COPYandPAY Use Cases

The Use Case descriptions give you an overview of possible integration scenarios and how to handle them.

COPYandPAY version 3
These descriptions assume you're using COPYandPAY version 4 and above. For the use case descriptions for COPYandPAY version 3 and below click here

Please read the Integration guide before you continue here.

When you plan more extensive adaptions of the COPYandPAY form it's recommended to load the widget.js using the parameter style=plain or even style=none. The less styles you get predefined, the easier it is to adapt the form.

Add additional fields to the payment form

Use case

- A merchant wishes to add custom fields to the form and, optionally, retrieve this information after the payment.
- In this example: the shopper selects the number of installments that will be made, as controlled by the merchant's business logic.

Solution

- Utilize COPYandPAY's ability to execute Javascript upon loading to create the installment dropdown box. The dropdown box will be delivered with the COPYandPAY payment elements.

Let’s walk through the steps

Start with step 1:

Simply execute step number 1 'Generate a token for payment' as described in the Integration Guide.

Step 2:

Adding the additional field to the COPYandPAY forms:

- First, integrate the COPYandPAY Javascript and HTML form (where the COPYandPAY elements are attached) into your page as described in the Integration Guide.
- Find the name for the field you want to add or modify
  - A list of standard fields are available in the following list of Transaction Parameters.
  - If the field is not a standard field, it can be specified as a CRITERION; the parameter's name *must* start with CRITERION (ex: CRITERION.CUSTOM_number_of_installments)
  - In this example, the name of the field is 'CRITERION.CUSTOM_number_of_installments'.
- Add the var cnp_Options function Javascript as described in the following page COPYandPAY Options & API with the addition of the Merchant-defined field(s)

Example:

COPYandPAY: Add new field Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  onLoad: function(){
   // use jQuery to find all the credit card forms and insert
   // a 'Number of Installments' select row just before the submit button.
   var numberOfInstallmentsHtml = '<div class="customLabel">Number of
Installments</div><div class="customSelect"><select
name="CRITERION.CUSTOM_number_of_installments"><option value="1">1</option><option
value="3">3</option><option value="5">5</option></select></div>';
   $('.cnpForm.card').find('.submitInput').before(numberOfInstallmentsHtml);
  }
 }
</script>
```

Below is a sample showing the changes facilitated by the above script when using the "plain" skin for COPYandPAY.

Step 3:

After the payment is made, the payment will proceed to step 3 as described in the Integration Guide.

The key difference is that the JSON response will also include the custom parameter value pair as defined in cnp_Options

Advanced: Add interactive controls to your field

It is possible to add additional interaction with the page by binding events to the added control. Ex: an 'onclick' event could be bound to the control to open an overlay widget with a sophisticated installment calculator.

Should the number of installments influence the total amount, the calculator can do an ajax call to a script / server, triggering a new token for the new amount. Javascript would then refresh the page with the new token in place.

Change a label of the payment form

Use case

- A merchant would like to rename one of the input fields per business requirements. Ex: corporate identity requirements.
- In this use case the label 'Card Brand' will be 'Credit Card Brand'.

Solution

- A small script will be added to the merchant's checkout page. The script will change the name of the default input field label prior to displaying the fields to the user.

Let’s walk through the steps

Start with step 1:

Simply execute step number 1 'Generate a token for payment' as described in the Integration Guide.

Go to step 2:

Build the payment form for the shopper to make the payment - it is at this step that the change will be made to the aforementioned field.

- First, integrate the COPYandPAY Javascript and HTML form (where the COPYandPAY elements are attached) into your page as described in the Integration Guide.
- Add the var cnp_Options function Javascript as described in the following page COPYandPAY Options & API with the additional function to rename the card brand field.
    - A CSS / HTML debugger such as firebug will help identify the elements for customization; the Customization Guide goes into greater detail

Example:

COPYandPAY: Payment form label script Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  onLoad: function(){
   $('.brandLabel').html('Credit Card Brand');
  }
 }
</script>
```

