COLLECTING SOLUTION

Quick start

Implementation Guide

Document version 1.8

Contents

1. HISTORY OF THE DOCUMENT

Version	Author	Date	Comment
1.8	Lyra Collect	7/30/2020	• Correction of field format for **vads_trans_date**. • Update of the *Setting up the Instant Payment Notification* chapter.
1.7	Lyra Collect	12/9/2019	• Update of the IPN setup procedure. • Addition of the **Implementing the IPN** chapter. • Correction of field format for **vads_product_label**. • Modification of field format for **vads_trans_id**.
1.6	Lyra Collect	6/17/2019	The hash algorithm is now available via Settings Shop, Keys tab.
1.5	Lyra Collect	1/23/2019	• Update of the **Identifying yourself when exchanging with the payment gateway** chapter. • “Certificate” replaced with “Key” in all menus
1.4	Lyra Collect	10/1/2018	Initial version

2. OBTAINING HELP

Looking for help? Check our FAQ on our website

https://lyra.com/doc/fr/collect/faq/sitemap.html

If you have any technical questions or need assistance, our tech support is available

from Monday to Friday from 9 a.m. to 6 p.m.

by phone at: **0811900475** Service fee 0.06 € / min + call charge

by e-mail : *support-ecommerce@lyra-collect.com*

and via your Expert Back Office, **Help** > **Contact support**

To facilitate the processing of your demands, you will be asked to communicate your shop ID (an 8-digit number) .

3. ESTABLISHING INTERACTION WITH THE PAYMENT GATEWAY

The merchant website and the payment gateway interact by exchanging data.

To create a payment, this data is sent in an HTML form via the buyer's browser.

At the end of the payment, the result is transmitted to the merchant website in two ways:

- Via the browser when the buyer clicks the button to return to the merchant website.
- automatically by means of a notification called Instant Notification URL (also called IPN), see chapter **Setting up the end of payment notification**.

To guarantee the security of the exchange, the data is signed with a key known only to the merchant and the payment gateway.

3.1. Setting up the payment page URL

The merchant website interacts with the payment gateway by redirecting the buyer to the following URL:

https://secure.lyra.com/vads-payment/

3.2. Identifying yourself when exchanging with the payment gateway

To be able to interact with the payment gateway, the merchant needs to have:

- **The shop ID:** allows to identify the merchant website during the exchange. Its value is transmitted in the **vads_site_id** field.
- **The key:** allows to compute the alphanumeric signature transmitted in the **signature** field.

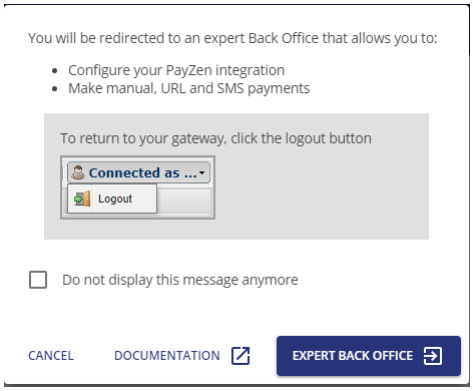
To retrieve these values:

1. Sign in to the **Lyra Collect Back Office**: *https://secure.lyra.com/portal/*
2. Enter your login.
3. Enter your password.
4. Click **Login**.

   In case of an entry error of the login and/or password, the error message "*Invalid username or password*" will appear.

   You can correct your entry or click on the link **Forgotten password or locked account**.
5. Click **Other actions**.

   The following window appears:



6. Click on **Expert Back Office** to access your Expert Back Office

7. Click **Settings > Shop**.

8. Select **Keys**.

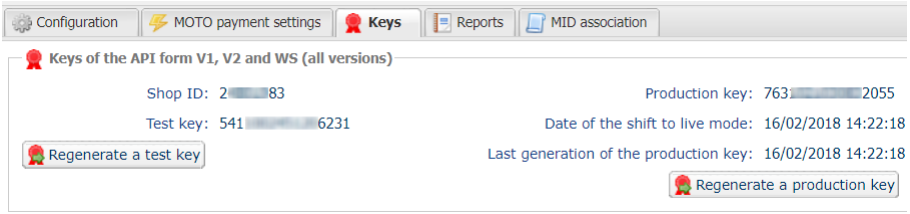


Figure 1: Keys tab

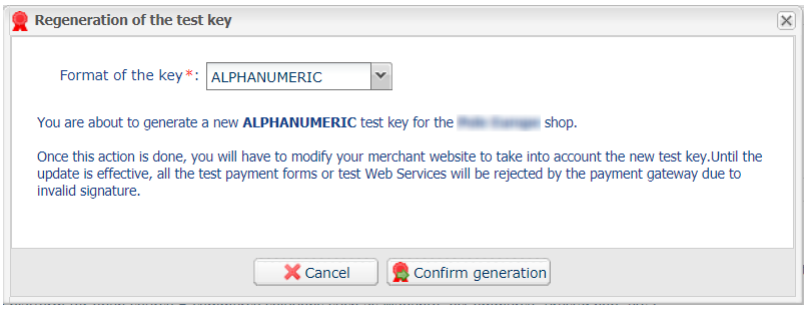
Two types of keys are available:

- The **test key** that allows to generate the form signature in test mode.
- The **production key** that allows to generate the form signature in production mode.

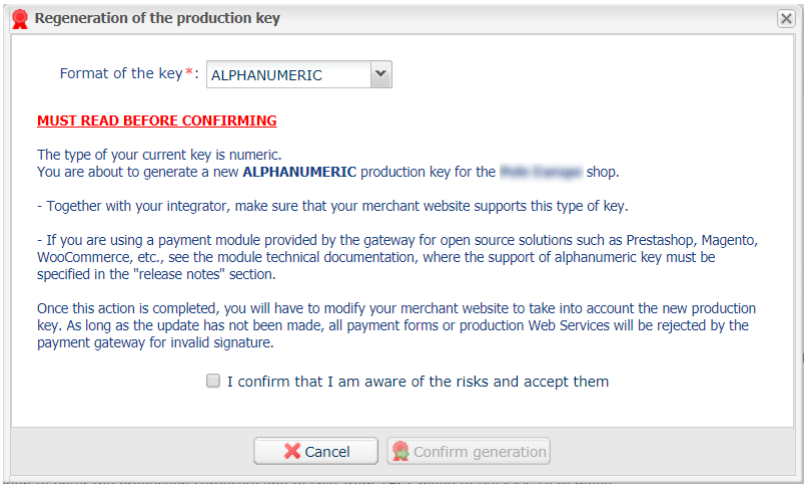
These keys can be numeric or alphanumeric.

