended structure of services and billing points

The partner structure consists of at least 1 service (identified by an identifier **ServiceID**) and any number of points accounts (identified by an identifier **idBalancePoint**).

- Services are usually the sources of Payment Links (website, mobile app, emails, etc.). Services also distribute traffic relating to different industries (invoice payments, e-Commerce purchases, etc.). As the transaction is identified by a pair of **OrderID** and **ServiceID**, it can be said that "the service corresponds to the level of the transaction".

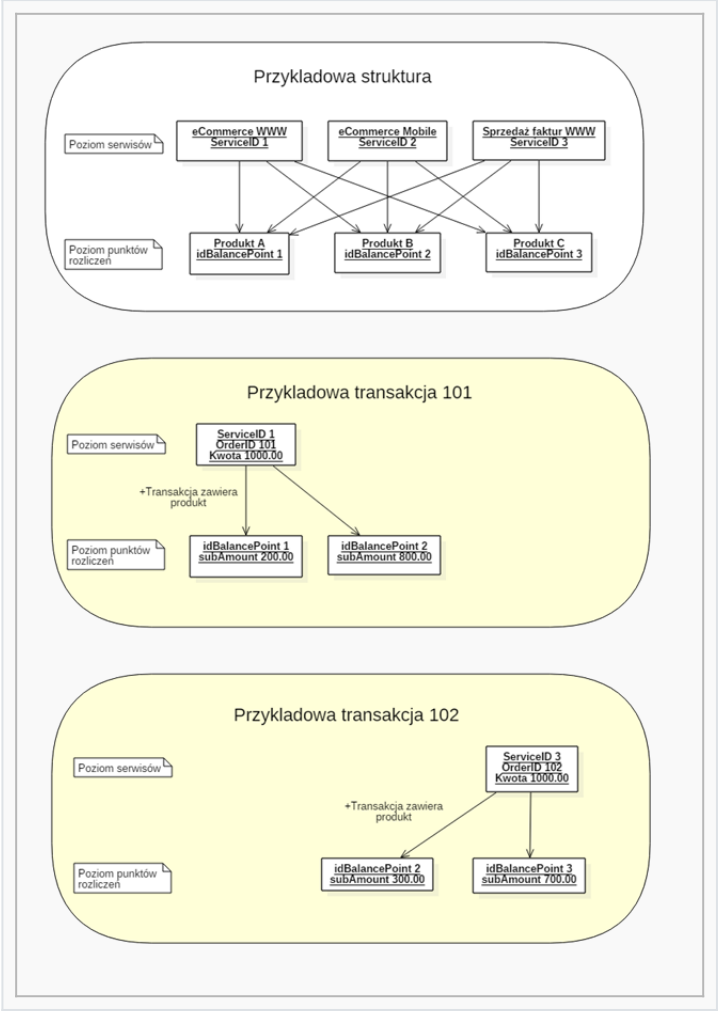
- Settlement points are defined if there is a need to distinguish in some way the constituent payments (e.g. by indicating them in reports or independent settlement). Since the product of a transaction is identified by its associated settlement point (**idBalancePoint**), it can be said that the "billing point corresponds to the level of the product".

The product basket (and therefore also the billing points) may not be present in the structure describing the Partner. The reason for adding billing points to the structure influences the decision on the billing model:

- The need to distinguish the components of a deposit in the list of transactions (in reports) does not necessarily entail separate settlement of each product or billing point; in this situation, service-level billing models (batch or after each deposit) are usually sufficient.

- The need to separate the component settlements of a deposit results in the use of a settlement model at the settlement point level (aggregate or after each deposit).