The above script contains the following line:

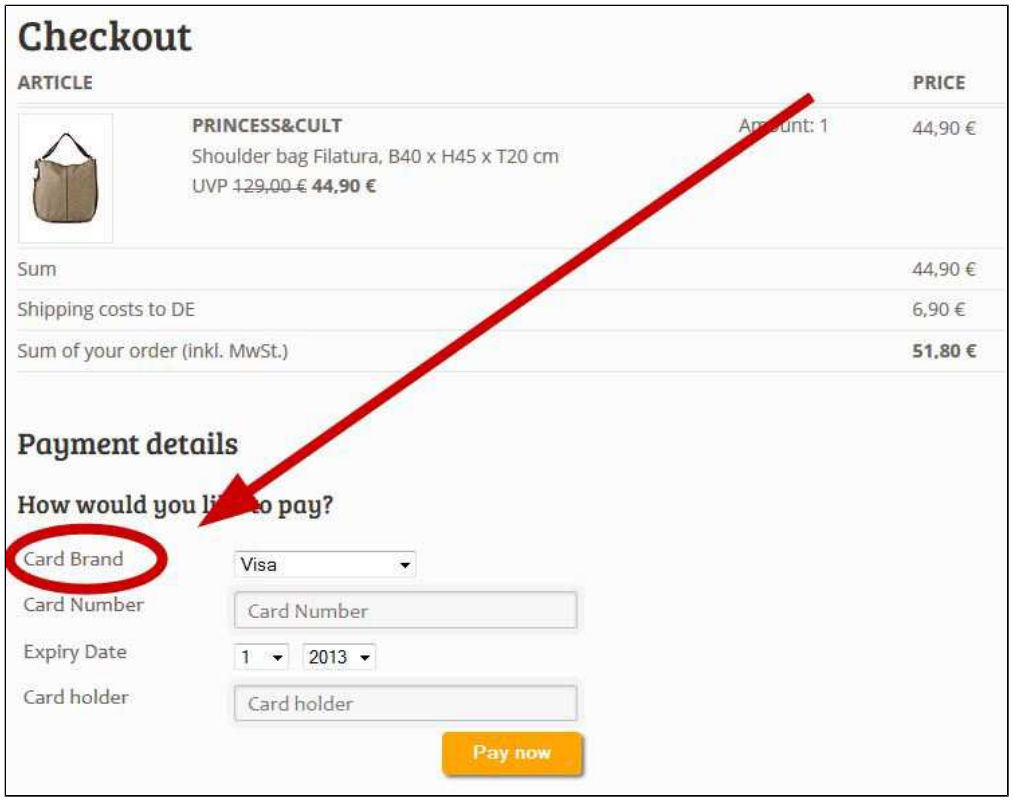
```
$('.brandLabel').html('Credit Card Brand');
```

In this code, .brandLabel defines the element as returned by COPYandPAY (with the text "Card Brand")

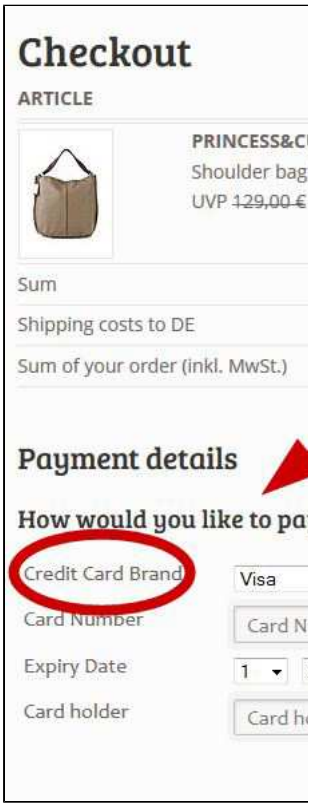
The portion in single quotes, 'Credit Card Brand', is the new label.

