A Simple PAYONE Integration in PHP

This document aims to help you in understanding the basics of our API. The code in our examples works, though should only be used as an example to make yourself familiar. We've provided the basic framework and some examples in our GitHub repository.

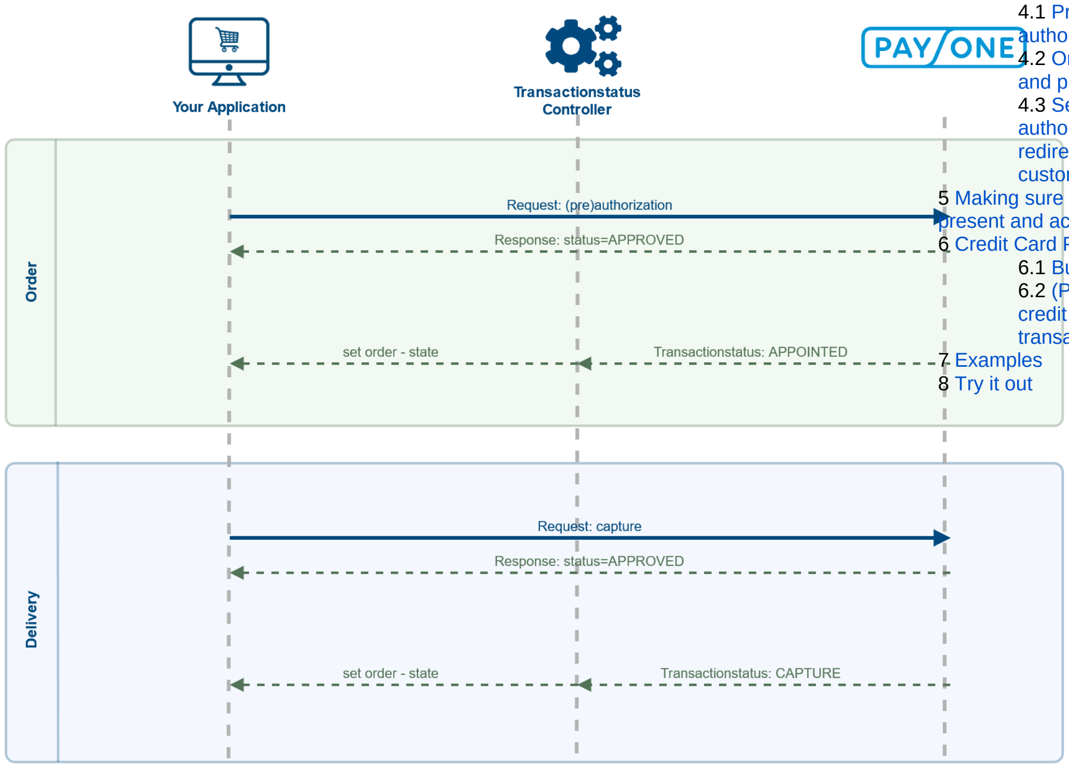
nload from Git

The code is not checked for security vulnerabilities and should not be used in a production environment without prior assessment.

Prerequisites

- Please use an up-to-date version of PHP (https://secure.php.net/)
- Composer (https://getcomposer.org/download/)
- A PAYONE account or test account (don't have one yet? Contact us!)

Communication Principles



Your Application
Transactionstatus Controller
PAYONE
Order
Request: (pre)authorization
Response: status=APPROVED
set order - state
Transactionstatus: APPOINTED
Delivery
Request: capture
Response: status=APPROVED
set order - state
Transactionstatus: CAPTURE

Communication from your server to our platform is performed by sending key-value-pairs per HTTP Post over a secure channel. In return, your application will receive a response string containing the result of your request. For sending the request to us, we recommend using a cURL wrapper that sends an array as key-value-pairs. The response are key value pairs delimited by EOL breaks, which can easily be parsed into an array. See `Payone.php` for Details.

For a detailed description of every parameter please refer to the Server API Description.