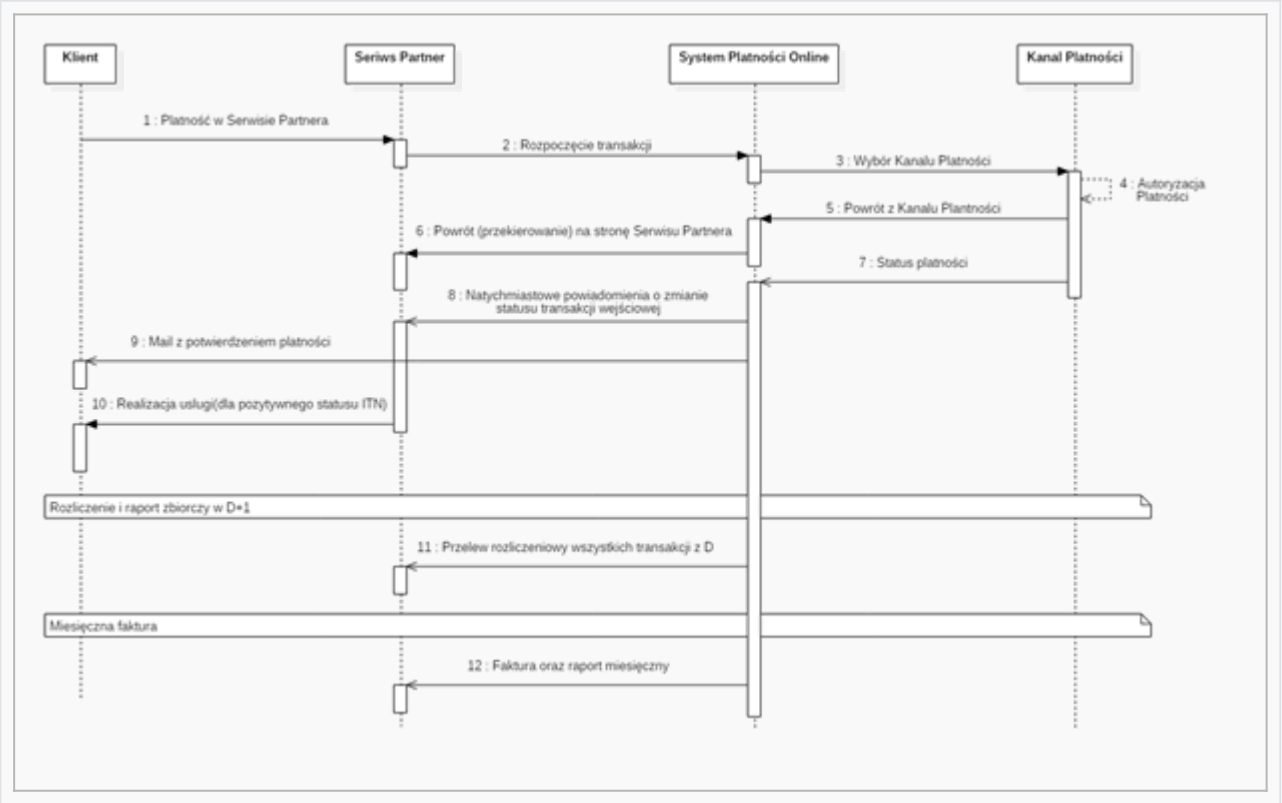
  Below is an illustration of an example structure (without indicating a specific settlement model)



Settlement models

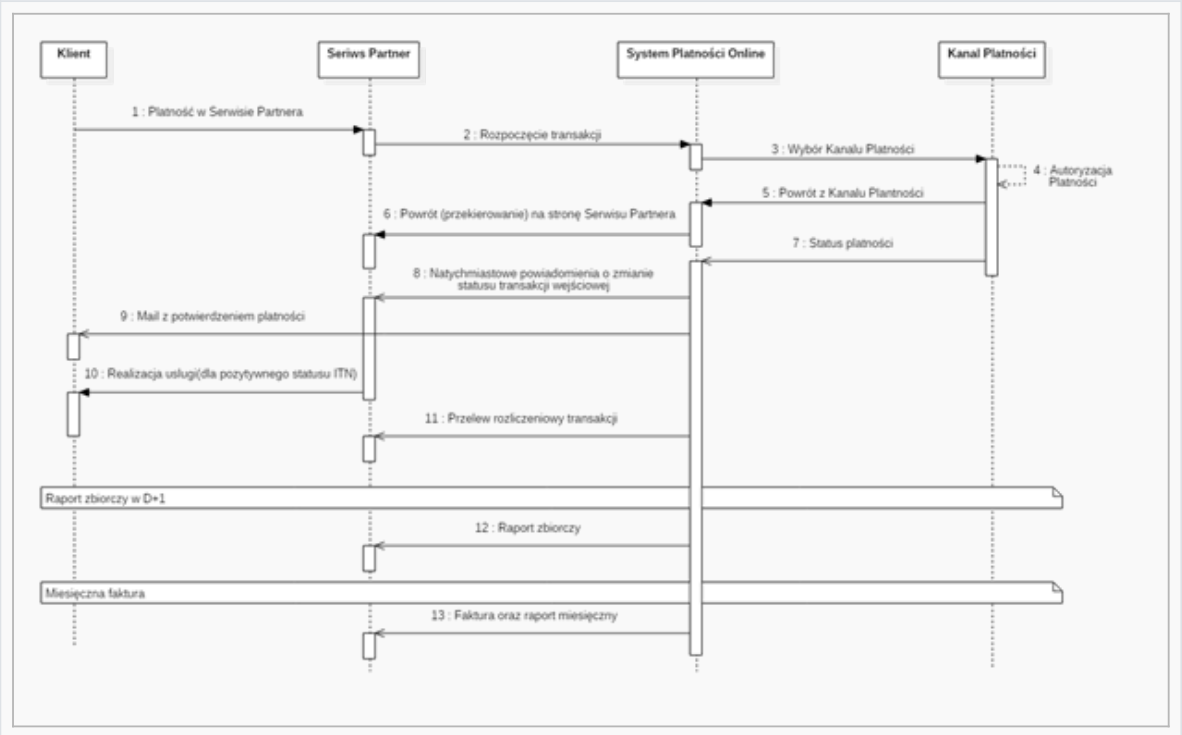
Collective settlement model for transactions (default model)