For maximum security, it is recommended to use an alphanumeric key.

To change the format of your test key, click the **Regenerate a test key** button and select the format ("ALPHANUMERIC" or "NUMERIC").



To change the format of your production key, click the **Regenerate a production key** button and select the format ("ALPHANUMERIC" or "NUMERIC").


Regeneration of the production key
Format of the key*: ALPHANUMERIC
MUST READ BEFORE CONFIRMING
The type of your current key is numeric.
You are about to generate a new ALPHANUMERIC production key for the [redacted] shop.
- Together with your integrator, make sure that your merchant website supports this type of key.
- If you are using a payment module provided by the gateway for open source solutions such as Prestashop, Magento, WooCommerce, etc., see the module technical documentation, where the support of alphanumeric key must be specified in the "release notes" section.
Once this action is completed, you will have to modify your merchant website to take into account the new production key. As long as the update has not been made, all payment forms or production Web Services will be rejected by the payment gateway for invalid signature.
I confirm that I am aware of the risks and accept them
Cancel
Confirm generation

3.3. Managing interaction with the merchant website

Two types of URLs are used to manage the dialog with the merchant website:

- **Instant Payment Notification**, also called the IPN,
- **Return URL** to the merchant website.

Instant Payment Notification - IPN

The **Notification URL** is the URL of a specific page on the merchant website that is **automatically** called by the payment gateway when certain events take place.

By default, the rules are created to manage the events below:

- end of payment (accepted or rejected),
- payment abandoned or canceled,
- token creation or update,
- recurring payment creation,
- new installment date,
- authorization made in case of a deferred payment,
- update of a transaction status by the acquirer,
- operation made via the Expert Back Office (cancellation, refund, duplication, manual payment, etc.).

These rules must be enabled and configured according to the needs of the merchant.

With each call, the payment gateway transmits transaction details to the merchant website. It is called instant notification (or **IPN** as in Instant Payment Notification).

To guarantee the security of the exchange, the data is signed with a key known only to the merchant and the payment gateway.

Return URL to the merchant website

In the Expert Back Office, the merchant can configure the "default" return URLs via the menu **Settings** > **Shop** > **Configuration** tab:

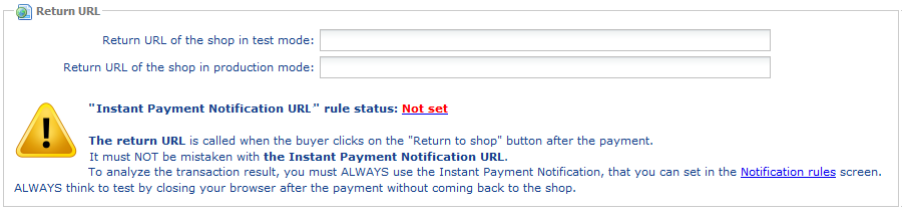


Figure 2: Setting up return URLs

The merchant can set up a different return URL for each mode.

By default, the buyer is redirected to the URL regardless of the payment result.

If no URL has been set up, the main URL of the shop will be used for redirection (**URL** parameter defined in the **Details** section of the shop).

The merchant will be able to override this setting in his/her payment form (see chapter **Setting up return URLs**).

Note:

The status of the "Instant Payment Notification at the End of Payment" (IPN) rule is displayed in this window. If the URL has not been set up, make sure to specify it (see chapter **Setting up notifications**).

3.4. Managing security

There are several ways to guarantee the security of online payments.

Ensuring interaction integrity

The integrity of exchanged information is preserved by the exchange of alphanumeric signatures between the payment platform and the merchant website.

The payment gateway and the merchant website interact via HTML forms.

A form contains a list of specific fields (see chapter **Generating a payment form**) used to generate a chain.

This chain is then converted to a smaller chain using a hash function (SHA-1, HMAC-SHA-256).

*The merchant will be able to choose the hash algorithm in their Expert Back Office (see chapter **Choosing the hash algorithm**).*

The resulting chain is referred to as the **digest** (*empreinte* in French) of the initial chain.

The digest must be transmitted in the **signature** field (see chapter **Computing the signature**).

Modeling security mechanisms:

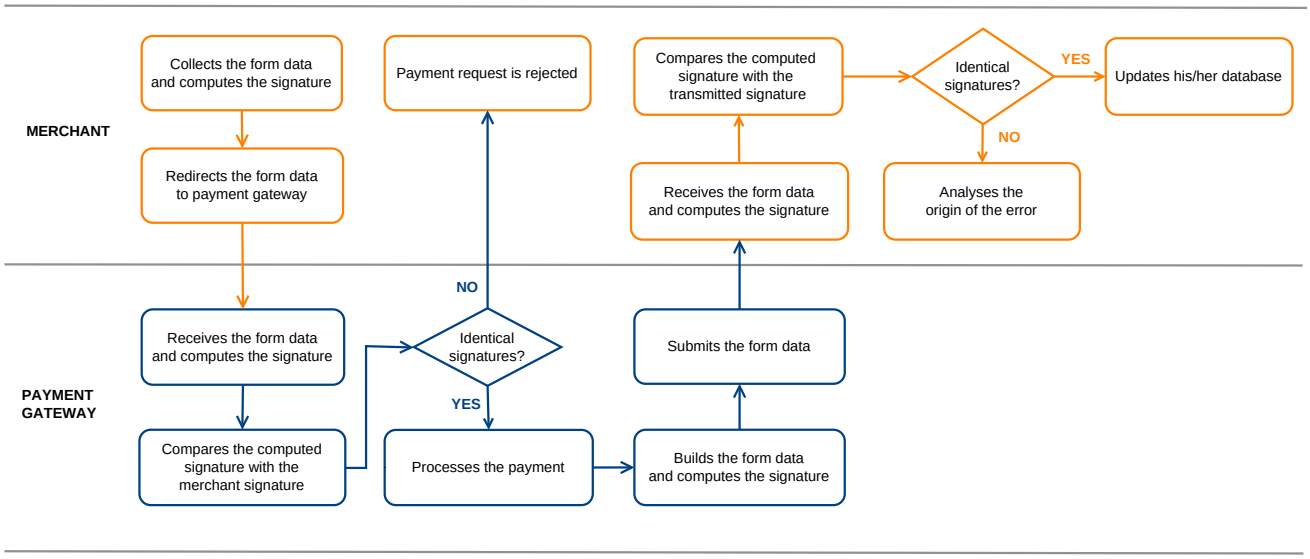


Figure 3: Diagram of a security mechanism

1. The merchant website builds the form data and computes the signature.
2. The merchant website submits the form to the gateway.
3. The gateway receives the form data and computes the signature.
4. The gateway compares the computed signature with the signature that was transmitted by the merchant website.
5. If the signatures are different, the payment request is rejected.

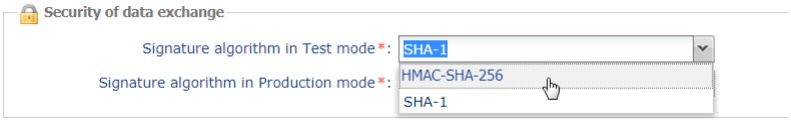
   If not, the gateway proceeds to payment.
6. The gateway builds the result data and computes the response signature.

7. Depending on the shop configuration (see chapter **Setting up notifications**), the payment gateway transmits the payment result to the merchant website.
8. The merchant website receives the data and computes the signature. It compares the computed signature with the signature that was transmitted by the payment gateway.
9. If the signatures are different, the merchant analyses the source of the error (computation error, attempted fraud, etc.).

   If not, the merchant proceeds to update their database (stock status, order status, etc.).

Selecting the hash algorithm

In the Expert Back Office, (**Settings** > **Shop** > **Keys**), the merchant can choose the hash function to use for generating signatures.



HMAC-SHA-256 signature algorithm is applied by default.

Important

You can select a different signature algorithm for TEST mode and for PRODUCTION mode.

However, be sure to use the same method to generate your payment forms and to analyze the data transmitted by the gateway during notifications.

In order to facilitate changing the algorithm, the SHA-1 or HMAC-SHA-256 signatures will be accepted without generating rejections due to signature error for 24h.

Storing the production key

For security reasons, the production key will be masked after the first real payment made with a real card.

It is strongly recommended to store the key in a safe place (encrypted file, database etc.).

In case of losing the key, the merchant will be able to regenerate a new one via their Expert Back Office.

Remember that the production key can be viewed in the Expert Back Office via **Settings** > **Shop** > **Keys** tab.

Managing sensitive data

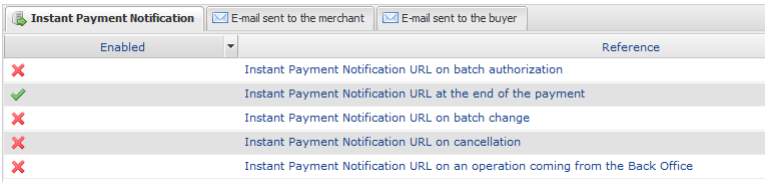
Online payment transactions are regulated by strict rules (PCI-DSS certification).

As a merchant, you have to make sure to never openly transcribe data that could resemble a credit card number. Your form will be rejected (code 999 - Sensitive data detected).

Special attention should be paid to order numbers containing between 13 and 16 numeric characters and beginning with 3, 4 or 5.

4. SETTING UP NOTIFICATIONS

To access the notification rule management, open the menu: **Settings** > **Notification rules**.



The rule configuration tab of “Instant Payment Notification URL call” type opens.

4.1. Setting up the Instant Payment Notification

This rule allows to notify the merchant website in the following cases:

- Payment accepted
- Payment refused
- Token creation or update
- Creation of a recurring payment

The **Payment accepted** event corresponds to the creation of a transaction in one of the (**vads_trans_status**) statuses below:

- **ACCEPTED**
- **AUTHORISED**
- **AUTHORISED_TO_VALIDATE**
- **CAPTURED**
- **INITIAL**
- **UNDER_VERIFICATION**
- **WAITING_AUTHORISATION**
- **WAITING_AUTHORISATION_TO_VALIDATE**
- **WAITING_FOR_PAYMENT**

This notification is required to communicate the result of a payment request.

It will inform the merchant website of the payment result even if your client has not clicked the “Return to the shop” button.

1. Right-click **Instant Payment Notification URL at the end of the payment**.
2. Select **Manage the rule**.
3. Enter the **E-mail address(es) to notify in case of failure** field in the **General settings** section.
   To specify several e-mail addresses, separate them with a semi-colon.
4. Check the box **Automatic retry in case of failure** if you wish to authorize the gateway to automatically resend the notification in case of a failure (can be done up to 4 times).
   For more information, please see chapter Automatic retry in case of failure on page 13.

5. In the **Instant Payment Notification URL of the API form V1, V2** section, specify the URL of your page in the fields **URL to notify in TEST mode** and **URL to notify in PRODUCTION mode**.

6. Save the changes.

4.2. Automatic retry in case of failure

Automatic retry does not apply to notifications manually triggered via the Expert Back Office.

The merchant can enable a mechanism that allows the payment gateway to automatically return notifications when the merchant website is temporarily unavailable, **up to 4 times**.

A notification will be considered as failed if the HTTP code returned by the merchant site is not on the following list: **200**,**201**, **202**, **203**, **204**, **205**, **206**, **301**, **302**,**303**, **307**, **308**.

Call attempts are scheduled at fixed intervals every 15 minutes (00, 15, 30, 45).

After each failed attempt, a notification e-mail is sent to the e-mail address specified in the configuration of the notification rule in question.