Creating a Payment

Default parameters

For every request to our platform, a set of default parameters is needed:

```
$defaults = array(
 "aid" => "your_account_id", //not actually a standard parameter, but
needed in most requests
 "mid" => "your_merchant_id",
 "portalid" => "your_portal_id",
 "key" => hash("md5", "your_secret_portal_key"), // the key has to be
hashed as md5
 "api_version" => "3.8",
 "mode" => "test", // can be "live" for actual transactions
 "encoding" => "UTF-8"
);
```

These parameters identify you as a merchant and ensure that only authorized API requests are processed by our platform. You can find these parameters in your merchant backend. See here for more info.

Personal Data

Additionally, some information about the customer can be transmitted, for instance received from your shop application or ERP system:

```
$personalData = array(
 "salutation" => "Mr.",
 "firstname" => "Henry",
 "lastname" => "Tudor", // mandatory
 "street" => "Royal Street 1",
 "zip" => "24118",
 "city" => "Kiel",
 "country" => "DE", // mandatory
 "email" => "henry.viii@tudor.gov.uk",
 "birthday" => "19700204",
 "language" => "de"
);
```

For details about the individual parameters and more parameters that are available but not listed here, please refer to the Server API Description.

Initiating a payment request

In this example, processing payment is a three step process. In the first step, the order is created (preauthorization). When we're ready to ship the goods, the receivable is booked (capture). In the third step we'll check if the funds have arrived.

1 **Merging the arrays**

```
$invoice = array(
 "clearingtype" => "rec", // classic invoice
 "reference" => "your_unique_reference", // a unique
reference, e.g. order number
 "amount" => "10000", // amount in smallest currency unit, i.
e. cents
 "currency" => "EUR",
 "request" => "preauthorization" // create order (but no
receivable yet)
);
```

Together with the recently gathered default parameters and personal data, the request can easily be merged and sent to the platform using the simple functions from the `Payone` class:

```
$request = array_merge($defaults, $personalData,
$invoice);$response = Payone::sendRequest($request);
```

You'll receive a status response informing you whether the request was successful and if not, about the cause of the error. However, if your API credentials were correct, you'll have received a `status=APPROVED` response. In any successful response you'll always receive a transaction ID parameter txid. This parameter is vital for any further communication about that transaction.

Your First Preauthorization

This set of parameters would tell PAYONE to create an order with the aforementioned personal data and the payment method "open invoice".

Request

Parameters in in alphabetical order

```
aid=12345
amount=10000
api_version=3.8
city=Kiel
clearingtype=rec
country=DE
currency=EUR
email=henry.viii@tudor.gov.uk
encoding=UTF-8
firstname=Henry
key=c4ca4238a0b923820dcc509a6f75849b
language=de
lastname=Tudor
mid=12345
mode=test
portalid=123456
reference=your_unique_reference_v1_final_latest
request=preauthorization
street=Royal Street 1
zip=24118
```

Response

```
status=APPROVED
txid=987654321
userid=123456789
```

2

Booking the amount, capturing the funds

Once the shipment is ready, the funds can be captured. Capturing indicates that the order should be finalized in terms of bookkeeping and, if applicable, that the money can be transferred (i.e. for credit card payments or direct debit, see below):

```
// again, the default values will be needed
$capture = array(
 "request" => "capture",
 "txid" => "your_txid",
 "sequencenumber" => "previous_sequencenumber_plus_one", //
get this from the last received transactionsstatus
 "amount" => "your_amount", // you can either capture the
full amount of the tx, or less
 "currency" => "EUR"
);
$request = array_merge($defaults, $capture);
$response = Payone::sendRequest($request);
if ($response["status"] == "APPROVED") {
 // amount is booked, ship the item!
}
```

The `sequencenumber` parameter ensures that all transaction status notifications have been processed before new requests can be sent to our API. It is incremented with each pair of request and transaction status notification. For the `preauthorization` request it is always implied as 0 and must not be sent.