The above change is visualized below:

Before	After



Default input field labels from the COPYandPAY javascript library



Revised input field label

Step 3:

The checkout will finish as normal per the Integration Guide, no new parameters are returned via this customization.

Provide a custom UI when multiple payment methods are to be presented at once

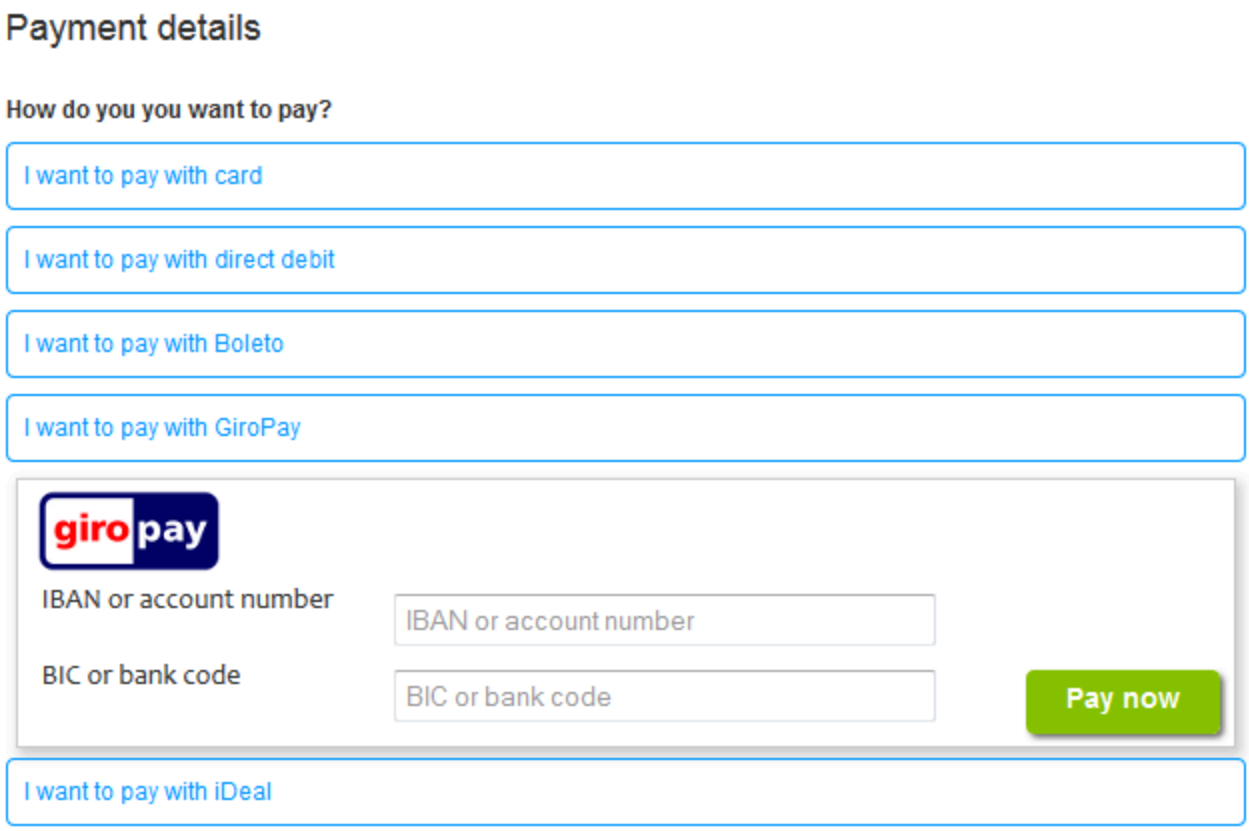
Use case

- A merchant wishes for a custom "window shade" UI to tidy up the checkout workflow while still presenting all available payment options.

Solution

- Add a script to the checkout page which adds clickable headlines for presenting the payment method(s).

⚠ This script also provides CSS which can be further customized or could override a merchant's CSS.