In this case, the subject of the e-mail contains the number corresponding to the notification retry attempt. It is presented as `attempt #` followed by the attempt number.

- Example of an e-mail subject following a first notification failure at the end of payment:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL
 [unsuccessful attempt #1]
```

- Example of an e-mail subject following a second failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL
 [unsuccessful attempt #2]
```

- Example of an e-mail subject following a third failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL
 [unsuccessful attempt #3]
```

- Example of an e-mail subject following the last failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL
 [unsuccessful attempt #last]
```

  To notify the merchant website of the last notification attempt, the e-mail subject will contain the mention `attempt #last`.

During the automatic retry, certain details are not stored in the database or are modified.

Examples of fields not available/not registered in the database:

Field name	Description
vads_page_action	Completed operation
vads_payment_config	Payment type (immediate or installment).
vads_action_mode	Acquisition mode for payment method data.

Examples of fields sent with different values:

Field name	New value
vads_url_check_src	Always set to **RETRY** in case of automatic retry.

Field name	New value
vads_trans_status	The transaction status may vary between the initial call and the automatic retry (cancellation by the merchant, transaction capture at the bank, etc.).
vads_hash	The value of this field is regenerated with each call.
signature	The signature value depends on the different statuses that may vary between the initial call and the automatic retry.

These e-mails contain:

- the encountered problem,
- parts of analysis depending on the error,
- its consequences,
- instructions for manually triggering the notification from the .

Note:

After the fourth attempt, it is still possible to retry the IPN URL **manually** via your .

Warning, during the automatic retry, any manual call to the IPN URL will affect the number of automatic attempts:

- a successful manual call will stop automatic retry,
- a failed manual call will have no impact on the current automatic retry.

4.3. Other cases of notification

Depending on the subscribed commercial options, the payment gateway will make a call to the notification URL in the following cases :

- abort or cancellation by the buyer on the payment page
- refund from the Expert Back Office
- cancellation of a transaction from the Expert Back Office
- validation of a transaction from the Expert Back Office
- modification of a transaction from the Expert Back Office
- etc.

For more information on configuring rules, see *Notification center* user guide.

5. SEND AN HTML PAYMENT FORM VIA POST

The merchant website redirects the buyer to the payment gateway using a POST form from HTML to HTTPS.

This form contains:

The following technical elements:

- The `<form>` and `</form>` tags that allow to create an HTML form.
- The `method="POST"` attribute that defines the method used for sending data.
- The `action="https://secure.lyra.com/vads-payment/"` attribute that defines where to send the form data.

Form data:

All the data in the form must be encoded in **UTF-8.**

Special characters (accents, punctuation marks, etc.) will then be correctly interpreted by the payment gateway. Otherwise, the signature will be computed incorrectly and the form will be rejected.

Please, consult the table below that indicates required formats.

Notation	Description
a	Alphabetic characters (from 'A' to 'Z' and from 'a' to 'z')
n	Numeric characters
s	Special characters
an	Alphanumeric characters
ans	Alphanumeric and special characters (except '<' and '>')
3	Fixed length of 3 characters
..12	Variable length up to 12 characters
json	JavaScript Object Notation. Object containing key/value pairs separated by commas. It starts with a left brace " `{` " and ends with a right brace " `}` ". Each key/value pair contains the name of the key between double-quotes followed by "`:`" followed by a value. The name of the key must be alphanumeric. The value can be: • a chain of characters (in this case it must be framed by double-quotes) • a number • an object • a table • a boolean • empty Example: `{"name1":45,"name2":"value2", "name3"=false}`
enum	Characterizes a field with a complete list of values. The list of possible values is given in the field definition.
Enum list	List of values separated by a " `;` ". The list of possible values is given in the field definition. Example: `vads_payment_cards=VISA;MASTERCARD`
map	List of key / value pairs separated by a "`;`". Each key/value pair contains the name of the key followed by "`=`", followed by a value. The value can be: • a chain of characters • a boolean • a json object

Notation	Description
	• an xml object The list of possible values for each key/value pair is provided in the field definition. Example: `vads_theme_config=SIMPLIFIED_DISPLAY=true;RESPONSIVE_MODEL=Model_1`

- Required fields:

Field name	Description	Format	Value
signature	Signature guaranteeing the integrity of the requests exchanged between the merchant website and the payment gateway.	ans	Ex : **ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0=**
vads_action_mode	Acquisition mode for payment method data	enum	**INTERACTIVE**
vads_amount	Payment amount in the smallest currency unit (cents for euro).	n..12	E.g.: 3000 for 30,00 EUR
vads_ctx_mode	Defines the mode of interaction with the payment gateway.	enum	**TEST** or **PRODUCTION**
vads_currency	Numeric currency code to be used for the payment, in compliance with the ISO 4217 standard (numeric code).	n3	E.g.: 978 for euro (EUR)
vads_page_action	Action to perform	enum	**PAYMENT**
vads_payment_config	Payment type	enum	**SINGLE** for an immediate payment **MULTI** for an installment payment
vads_site_id	Shop ID	n8	E.g.: 12345678
vads_trans_date	Date and time of the payment form in UTC format	n14	Respect the YYYYMMDDHHMMSS format E.g.: 20200101130025
vads_trans_id	Transaction number. **Warning: this field is not case sensitive.**	an6	E.g.: xrT15p
vads_version	Version of the exchange protocol with the payment gateway	enum	**V2**

Table 1: List of mandatories fields

- Recommended fields:
  - Order details

Field name	Description	Format	Value
vads_order_id	Order ID Can contain uppercase or lowercase characters, numbers or hyphens ([A-Z] [a-z], 0-9, _, -).	ans..64	E.g.: 2-XQ001
vads_order_info	Additional order info	an..255	E.g.: Door phone code 3125
vads_order_info2	Additional order info	an..255	E.g.: No elevator
vads_order_info3	Additional order info	an..255	E.g.: Express
vads_nb_products	Number of items in the cart	n..12	E.g.: 2
vads_product_ext_idN	Product barcode in the merchant's website. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..100	E.g.: vads_product_ext_id0 = "0123654789123654789" vads_product_ext_id1 = "0223654789123654789" vads_product_ext_id2 = "0323654789123654789"
vads_product_labelN	Item name. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	ans..255	E.g.: vads_product_label0 = "tee-shirt" vads_product_label1 = "Biscuit" vads_product_label2 = "sandwich"