Your First Capture

Request

Parameters in in alphabetical order

```
aid=12345
amount=10000
api_version=3.8
currency=EUR
encoding=UTF-8
key=c4ca4238a0b923820dcc509a6f75849b
mid=12345
mode=test
portalid=123456
reference=your_unique_reference_v1_final_latest
request=capture
txid=987654321
sequencenumber=1
```

Response

For payment methods that require the customer to send money themselves (like inovice), PAYONE will include clearing data in the response to the capture request. The customer has to know these bank account details to be able to wire the money to that account.

```
status=APPROVED
txid=987654321
settleaccount=no
clearing_bankaccountholder=PAYONE GmbH
clearing_bankaccount=0022520120
clearing_bankcode=21070020
clearing_bankcountry=DE
clearing_bankname=Testbank AG
clearing_bankbic=TESTDEXX210
clearing_bankiban=DE00123456781234567890
```

3

Waiting for payment confirmation

Depending on the payment method chosen, the confirmation of the customer's payment can be instant or take up to several days. Once we have new information about the transaction, we'll send a transaction status query to the URL configured in the PAYONE Merchant Interface. This status notification should be processed by your application in a separate controller, which should only be accessible from our platform (i.e. only from our IPv4 subnet). To prevent any tampering with the notification and minimize the danger of man-in-the-middle attacks, we highly recommend making this controller available through a secure connection only. Our platform expects a TSOK string as a confirmation that the notification has been received, if the TSOK hasn't been sent within 10 seconds, our platform will abandon the notification and try again at a later time.

```
// you'll need to include the $defaults array somehow, or at
least get the key from a secret configuration file
if ($_POST["key"] == hash("md5", $defaults["key"])) {
 // key is valid, this notification is for us
 echo "TSOK";
 /**
 * Most likely the notification will contain something like
$_POST["txaction"] == "paid"
 * but it can also include other parameters not used here.
Usually you'd log them and
 * update your order status accordingly. Refer to the Server
API Description, section 4.2
 * for details about the transaction status notification.
 */
 if ($_POST["txaction"] == "paid") {
 // update your transaction accordingly, e.g. by $_POST
["reference"]
 }
}
```

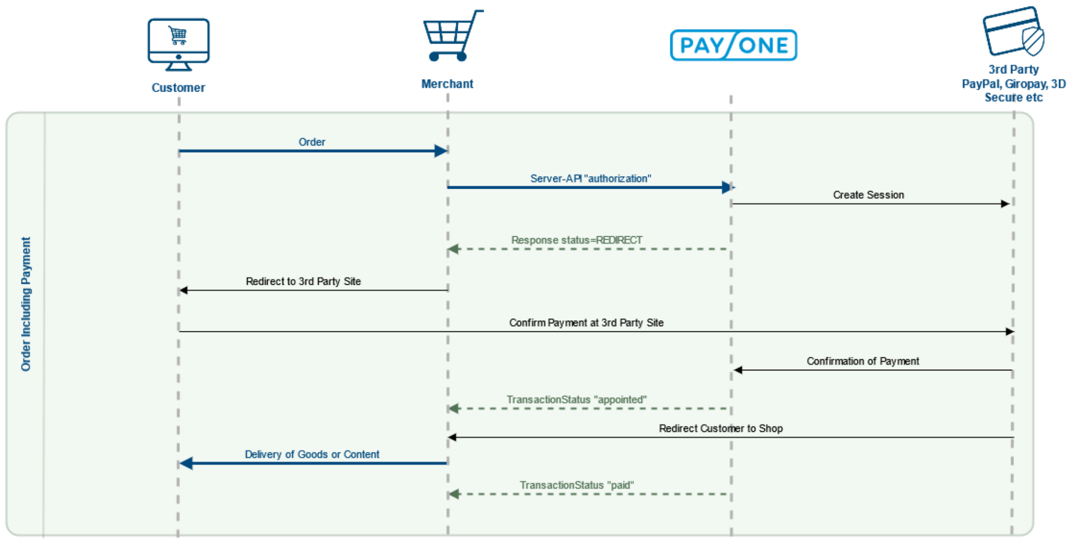
N.B. our platform does not follow HTTP 30x or any other form of redirects.

Since this is a simplified example, we left out the first transactionstatus message (txaction=appointed). For more info on transactionstatus messages, see here: Getting Started with TransactionStatus Notifications

Where to go from here

This intro is just the tip of the iceberg. For online bank transfer like Sofort.com, for instance, you'll need to redirect the customer to an URL specified in the response. For credit card processing, you'll need to setup a HTML container for input fields made available through our invisible iFrame integration and make sure your system never comes in contact with genuine credit card data. PAYONE will provide you with a pseudo card number that you can use to preauthorize and capture transactions just like in the examples above.