The code for this Use Case is broken into several parts:

1. The CSS or <style> to skin the flow as shown above
2. Importing jQuery
3. The method var methodMapping {} which maps text to particular payment methods. (useful for localization)
4. Adding Javascript to provide the clickable logic to make the window shade effect.

Example:

COPYandPAY Expand source

```
<style>
 h4.payHead {
  font-family: Candara, Calibri, Segoe, "Segoe UI", Optima, Arial, sans-serif;
  font-size: 14px;
  color: #09f;
  border: 2px solid #09f;
  border-radius: 4px;
  padding: 5px 8px;
  cursor: pointer;
 }
</style>
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var methodMapping = { //map is used for providing the h4 content
  cardPayment :                  "   I want to pay with card",
     ddPayment :                   "   I want to pay with direct debit",
     ppPayment_BOLETO :             "   I want to pay with Boleto",
     ppPayment_BARPAY :             "   I want to pay with Barpay",
     otPayment_IDEAL :             "   I want to pay with iDeal",
     otPayment_GIROPAY :         "   I want to pay with GiroPay",
     ivPayment_INVOICE :         "   I want to pay with Invoice",
     otPayment_SOFORTUEBERWEISUNG :   "   I want to pay with SOFORT Überweisung",
     vaPayment_PASTEANDPAY_V :           "   I want to pay with PASTEandPAY",
     vaPayment_VSTATION_V :              "   I want to pay with voucherstation",
     vaPayment_PAYPAL :                   "   I want to pay with PayPal",
     vaPayment_UKASH :      "   I want to pay with Ukash",
  vaPayment_QOOQO :      "   I want to pay with QOOQO",
     vaPayment_KLARNA_INVOICE :    "   I want to pay with Klarna Invoice",
     vaPayment_KLARNA_INSTALLMENTS :  "   I want to pay with Klarna Installments"
 }

 var cnp_Options = {
  onLoad: function(){
   $("div.cf").each( function (){
    var id = $(this).attr("id");
          $(this).before("<h4 class='payHead'>" + methodMapping[id.substring(0,
id.lastIndexOf("_"))] +"</h4>"); //add h4 headlines
          $(this).hide(); //hide the payment forms
   });
   $("h4").click( function (){ //register the click event
    $(this).next().slideToggle();
      });
  }
 }
</script>
```

The checkout will finish as normal per the Integration Guide, no new parameters are returned via this customization.

Use the payment form to store custom information for retrieval after payment

(aka pass through)

Use case

- A merchant wishes to add custom fields to the payment form to be passed through for processing post-payment

Solution

- Add the script to provide the fields to the var cnp_Options function Javascript as described in the following page COPYandPAY Options & API

Let's walk through the steps

Start with step 1:

Simply execute step number 1 'Generate a token for payment' as it is described in the Integration Guide.

Go to step 2:

Add the new, custom field:

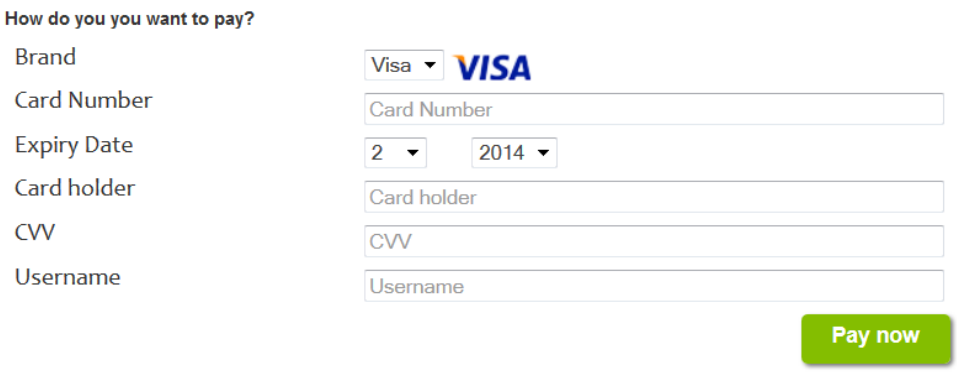
- First, integrate the COPYandPAY Javascript and HTML form (where the COPYandPAY elements are attached) into your page as described in the Integration Guide.
- Add the field
  - A list of standard fields are available in the following list of Transaction Parameters.
  - Specify the field to be added as a CRITERION (as in the use case above); the parameter's name *must* start with CRITERION (ex: CRITERION.Username)
  - In this example, the name of the field is 'CRITERION.Username'.
- Working with cnp_Options is further described in the COPYandPAY Options & API document