Field name	Description	Format	Value
vads_product_amountN	Item amount. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_amount0 = "1200" vads_product_amount1 = "800" vads_product_amount2 = "950"
vads_product_typeN	Item type. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	enum	E.g.: vads_product_type0 = "CLOTHING_AND_ACCESSORIES" vads_product_type1 = "FOOD_AND_GROCERY" vads_product_type2 = "FOOD_AND_GROCERY"
vads_product_refN	Item reference. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..64	E.g.: vads_product_ref0 = "CAA-25-006" vads_product_ref1 = "FAG-B5-112" vads_product_ref2 = "FAG-S9-650"
vads_product_qtyN	Quantity of items. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_qty0 = "1" vads_product_qty1 = "2" vads_product_qty2 = "2"

Note:

*When the **vads_nb_products** field is populated, the **Shopping cart** tab becomes available in the transaction details in the Expert Back Office.*

*However, if the other fields that start with **vads_product_** are not populated, the tab will not include any information. For this reason, when populating the **vads_nb_products** field, it becomes mandatory to populate the other fields that start with **vads_product_**.*

- Buyer details

Field name	Description
vads_cust_email	Buyer's e-mail address.
vads_cust_id	Buyer reference on the merchant website.
vads_cust_title	Buyer's title.
vads_cust_status	Status (**PRIVATE**: for private clients / **COMPANY** for companies)
vads_cust_first_name	First name.
vads_cust_last_name	Last name
vads_cust_legal_name	Buyer's legal name.
vads_cust_cell_phone	Cell phone number
vads_cust_phone	Phone number.
vads_cust_address_number	Street number.
vads_cust_address	Postal address.
vads_cust_district	District.
vads_cust_zip	Zip code.
vads_cust_city	City.
vads_cust_state	State / Region.
vads_cust_country	Country code according to the ISO 3166 standard.

Table 2: Parameter list - Buyer details

- Shipping details

Field name	Description	Format	Value
vads_ship_to_city	City	an..128	E.g.: Bordeaux
vads_ship_to_country	Country code in compliance with the ISO 3166 standard (required for triggering one or more actions if the **Shipping**	a2	E.g.: FR

Field name	Description	Format	Value
	country control profile is enabled).		
vads_ship_to_district	District	ans..127	E.g.: La Bastide
vads_ship_to_first_name	First name	ans..63	E.g.: Albert
vads_ship_to_last_name	Name	ans..63	E.g.: Durant
vads_ship_to_legal_name	Legal name	an..100	E.g.: D. & Cie
vads_ship_to_phone_num	Phone number	ans..32	E.g.: 0460030288
vads_ship_to_state	State / Region	ans..127	E.g.: Nouvelle aquitaine
vads_ship_to_status	Allows to specify the type of the shipping address.	enum	**PRIVATE**: for shipping to a private individual **COMPANY**: for shipping to a company
vads_ship_to_street_number	Street number	ans..64	E.g.: 2
vads_ship_to_street	Postal address	ans..255	E.g.: Rue Sainte Catherine
vads_ship_to_street2	Second line of the address	ans..255	
vads_ship_to_zip	Zip code	an..64	E.g.: 33000

- Optional fields :

  You can use additional optional parameters.

  See the chapter **Data Dictionary** of the Hosted Payment Page Implementation guide available on our web site to see the list of the available fields.

The **Pay** button that will allow to send data:

```
<input type="submit" name="pay" value="Pay"/>
```

6. COMPUTING THE SIGNATURE

To be able to compute the signature, you must have:

- all the fields that start with **vads_**
- the signature algorithm chosen in the shop configuration
- the **key**

The value of the key is available in your Expert Back Office via **Settings** > **Shop** > **Keys** tab.

The signature algorithm is defined in your Expert Back Office via **Settings** > **Shop** > **Configuration** tab.

For maximum security, it is recommended to use HMAC-SHA-256 algorithm and an alphanumeric key.

The use of SHA-1 algorithm is deprecated but maintained for compliance reasons.

To compute the signature:

1. Sort the fields that start with **vads_** alphabetically.
2. Make sure that all the fields are encoded in UTF-8.
3. Concatenate the values of these fields separating them with the "**+**" character.
4. Concatenate the result with the test or production key separating them with a "**+**".
5. According to the signature algorithm defined in your shop configuration:
   a. if your shop is configured to use "SHA-1", apply the **SHA-1** hash function to the chain obtained during the previous step. **Deprecated.**
   b. if your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
      - the SHA-256 hash function,
      - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
      - the result of the previous step as the message to authenticate.
6. Save the result of the previous step in the **signature** field.

Example of parameters sent to the payment gateway:

```
<form method="POST" action="https://secure.lyra.com/vads-payment/">
<input type="hidden" name="vads_action_mode" value="INTERACTIVE" />
<input type="hidden" name="vads_amount" value="5124" />
<input type="hidden" name="vads_ctx_mode" value="TEST" />
<input type="hidden" name="vads_currency" value="978" />
<input type="hidden" name="vads_page_action" value="PAYMENT" />
<input type="hidden" name="vads_payment_config" value="SINGLE" />
<input type="hidden" name="vads_site_id" value="12345678" />
<input type="hidden" name="vads_trans_date" value="20170129130025" />
<input type="hidden" name="vads_trans_id" value="123456" />
<input type="hidden" name="vads_version" value="V2" />
<input type="hidden" name="signature" value="ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0="/>

<input type="submit" name="pay" value="Pay"/>
</form>
```

This sample form is analyzed as follows:

1. The fields whose names start with **vads_** are sorted **alphabetically**:
   - vads_action_mode
   - vads_amount
   - vads_ctx_mode
   - vads_currency
   - vads_page_action
   - vads_payment_config
   - vads_site_id
   - vads_trans_date
   - vads_trans_id
   - vads_version
2. The values of these fields are concatenated using the "**+**" character:

   ```
   INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2
   ```

3. The value of the test key is added at the end of the chain and separated with the "**+**" character. In this example, the test key is **1122334455667788**

   ```
   INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2+1122334455667788
   ```

4. If you use the SHA-1 algorithm, apply it to the obtained chain.

   The result that must be transmitted in the signature field is: **59c96b34c74b9375c332b0b6a32e6deeec87de2b**

5. If your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
   - the SHA-256 hash function,
   - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
   - the result of the previous step as the message to authenticate.