Redirect payment methods


Customer
Merchant
PAYONE
3rd Party
PayPal, Giropay, 3D Secure etc
Order Including Payment
Order
Server-API "authorization"
Create Session
Response status=REDIRECT
Redirect to 3rd Party Site
Confirm Payment at 3rd Party Site
Confirmation of Payment
TransactionStatus "appointed"
Redirect Customer to Shop
Delivery of Goods or Content
TransactionStatus "paid"

Sometimes, payment methods require information from the customer on 3rd party websites. Usually this is the case if the customer needs to enter transaction credentials, such as username/password or a TAN. Once you have acquainted yourself with the basic transaction principles outlined in the 1 Getting started tutorial, you're ready to tackle these payment methods. This is, again, a three step process: Preparing the (pre-)authorization, redirecting the customer, and verifying the transaction status.

Preparing the authorization

For the authorization, we'll rely on the `Payone::sendRequest()` function as before. So first, we need the `$defaults` and `$personalData` arrays:

```
$defaults = array(
 "aid" => "your_account_id",
 "mid" => "your_merchant_id",
 "portalid" => "your_portal_id",
 "key" => hash("md5", "your_secret_portal_key"), // the key has to be
hashed as md5
 "api_version" => "3.8",
 "mode" => "test", // can be "live" for actual transactions
 "encoding" => "UTF-8"
);
$personalData = array(
 "salutation" => "Mr.",
 "firstname" => "Henry",
 "lastname" => "Tudor",
 "street" => "Royal Street 1",
 "zip" => "24118",
 "city" => "Kiel",
 "country" => "DE",
 "email" => "henry.viii@tudor.gov.uk",
 "birthday" => "19700204",
 "language" => "de"
);
```

However, a couple of new parameters need to be introduced:

```
$onlineTransfer = array(
 "clearingtype" => "sb", // sb means online bank transfer
 "reference" => "your_unique_reference",
 "amount" => "10000", // amount in smallest currency unit, i.e. cents
 "currency" => "EUR",
 "request" => "authorization", // create account receivable and instantly
book the amount
 "onlinebanktransfertype" => "PNT", // this is the type for Sofort.com
 "bankcountry" => "DE", // we need to know the country of the customer's
bank, i.e. of the invoice address
 /**
 * As of July 2016, IBAN and BIC are optional for Sofort transactions as
long as we get a bankcountry
 */
 //"iban" => "DE85123456782599100003",
 //"bic" => "TESTTEST",
 "successurl" => "https://yourshop.com/payment/success?
reference=your_unique_reference",
 "errorurl" => "https://yourshop.com/payment/error?
reference=your_unique_reference",
 "backurl" => "https://yourshop.com/payment/back?
reference=your_unique_reference"
);
```

As a change, we'll be using `"request" => "authorization"` here, which means that not only the account receivable is created but also instantly booked. This is possible because Sofort.com provides instant notification about a successful payment and it saves you the hassle of implementing a separate `"request" => "capture"`. For rather asynchronous payment methods like prepayment or invoice this is not possible. Refer to the Server API Description for details. However, you are free to use a preauthorization/capture setup here as well if you see fit, for instance for bookkeeping reasons. The URL parameters determine to which page the customer gets redirected as soon as they a) complete the transaction (successurl), b) the transaction fails (errorurl) and the customer is advised to choose a different payment method or try again, or c) the customer hits "Back to shop" or any similar button on the 3rd party site to get back to the payment method selector (backurl). These controllers need to be implemented by you.

On authorization and preauthorization

The main difference between authorization and preauthorization is, that with authorization the funds are due instantly, while with preauthorization it is merely an "arrangement to pay". Funds in the preauthorization mode are due when a capture call is sent. With certain payment methods, the preauthorized funds are reserved (e.g. on a credit card), this is, however, not always the case. The time of the capture has implications on e.g. the dunning process as well.

To make a better example:

Imagine a standard fashion retailer. They would normally use a preauthorization workflow. The amount of the order is preauthorized when the customer finishes their checkout. When the goods are shipped (maybe partially, as not all items might be in stock), a capture API call is sent to indicate the funds are due now. There can be more than one capture call for a single preauthorization.