Example:

COPYandPAY: Add new field Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  onLoad: function(){
   var usernameHtml = '<div class="customLabel">Username</div><div
class="customInput"><input autocomplete="off" type="input" name="CRITERION.Username"
class="customInputField" placeholder="Username"></div>';
   $('.cnpForm.card').find('.submitInput').before(usernameHtml);
  }
 }
</script>
```

The above change returns the following result when using the "plain" COPYandPAY skin.



Step 3:

After the payment is done, call GetStatus as described in step 3 of the Integration Guide.

The name / value pairs of the custom field will be returned in the GetStatus-Response (in the example "BestPayerEver" was entered as a Username).

Example:

Custom Criterion in GetStatus Response › Expand source

```
criterions: [
 {
  name: "Username",
  value: "BestPayerEver"
 }
],
```

How to process prepayments (WIRE and CHECKS) with COPYandPAY

Use case

- A merchant wishes to offer offline / prepayment payment methods via a COPYandPAY integration.

Solution

- First, integrate the COPYandPAY Javascript and HTML form (where the COPYandPAY elements are attached) into your page as described in the Integration Guide.
- In the HTML form, add the payment brand PREPAYMENT. PREPAYMENT can be added with other payment types in the same form.

Example:

```
<form action="{url for redirecting the shopper after the payment}"id="{token}">VISA
MASTER PREPAYMENT</form>
```

PREPAYMENT requires the following configuration:

Step 1:

- For better usability, customize the Pre-Payment-Button with CSS and define how it shall be named (WIRE vs CHECKS).
  - Read the following for how to change the label name
- Depending on which channel shall be used for CHECKS and WIRE, you can use Channel Dispatching to define a specific channel for processing these payments.

Step 2:

- Create the Merchant Accounts
  - For multiple (different) countries, multiple Merchant Accounts should be created. Each Merchant Account will have specific information for WIRE/CHECK payments within that particular country.
  - Dispatching advice:
    - Create Merchant Accounts with additional information for each currency / country / bank
    - Depending on shopper location (country) and currency selected, the merchant can dispatch the payment to the appropriate bank (channel)
  - Each Clearing Institute defined for the above Merchant Accounts contains the parameters listed below. They have to be filled depending on the purpose of the Merchant Account (WIRE or CHECK), with bank account, or only the address - each bank /

country may have a slightly different set of required fields for this flow. Please contact your account manager if there are any questions regarding this step.

- Account data
- Name
- Country
- Id
- Gateway Address
- Kind
- Timezone
- UTC
- Bank Code
- BIC
- Street
- Zip
- City
- Aquirer Id
- Aquirer Public Key

Step 3:

- Display the bank data to the shopper - this step is executed after the shopper clicks "Pay Now" on the COPYandPAY form
  - After clicking "Pay Now", the merchant will receive banking details in the 'GetStatus request' (per step 3 in the Integration Guide and Code Samples).
  - Based on information received, the merchant displays the relevant bank data needed for the shopper to make the transfer / send the CHECK.

Complete it with Step 4:

- When the bank transfer payment (WIRE) is processed or the Check is cleared, you receive this information within the bank files in the BIP. Depending on the number of transactions there are 2 ways to handle this information:
  - For few transactions you can manually create the receipts (payment confirmation) for each Preauthorization
  - For more transactions you can convert the bank's response into a CSV file and upload it into the BIP

How to display an order summary page with COPYandPAY

Use case

- Per business or legal requirements, the following flow is required: shopping cart (1) -> Address data (2) -> payment form (3) -> Order overview (confirm transaction to be executed) (4) -> Execute Payment (5).
- This means the following changes as far as COPYandPAY is concerned: In page (3) the shopper fills the COPYandPAY payment form, but doesn't execute the payment. The actual execution is in step (5)

Solution

- Don't post the payment form, when the user is active on (3), with an *ExecutePayment* call, but call *GenerateToken* instead (with the same form data). When the payment is to be executed (5), post an empty form to *ExecutePayment*. This is facilitated by passing a special parameter to COPYandPAY.

Walkthrough of the Steps

Start with step 1:

- Within cnp_Options set *useSummaryPage* to *true* (see COPYandPAY Options & Api for utilizing this functionality). This changes the payment button's name and modifies the action attribute of the form from pointing to *ExecutePayment* to *GenerateToken*.
- All payment information is held on the payment server, however the payment is not immediately executed.
- Optionally the *onGenerateToken* function can be utilized. This is the callback inclusive of the result of the *GenerateToken* callback. The JSON payment response will be returned with the token (per step 3 in the Integration Guide and Code Samples).

Example:

Modifying the payment form Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  useSummaryPage: true,
  onGenerateToken: function(data){
   // data is the JSON response: { "transaction": { "token": "token"}  }
   console.log(data.transaction.token);
  }
 }
</script>
```