Summary settlements take place on the next working day (D+1).



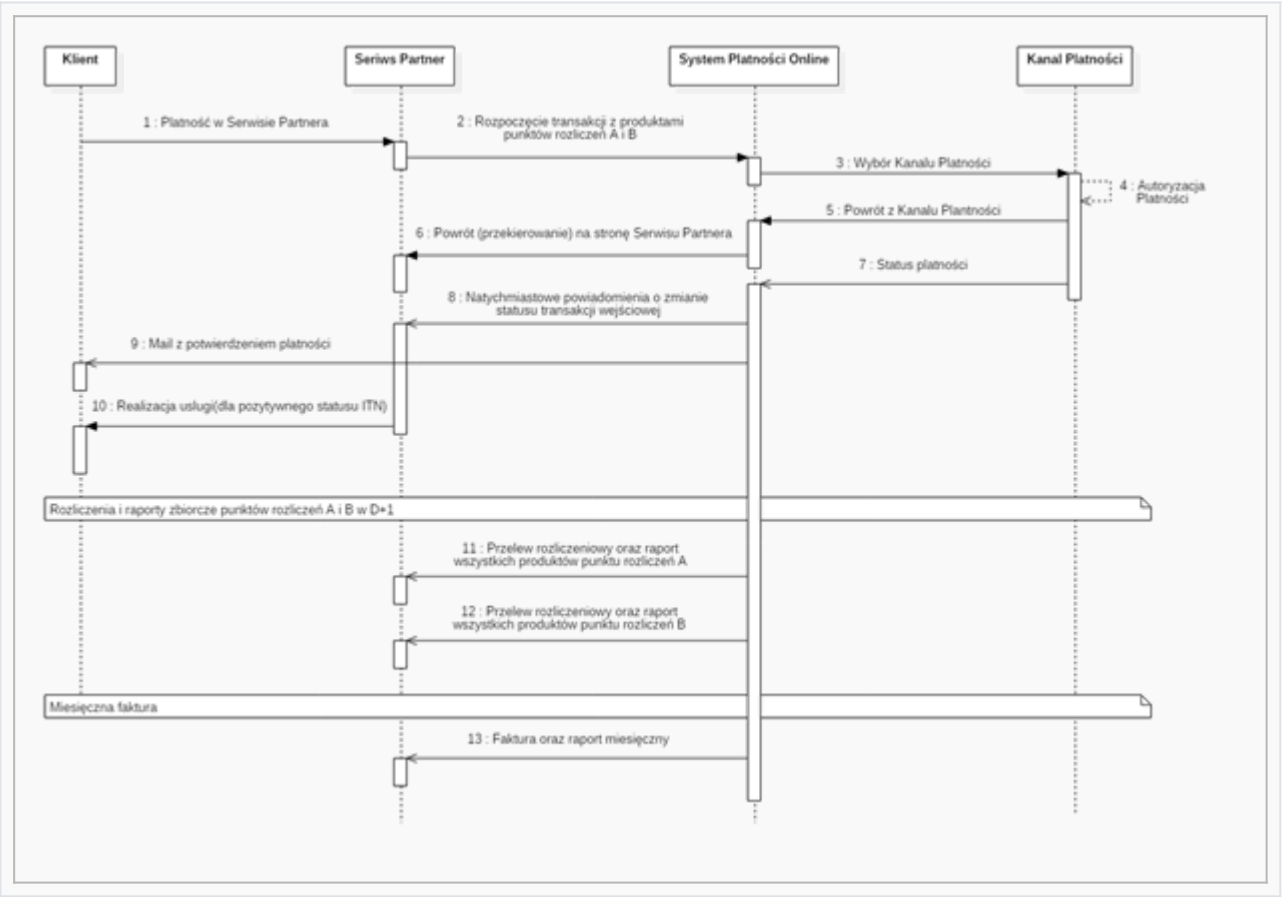
Settlement model for transactions after each payment

Settlement after each payment can be performed immediately after receipt of the payment from the Customer to the transaction data indicated in the parameters of the Payment Link (options: Recipient account, Title of settlement transfer, Name of recipient of settlement transfer).