On the other hand, a merchant of digital content (e.g. e-books) would typically use an authorization flow to be able to make the goods available immediately.

In a typical integration, the merchant would be able to configure in their backend, which mode to use in which setting for which payment method.

Sending the authorization and redirecting the customer

Once you have prepared all the parameters needed for that transaction you're ready to authorize the transaction and then redirect the customer. We'll merge the arrays into one request, send the request to PAYONE, and see whether we get a status=REDIRECT response:

```
$request = array_merge($defaults, $personalData, $onlineTransfer);
$response = Payone::sendRequest($request);
if ($response["status"] == "REDIRECT") {
 header("Location: " . $response["redirecturl"]); // or other redirect
method
}
else {
 echo "Something went wrong. :(";
}
```

After the customer completed the transaction on the 3rd party site, they'll be redirected to the `successurl` so be sure to inform them about the success of their order. However, the order is not quite finished yet.

Making sure everything is present and accounted for

However, to prevent people with fraudulent intentions from directly calling the `successurl`, you can't rely on it alone. Additionally, PAYONE will sent a transaction status notification to the shop to indicate that the payment transaction has been completed. When verifying the transaction status take care to validate the key and probably the source IPv4 subnet as well. However, you won't receive a `txaction=paid` notification right away but rather a `txaction=appointed` to indicate that the transaction has been triggered and will be marked paid in a couple of minutes:

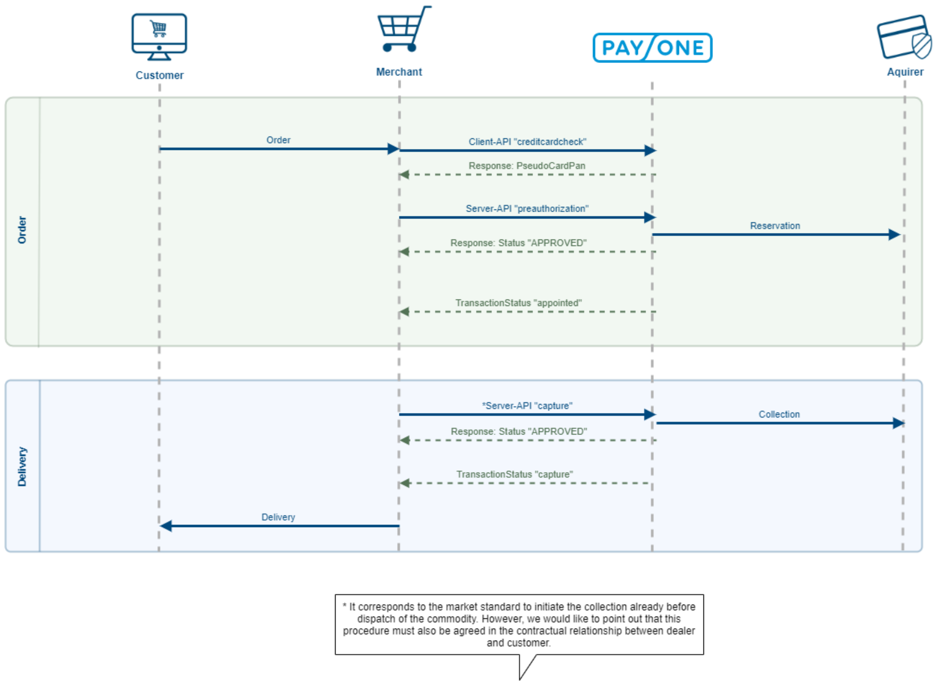
```
// you'll need to include the $defaults array somehow, or at least get the
key from a secret configuration file
if ($_POST["key"] == hash("md5", $defaults["key"])) {
 // key is valid, this notification is for us
 echo "TSOK";
 if ($_POST["txaction"] == "appointed") {
 // a freshly created transaction has been marked successfully initiated
 // update that transaction accordingly, e.g. by $_POST["reference"]
 }
 if ($_POST["txaction"] == "paid") {
 // update your transaction accordingly, e.g. by $_POST["reference"]
 }
}
```

Credit Card Payments

Online payments with credit cards are de facto mandatory for every online shop. However, the credit card issuers have high requirements concerning the security of credit card transactions. The Payment Card Industry Data Security Standard (PCI DSS) defines the prerequisites and certification steps. As certification is quite complex for merchants, PAYONE developed a solution that only requires the lowest level of PCI DSS compliance. Essentially, processing a credit card transaction is a three step process:

1. Create the form to capture the credit card details
2. Send them to PAYONE and receive a token in a callback
3. Perform (pre-)authorization using the token

The token is a so called pseudo card PAN, a number that resembles a credit card number, so that 3rd party systems can use it, but doesn't entail the PCI DSS requirements for storing card data. To avoid the software on the server to come in contact with credit card data, the Client API is used for communication between the buyer's browser and PAYONE.