Go to step 2:

- On the last page (5) the merchant needs to make note of the "PAY"-Button. It is at this point that the merchant will POST an empty form, who's action points to the original address of the payment form. ex: this form now does the call to *ExecutePayment*.
- The response to this call will redirect the shopper to the merchant's result page. It is here that *GetStatus* is called just per the usual COPYandPAY payment workflow.

Listen to the 'form submit' event, ie when the payment form was submitted by the shopper

Use case

- A merchant wants to measure conversion via an analytics tool.
- It's hard to find out when the payment form is actually sent, because tracking of "submit button pressed" is not sufficient as there's a validation in between that potentially stops listening to the event by default.

Solution

- Add Javascript on the merchant's checkout flow
- The script will listen to the 'form submit event' and execute the code - if there are no validation errors - right before the form gets submitted.

Let’s walk through the steps

Step 1:

Simply execute step number 1 'Generate a token for payment' as it is described in the Integration Guide.

Step 2:

Build the payment form and let the shopper pay - it is at this step where the script will execute.

- First, integrate the COPYandPAY Javascript and HTML form (where the COPYandPAY elements are attached) into your page as described in the Integration Guide.
- Add the script within the onAfterSubmit functionality of the COPYandPAY Options & API

Example:

COPYandPAY: Payment submitted callback Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  onAfterSubmit: function(){
   // code executed right before the form gets send (at this point the validation
took already place)
  }
 }
</script>
```

Step 3:

After the payment is done, proceed to step 3 as it is described in the Integration Guide.

How to handle shoppers that do not return to the shop after payment?

Use case

- The shopper does not successfully return to the shop after payment (ex: closes browser). The merchant regardless wants notification of the transaction's final result.

Solution

- Before the session times out, and if there was no *GetStatus* call yet executed, COPYandPAY calls the shop's result page, just as the shopper would do if properly redirected.
- The shop then calls the GetStatus-URL which returns the JSON-formatted transaction results as response.

Let’s walk through the steps

Step 1:

You will receive the notification from COPYandPAY just like if a shopper had completed the process normally: a call to the return URL as configured in the payment form. The only difference is that this call doesn't come from the shopper's browser, but from our servers. As in the usual workflow, the merchant should call GetStatus with the token from the notification call (same token as used for the payment).

Step 2:

You will receive the response from the GetStatus, just as in the traditional workflow. It should be treated like a normal shopper call.

FAQ

Is this call a security threat?

- This is not a security threat, it is just a server to server call to the return url you've configured in the payment form. The notification call is empty, no other information or paramters are added. You actively have to request (pull) the status from our server. In case a third party would acquire the token and trigger the call, the potential attacker still wouldn’t be in the chain of the GetStatus-request/response.

How long does it take to receive this call?

- The call might take up to 20 minutes. The length of time for the call delivery depends on several factors, including the payment method that the shopper abandoned's time out tolerances.

What do I have to do to integrate this?

- The main difference is that the call comes from the COPYandPAY servers instead of being triggered directly by the shopper, thus this workflow needs to be IP whitelisted (if needed) and scripted against due to the potential for the call to be delayed.