   The result that must be transmitted in the signature field is:

   ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0=

6.1. Example of implementation with JAVA

Definition of the utility class SHA that will include the elements required to process the HMAC-SHA-256 algorithm

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.TreeMap;

public class VadsSignatureExample {
 /**
  * Build signature (HMAC SHA-256 version) from provided parameters and secret key.
  * Parameters are provided as a TreeMap (with sorted keys).
  */
 public static String buildSignature(TreeMap<String, String> formParameters, String
  secretKey) throws NoSuchAlgorithmException, InvalidKeyException, UnsupportedEncodingException
 {
   // Build message from parameters
   String message = String.join("+", formParameters.values());
   message += "+" + secretKey;
   // Sign
   return hmacSha256Base64(message, secretKey);
  }
    /**
    * Actual signing operation.
    */
 public static String hmacSha256Base64(String message, String secretKey) throws
  NoSuchAlgorithmException, InvalidKeyException, UnsupportedEncodingException {
   // Prepare hmac sha256 cipher algorithm with provided secretKey
   Mac hmacSha256;
   try {
    hmacSha256 = Mac.getInstance("HmacSHA256");
   } catch (NoSuchAlgorithmException nsae) {
    hmacSha256 = Mac.getInstance("HMAC-SHA-256");
   }
   SecretKeySpec secretKeySpec = new SecretKeySpec(secretKey.getBytes("UTF-8"), "HmacSHA256");
   hmacSha256.init(secretKeySpec);
   // Build and return signature
   return Base64.getEncoder().encodeToString(hmacSha256.doFinal(message.getBytes("UTF-8")));
  }
}
```

Definition of the utility class SHA that will include the elements required to process the SHA-1 algorithm

```
import java.security.MessageDigest;
import java.security.SecureRandom;

public class Sha {
    static public final String SEPARATOR = "+" ;
    public static String encode(String src) {
        try {
            MessageDigest md;
            md = MessageDigest.getInstance( "SHA-1" );
            byte bytes[] = src.getBytes( "UTF-8" );
            md.update(bytes, 0, bytes. length );
            byte[] sha1hash = md.digest();
            return convertToHex(sha1hash);
        }
        catch(Exception e){
            throw new RuntimeException(e);
        }
    }
    private static String convertToHex(byte[] sha1hash) {
        StringBuilder builder = new StringBuilder();
            for (int i = 0; i < sha1hash. length ; i++) {
               byte c = sha1hash[i];
               addHex(builder, (c >> 4) & 0xf);
               addHex(builder, c & 0xf);
        }
        return builder.toString();
    }
    private static void addHex(StringBuilder builder, int c) {
        if (c < 10)
            builder.append((char) (c + '0' ));
        else
            builder.append((char) (c + 'a' - 10));
    }
```

```
}
```


Function that computes the signature:

```
public ActionForward performCheck(ActionMapping actionMapping, Basivoirorm form,
    HttpServletRequest request, HttpServletResponse response){
        SortedSet<String> vadsFields = new TreeSet<String>();
        Enumeration<String> paramNames = request.getParameterNames();

        // retrieve and sort the fields starting with vads_* alphabetically
        while (paramNames.hasMoreElements()) {
            String paramName = paramNames.nextElement();
            if (paramName.startsWith( "vads_" )) {
                vadsFields.add(paramName);
            }
        }
// Compute the signature
String sep = Sha.SEPARATOR;
StringBuilder sb = new StringBuilder();
for (String vadsParamName : vadsFields) {
    String vadsParamValue = request.getParameter(vadsParamName);
    if (vadsParamValue != null) {
        sb.append(vadsParamValue);
    }
    sb.append(sep);
}
sb.append( shaKey );
String c_sign = Sha.encode(sb.toString());
return c_sign;}
```


6.2. Example of implementation with PHP

Example of a signature computation using the HMAC-SHA-256 algorithm:

```
function getSignature ($params,$key)
{
    /**
     *Function that computes the signature.
     * $params : table containing the fields to send in the payment form.
     * $key : TEST or PRODUCTION key
     */
    //Initialization of the variable that will contain the string to encrypt
    $contenu_signature = "";

    //sorting fields alphabetically
    ksort($params);
    foreach($params as $name=>$value){

        //Recovery of vads_ fields
        if (substr($nom,0,5)=='vads_'){

            //Concatenation with "+"
            $contenu_signature .= $value."+";
         }
    }
    //Adding the key at the end
    $contenu_signature .= $key;

    //Encoding base64 encoded chain with SHA-256 algorithm
    $signature = base64_encode(hash_hmac('sha256',$contenu_signature, $key, true));
    return $signature;
 }
```

Example of a signature computation using the SHA-1 algorithm:

```
function getSignature($params, $key)
{
    /**
     * Function that computes the signature.
     * $params : table containing the fields to send in the payment form.
     * $key : TEST or  PRODUCTION key
     */
    //Initialization of the variable that will contain the string to encrypt
    $contenu_signature = "" ;

    // Sorting fields alphabetically
    ksort($params);
        foreach ($params as $name =>$value)
    {
    // Recovery of vads_ fields
        if (substr($nom,0,5)=='vads_') {
            // Concatenation with "+"
            $contenu_signature .= $value."+";
        }
    }
    // Adding the key at the end
    $contenu_signature .= $key;