Customer
Merchant
PAYONE
Aquirer
Order
Order
Client-API "creditcardcheck"
Response: PseudoCardPan
Server-API "preauthorization"
Reservation
Response: Status "APPROVED"
TransactionStatus "appointed"
Delivery
*Server-API "capture"
Collection
Response: Status "APPROVED"
TransactionStatus "capture"
Delivery

* It corresponds to the market standard to initiate the collection already before dispatch of the commodity. However, we would like to point out that this procedure must also be agreed in the contractual relationship between dealer and customer.

Build the form

PAYONE's "Hosted iFrame" solution gives the merchant the most flexibility in designing the checkout form while at the same time being PCI DSS compliant. The actual `<input>` elements are loaded from a secure PAYONE system so that the shop software itself isn't in the scope of PCI DSS certification. First, we need a sceleton of the form:

```
<script type="text/javascript" src="https://secure.pay1.de/client-api/js/v1
/payone_hosted_min.js"></script>
<form name="paymentform" action="" method="post">
    <fieldset>
        <input type="hidden" name="pseudocardpan" id="pseudocardpan">
        <input type="hidden" name="truncatedcardpan" id="truncatedcardpan">

        <!-- configure your cardtype-selection here -->
        <label for="cardtypeInput">Card type</label>
        <select id="cardtype">
            <option value="V">VISA</option>
            <option value="M">Mastercard</option>
            <option value="A">Amex</option>
        </select>

        <label for="cardpanInput">Cardpan:</label>
        <span class="inputIframe" id="cardpan"></span>

        <label for="cvcInput">CVC:</label>
        <span id="cardcvc2" class="inputIframe"></span>

        <label for="expireInput">Expire Date:</label>
        <span id="expireInput" class="inputIframe">
            <span id="cardexpiremonth"></span>
            <span id="cardexpireyear"></span>
        </span>

        <label for="firstname">Firstname:</label>
        <input id="firstname" type="text" name="firstname" value="">
        <label for="lastname">Lastname:</label>
        <input id="lastname" type="text" name="lastname" value="">

        <div id="errorOutput"></div>
        <input id="paymentsubmit" type="button" value="Submit" onclick="
check();">
    </fieldset>
</form>
<div id="paymentform"></div>
```