- A possible consequence of this is that the shopper's session is lost. This is especially true when relying on browser cookies. The recommendation is to use a session-identifier as part of the return-url – thus this reference would be returned with the GetStatus call for further action (ex: notify user of abandoned payment).

Change size of 3D Secure IFRAME

Use case

The standard size of 3D Secure is:

- width: 400px
- height: 800px

Some issuers don't follow this rule and the size may differ.

Solution

- Set up the desired iFrame size in cnp_Options. See the COPYandPAY Options & API guide

Example:

COPYandPAY: Change iFrame Size › Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  iFrameSize: {width: 400, height: 580} // resize 3D Secure IFRAME
 }
</script>
```

Use COPYandPAY for Callcenters

Use case

- A merchant or PSP wishes to offer a call center product for customer service agents and / or mail order ordering
- The provided eTerminal doesn't offer enough flexibility for a particular use case
- The payment form also requires the submittal of additional fields

Solution

There is one main difference between a callcenter agent and a normal shopper: A callcenter agent is allowed to see more data, i.e. it's ok for a callcenter agent to see or even change data like the payment type, amount, transactionID, currency, usage. Sometimes it even makes sense for a callcenter agent to change the channel on the fly (e.g. depending on the status/country of the shopper, the selected products or payment provider).

⚠ None of these data should ever be visible to a normal shopper as this could pose a security risk!

Let’s walk through the steps

This functionality can be accomplished with the following steps from the use cases on this page:

- To offer callcenter agents an efficient user experience, it makes sense to add fields like e.g. the amount directly to the payment form .
- The workflow may require an agent to store additional information within the form.
- To flexibly switch between payment channels, have a look at the Use Cases for Channel Dispatching.

Intercept COPYandPAY submitting payment form and add some dynamic parts

Use case

The merchant wishes to add additional parameters to the COPYandPAY request before the form is subitted.

Solution

- The submit event of the form needs to be intercepted
- An ajax.post call is done the same origin
- After the ajax.post call was successfully executed, parameters can be added

Example:

COPYandPAY Expand source

```
<script src="https://code.jquery.com/jquery.js" ></script>
<script type="text/javascript">
 var cnp_Options = {
  onBeforeSubmitCard: function(event){
   var $form = $(this);
   if( $form.find(".inputError").length === 0 ) // no errors, submit
    //do your ajax call to the same origin
             cnp_jQuery.post("{url}").done( function(){ //executed if the ajax call
was sucessful
     $form.append("<input type='hidden' name='CRITERION.test' value='test' />")
//needs to a variable name we can read, see use cases regarding the use of
CRITERIONs
            $form.submit(); // here you submit the form
    }).fail( function(){//executed if the ajax call fails

    }).always (function(){//its always executed
    });

   }
   return false; // don't submit the form by default.
  }
 }
</script>
```

Recurring card payments - combine RG.DB or RG.PA

Use case

For recurring card payments, COPYandPAY offers a special workflow. Instead of sending a stand-alone Registration (RG) for storing the shoppers' data followed by a Debit (DB) or Preauthorization (PA), it is possible to combine the Registration with the payment in just one request. Simply send a CC.RG.DB or CC.RG.PA transaction. With this combined request, a registration is send to the system which saves all shopper data including credit card numbers and in addition, a Debit or Authorisation is triggered.