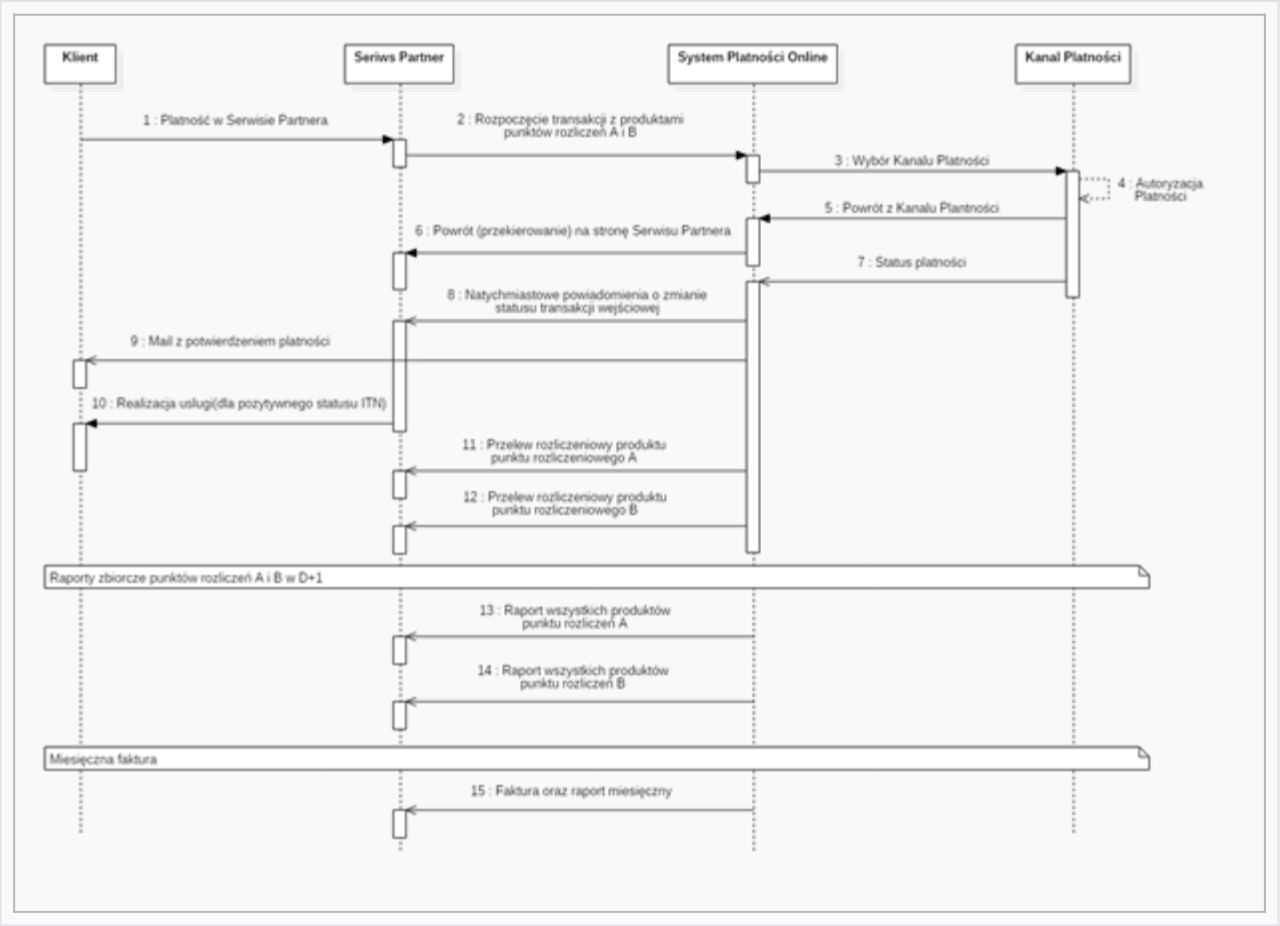
Collective product billing model

Summary settlements take place on the next working day (D+1).



Product billing model after each payment

Settlement after each payment can be made immediately after receipt of the payment from the Customer to the product data indicated in the parameters of the Payment Link (options: Recipient account, Title of settlement transfer, Name of recipient of settlement transfer).



On-demand settlement model

Settlements can be ordered by the Partner by calling the method: **transactionSettlement**.