To configure the appearance of the various form elements delivered by PAYONE, a little JavaScript is neccessary:

```
<script>
    var request, config;

    config = {
        fields: {
            cardpan: {
                selector: "cardpan",                 // put name of your
div-container here
                type: "text",                        // text (default),
password, tel
                style: "font-size: 1em; border: 1px solid #000;"
            },
            cardcvc2: {
                selector: "cardcvc2",                // put name of your
div-container here
                type: "password",                    // select(default),
text, password, tel
                style: "font-size: 1em; border: 1px solid #000;",
                size: "4",
                maxlength: "4",
                                length: { "A": 4, "V": 3, "M": 3, "J": 0 }
                          },
            cardexpiremonth: {
                selector: "cardexpiremonth",         // put name of your
div-container here
                type: "select",                      // select(default),
text, password, tel
                size: "2",
                maxlength: "2",
                iframe: {
                    width: "50px"
                }
            },
            cardexpireyear: {
                selector: "cardexpireyear",          // put name of your
div-container here
                type: "select",                      // select(default),
text, password, tel
                iframe: {
                    width: "80px"
                }
            }
        },
        defaultStyle: {
            input: "font-size: 1em; border: 1px solid #000; width: 175px;",
            select: "font-size: 1em; border: 1px solid #000;",
            iframe: {
                height: "33px",
                width: "180px"
            }
        },
        error: "errorOutput",                        // area to display
error-messages (optional)
        language: Payone.ClientApi.Language.de       // Language to
display error-messages
                                                     // (default: Payone.
ClientApi.Language.en)
    };

    request = {
        request: 'creditcardcheck',                  // fixed value
        responsetype: 'JSON',                        // fixed value
        mode: 'live',                                // desired mode
        mid: '10001',                                // your MID
        aid: '10002',                                // your AID
        portalid: '2000002',                         // your PortalId
        encoding: 'UTF-8',                           // desired encoding
        storecarddata: 'yes',                        // fixed value
        // hash calculated over your request-parameter-values
(alphabetical request-order) plus PMI portal key
        hash: '5c4014cebeb361d9e186fd42c810b9b1'     // see https://docs.
payone.com/x/K4gS
    };
    var iframes = new Payone.ClientApi.HostedIFrames(config, request);
    iframes.setCardType("V");

    document.getElementById('cardtype').onchange = function () {
       iframes.setCardType(this.value);              // on change: set new
type of credit card to process
    };

    function check() {                               // Function called by
submitting PAY-button
        if (iframes.isComplete()) {
            iframes.creditCardCheck('checkCallback');// Perform
"CreditCardCheck" to create and get a
                                                     // PseudoCardPan;
then call your function "checkCallback"
        } else {
            console.debug("not complete");
        }
    }

    function checkCallback(response) {
        console.debug(response);
        if (response.status === "VALID") {
            document.getElementById("pseudocardpan").value = response.
```

```
pseudocardpan;
            document.getElementById("truncatedcardpan").value = response.
truncatedcardpan;
            document.paymentform.submit();
        }
    }

</script>
```

Never calculate the hash in JS!

Since JavaScript is run client-side, users could retrieve your portal key, which must be kept secret at all times! Since no personal details are part of the hash calculation, you can safely pre-calculate the hash and deliver it to the client.

After the check has gone through, the pseudo card PAN and truncated card PAN (e.g. 4111xxxxxxxx1111) will be stored in the hidden input fields and submitted to your application through http post.

(Pre-)authorizing credit card transactions

After you have obtained the pseudo card PAN through the Client API channel, the rest of the transaction can be continued through the Server API channel following the scheme above:

```
$creditCard = array(
 "clearingtype" => "cc", // cc for credit card
 "reference" => "your_unique_reference",
 "amount" => "10000", // amount in smallest currency unit, i.e. cents
 "currency" => "EUR",
 "request" => "preauthorization",
 "successurl" => "https://yourshop.com/payment/success?
reference=your_unique_reference", // URLs might be necessary since some
cards require REDIRECT for 3d secure
 "errorurl" => "https://yourshop.com/payment/error?
reference=your_unique_reference",
 "backurl" => "https://yourshop.com/payment/back?
reference=your_unique_reference",
 "pseudocardpan" => $pseudocardpan // pseudo card pan received from
previous checkout steps, no other card details required
);

$request = array_merge($defaults, $personalData, $creditCard);
$response = Payone::sendRequest($request);
if ($response["status"] == "REDIRECT") { // this happens when the card
needs a 3d secure verification
 header("Location: " . $response["redirecturl"]); // or other redirect
method
} elseif ($response["status"] == "APPROVED") { // no 3d secure
verification required, transaction went through
 echo "Thank you for your purchase.";
} else {
 echo "There has been an error processing your request.";
}
```

If the card requires 3D Secure verification, our platform will respond with a `status=redirect` response and give the URL the 3D Secure verification has to be carried out. After the customer successfully completed the 3D Secure form, we will send a transaction status notification `appointed` and then redirect them to the URL stated in the `successurl` parameter. The transaction status notification coming in before the redirect to the `successurl` asserts that the customer has indeed completed their 3D Secure form.

After the preauthorization is completed, you can continue with the transaction e.g. by capturing the full or a partial amount.

Examples

There are some handy examples in the examples folder. You're welcome to add more, if you feel like it!

Try it out

If you want to try out the examples provided here with your own account credentials, please install the required libraries through composer first:

```
git clone https://github.com/PAYONE-GmbH/simple-php-integration
cd simple-php-integration/
composer install
```

Then, adapt the individual files in `examples/` to your needs.