Solution

Use the combination worfklow and send CC.RG.DB/RG./PA in one request.

Option 1) Send a Registration with a following Debit - CC.RG.DB

- When generating a COPYandPAY token, set PAYMENT.TYPE=RG.DB

Option 2) Registration with a following Preauthorisation - CC.RG.PA

- When generating a COPYandPAY token, set PAYMENT.TYPE=RG.PA

Example:

Sample GenerateToken request

```
SECURITY.SENDER=8a82941743e2e2d80143e73bde660493
TRANSACTION.CHANNEL=8a82941743e2e2d80143e73bde660498
USER.LOGIN=8a82941743e2e2d80143e73bde660497
USER.PWD=demo
TRANSACTION.MODE=INTEGRATOR_TEST
PAYMENT.TYPE=RG.DB <!-- Sample combined call -->
PRESENTATION.AMOUNT=50
PRESENTATION.CURRENCY=EUR
ADDRESS.STREET=BAYERN
ADDRESS.ZIP=12345
ADDRESS.CITY=MUNCHEN
ADDRESS.COUNTRY=DE
CONTACT.EMAIL=shopper@generic.com
NAME.GIVEN=shopper
NAME.FAMILY=user
```

In both cases, the COPYandPAY payment widget renders the credit card form and the shopper enters credit card data. The response of the GetStatus call includes the status of the payment and a parameter "registration". The parameter registration is the uniqueID of the RG transaction. This id can be used for future recurring payments.

Payment response

```
{
"token":"9B7C1FAAF3730BDF4FCBAEFEA0CAF704.sbg-vm-fe01",
"transaction":{
"account":{
"bin":"420000",
"brand":"VISA",
"expiry":{
"month":"12",
"year":"2014"
},
"holder":"christian",
"last4Digits":"0000",
"registration":"ff80808146ec94b70146ed54ecc64dce"
```

```
},
"channel":"8a82941743e2e2d80143e73bde660498",
"criterions":[{
"name":"mode",
"value":"copyandpay"
}],
"customer":{
"address":{
"city":"MUNCHEN",
"country":"DE",
"street":"BAYERN",
"zip":"12345"
},
"contact":{
"ip":"82.135.34.186",
"ipCountry":"de"
}
},
"identification":{
"shortId":"0570.1169.2194",
"uniqueId":"ff80808146ec94b70146ed54f0544dee"
},
"mode":"INTEGRATOR_TEST",
"payment":{
"clearing":{
"amount":"50.0",
"currency":"EUR"
},
"code":"CC.DB"
},
"processing":{
"code":"CC.DB.90.00",
"connectorDetails":[],
"reason":{
"code":"00",
"message":"Successful Processing"
},
"result":"ACK",
"return":{
"code":"000.100.110",
"message":"Request successfully processed in 'Merchant in Integrator Test Mode'"
},
"timestamp":"2014-06-30 15:11:45"
},
```

```
"response":"SYNC"
}
}
```

Related pages:

- COPYandPAY Use Cases (Documentation)
- COPYandPAY Options & API (Documentation)
- COPYandPAY Performance (Documentation)
- COPYandPAY Use Cases for Channel Dispatching (Documentation)
- COPYandPAY New & Noteworthy (September 2014) (Documentation)
- Frontend Integrations (Documentation)
- Generate Token - Which parameters are allowed to send? (Documentation)
- Integration Guide and Code Samples (Documentation)
- COPYandPAY (Documentation)
- COPYandPAY Version 3 Use Cases (Documentation)
- Questions & Answers (PSP) (Documentation)
- Frontend Integrations (PSP) (Documentation)
- POST Parameters (Documentation)
- Profile Tracking Option (Documentation)
- COPYandPAY (PSP) (Documentation)

20 related results