    // Applying SHA-1 algorithm
    $signature = sha1($contenu_signature);
    return $signature ;
}
```


7. IMPLEMENTING THE IPN

The script must include at least the following steps:

- Retrieve the field list sent with the POST response
- Compute the signature taking into account the received data
- Compare the computed signature with the received signature
- Analyze the nature of the notification
- Retrieve the payment result

The script may check the order status (or any information of your choice) to see if it has not been already updated.

Once these steps are completed, the script can update the database (new order status, stock update, registration of payment information, etc.).

In order to facilitate support and diagnosis by the merchant in the event of a notification error, we recommend to write messages that will allow you to know at which stage of processing the error occurred.

The gateway reads and stores the first 256 bytes of the HTTP response.

You can write messages throughout the processing. Here are some examples of messages that you can use:

Message	Use case
Data received	Message to display when retrieving data. Allows to confirm that the notification has been received by the merchant website.
POST is empty	Message to display when retrieving data. Allows to bring out a possible redirection that would have caused the parameters posted by the payment gateway to be lost.
An error occurred while computing the signature.	Message to be displayed when the verification of the response signature has failed.
Order successfully updated.	Message to be displayed at the end of the file once your processing has been successfully completed.
An error occurred while updating the order.	Message to be displayed at the end of the file if an error occurred during your processing.

7.1. Preparing your environment

The notifications of type are the most important as they represent the only reliable way for the merchant website to obtain the payment result.

It is therefore necessary to make sure the notifications function properly.

Here are some guidelines:

- In order for the dialog between the payment gateway and your merchant website to work, you must make sure, together with your technical teams, that the **194.50.38.0/24** IP address range is authorized on the various devices within your system (firewalls, apache server, proxy server, etc.).

  Notifications are sent from an IP address in the 194.50.38.0/24 range **in Test and Production modes**.

- Using redirection leads to losing data presented in POST.

  This is the case if there is a configuration on your devices or on the side of your host that redirects the URLs of “http://**www.**example.com” type to “http://example.com” or “**http**://example.com” to “**https**://example.com”.

- HTML must not be visible on the page. Access to images or CSS slows down the exchange between the payment gateway and the merchant website.

- Avoid integrating time-consuming tasks, such as PDF invoice generation or sending e-mails in your script.

  The processing time has a direct influence on the time it takes to display the payment summary page.

  The longer the processing of the notification, the greater the delay for displaying the page. After 35 seconds, the payment gateway considers that the call has failed (timeout).

- If your page is only accessible in https, test your URL on the Qualys SSL Labs website (*https://www.ssllabs.com/ssltest/* and change your configuration if necessary in order to obtain the A score.

  Your SSL certificate must be signed by a certification authority known and recognized on the market.

- Make sure that you use the latest version of the TLS protocol in order to maintain a high level of security.

7.2. Retrieving data returned in the response

The data returned in the response depends on the parameters sent in the payment request, the payment type, the settings of your shop and the notification format.

The data is always sent by the payment gateway using the **POST** method.

The first step consists in retrieving the contents received via the POST method.

Examples:

- In PHP, data is stored in the super global variable **$_POST**,
- In ASP.NET (C#), you must use the **Form** property of the **HttpRequest** class.
- In Java, you must use the **getParameter** method of the **HttpServletRequest** interface.

The response consists of a field list. Each field contains a response value. The field list can be updated.

The script will have to create a loop to retrieve all the transmitted fields.

It is recommended to test the presence of the **vads_hash** field, which is only present during a notification.

```
if (empty ($_POST)){
 echo 'POST is empty';

}else{
 echo 'Data Received ';
 if (isset($_POST['vads_hash'])){

  echo 'Form API notification detected';
  //Signature computation
  //Signature verification
  //Order Update
 }
}
```


7.3. Computing the IPN signature

The signature is computed by following the same logic as for creating the payment request.

IMPORTANT

The data submitted by the payment gateway is encoded in UTF-8. Any alteration of received data will result in signature computation error.

You must compute the signature with the fields received in the notification and not the ones that you transmitted in the payment request.

1. Take all the fields whose name starts with **vads_**.
2. Sort these fields alphabetically.
3. Concatenate the values of these fields separating them with the "**+**" character.
4. Concatenate the result with the test or production key separating them with a "**+**".
5. According to the signature algorithm defined in your shop configuration:
   a. if your shop is configured to use "SHA-1", apply the **SHA-1** hash function to the chain obtained during the previous step. **Deprecated.**
   b. if your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
      - the SHA-256 hash function,
      - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
      - the result of the previous step as the message to authenticate.

Examples in PHP:

```
function getSignature ($params,$key)
{
    /**
     *Function that computes the signature.
     * $params: table containing the fields received in the IPN.
     * $key : TEST or PRODUCTION key
     */
    //Initialization of the variable that will contain the string to encrypt
    $signature_contents = "";

    //Sorting fields alphabetically
    ksort($params);
    foreach($params as $name=>$value){

        //Recovery of vads_ fields
        if (substr($name,0,5)=='vads_'){

            //Concatenation with "+"
            $signature_contents .= $value."+";
         }
    }
    //Adding the key at the end
    $signature_contents .= $key;

    //Encoding base64 encoded chain with HMAC-SHA-256 algorithm
    $sign = base64_encode(hash_hmac('sha256',$signature_contents, $key, true));
    return $sign;
 }
```


7.4. Comparing signatures

To ensure the integrity of the response, you must compare the signature contained in the IPN with the value computed in the previous step.

IMPORTANT
You should not compare the signature of the IPN with the signature that you transmitted in your payment request.

If the signatures match

- you may consider the response as safe and proceed with the analysis.
- otherwise, the script will have to raise an exception and notify the merchant about the anomaly.

Example in PHP:

```
if ($_POST['signature'] == $sign){

    //Processing data

}else{
    throw new Exception('An error occurred while computing the signature');
}
```

The signatures may not match in case of:

- an implementation error (error in your calculation, problem with UTF-8 encoding, etc.),
- an error in the value of the key or in the **vads_ctx_mode** field (frequent issue when shifting to production mode),
- a data corruption attempt.

7.5. Analyzing the nature of the notification

During a notification, the **vads_url_check_src** field allows to differentiate the notifications based on their triggering event:

- creation of a transaction.
- new notification sent by the merchant via the Expert Back Office.

It specifies the applied notification rule:

Value	Applied rule
PAY	The PAY value is sent in the following cases: • immediate payment (or first installment payment of a recurring payment) • payment deferred for less than 7 days only if the merchant has configured the **Instant Payment Notification URL at the end of payment** rule. • payment abandoned or canceled by the buyer only if the merchant has configured the rule **Instant Payment Notification URL on cancellation**
BO	Execution of the notification via the Expert Back Office (right-click a transaction > **Send the Instant Payment Notification**).
BATCH	The BATCH value is sent in case of an update of a transaction status after its synchronization on the acquirer side. This is the case of payments with redirection to the acquirer. Only if the merchant has configured the rule **Instant Payment Notification URL on batch change**.
BATCH_AUTO	The BATCH_AUTO value is sent in the following cases: • payment deferred for more than 7 days • installments of a recurring payment (except the first one) only if the merchant has configured the rule **Instant Payment Notification URL on batch authorization** The notification is sent with the authorization request for payments with "Waiting for authorization" status.
REC	The REC value is sent only for recurring payments if the merchant has configured the **Instant Payment Notification URL when creating recurring payments** rule.
MERCH_BO	The MERCH_BO value is sent: • during operation made via the Expert Back Office (refund, cancellation, modification, validation, duplication, creation and/or update of token), only if the merchant has configured the following notification rule: **Instant Payment Notification URL on an operation coming from the Back Office**
RETRY	Automatic retry of the IPN.

Table 3: Values associated with the vads_url_check_src field

After checking its value, the script can process differently depending on the nature of the notification.

For example:

If **vads_url_check_src** is set to **PAY** or **BATCH_AUTO**, the script will update the order status, etc.

If **vads_url_check_src** is set to **REC**, the script will retrieve the recurring payment reference and will increment the number of the expired installment payments in case the payment has been accepted, etc.

7.6. Processing the response data

Here is an example of analysis to guide you through processing the response data.

1. Identify the mode (TEST or PRODUCTION) that was used for creating the transaction by analyzing the value of the **vads_ctx_mode** field.

2. Identify the order by retrieving the value of the **vads_order_id** field if you have transmitted it to the payment gateway.

   Make sure that the order status has not been updated yet.

3. Retrieve the payment result transmitted in the **vads_trans_status** field.

   Its value allows you to define the order status.

Value	Description
ABANDONED	**Abandoned** Payment abandoned by the buyer The transaction has not been created, and **therefore cannot be viewed in the Expert Back Office.**
ACCEPTED	**Accepted**. Status of a VERIFICATION type transaction for which the authorization request or information request has been successfully completed. This status can not evolve. Transactions with the "**ACCEPTED**" status will never be captured.
AUTHORISED	**Waiting for capture** The transaction has been accepted and will be automatically captured at the bank on the expected date.
AUTHORISED_TO_VALIDATE	**To be validated** The transaction, created with manual validation, is authorized. The Merchant must manually validate the transaction in order for it to be captured. The transaction can be validated as long as the expiration date of the authorization request has not passed. If the authorization validity period has passed, the payment status changes to **EXPIRED**. The **Expired** status is final.
CANCELLED	**Canceled** The transaction has been canceled by the Merchant.
CAPTURED	**Captured** The transaction has been captured by the bank.
CAPTURE_FAILED	Capture failed Contact the technical support.
EXPIRED	**Expired** This status appears in the lifecycle of a payment with deferred capture. The expiry date of the authorization request has passed and the Merchant has not validated the transaction. The account of the cardholder will, therefore, not be debited.
REFUSED	**Refused** Transaction is declined
SUSPENDED	**Suspended** The capture of the transaction is temporarily blocked by the acquirer (AMEX GLOBAL or SECURE TRADING). Once the transaction has been correctly captured, its status changes to **CAPTURED**.
UNDER_VERIFICATION	**Control in progress** Waiting for the response from the acquirer. This status is temporary.

Value	Description
	For CB or PPRO transactions, this status indicates that a refund has been requested. Verification is being made in order to validate the refund. A notification will be sent to the merchant website to inform the Merchant of the status change. Requires the activation of the Instant Payment Notification URL on batch change notification rule.
WAITING_AUTHORISATION	**Waiting for authorization** The capture delay in the bank exceeds the authorization validity period.
WAITING_AUTHORISATION_TO_VALIDATE	**To be validated and authorized** The capture delay in the bank exceeds the authorization validity period. An authorization of 1 EUR (or information request about the CB network if the acquirer supports it) has been accepted. The Merchant must manually validate the transaction for the authorization request and the capture to occur.

4. Retrieve the payment reference transmitted in the **vads_trans_id** field.

5. Analyze the **vads_payment_config** field to determine whether it is an **immediate payment** or an **installment payment**.

Field name	Value for an immediate payment	Value for a payment in installments
vads_payment_config	SINGLE	MULTI (the exact syntax is MULTI:first=X;count=Y;period=Z)

Table 4: vads_payment_config field analysis

For a payment in installments, identify the installment number by retrieving the value of the **vads_sequence_number** field.

Value	Description
1	First installment
2	Second installment
3	Third installment
n	Installment number

Table 5: vads_sequence_number field analysis

6. Analyze the **vads_sequence_number** field to know the number of attempts that have been made to make the payment.

vads_payment_config = SINGLE:

vads_url_check_src	vads_sequence_number	Description
PAY	1	Payment made in 1 attempt
	2	Payment made in 2 attempts
	3	Payment made in 3 attempts
BATCH_AUTO	1	Deferred payment made in 1 attempt
	2	Deferred payment made in 2 attempts
	3	Deferred payment made in 3 attempts

Note

Installment payments are not compatible with the feature of additional attempts in case of a rejected payment.

7. Retrieve the value of the **vads_trans_date** field to identify the payment date.

8. Analyze the **vads_payment_option_code** field to determine whether it is an installment payment:

Value	Description
1	Payment in 1 installment
2	Payment in 2 installments
3	Payment in 3 installments
n	Payment in n installments

Table 6: vads_payment_option_code field analysis

9. Retrieve the value of the **vads_capture_delay** field to identify the number of days before the capture in the bank.

   It will allow you to identify whether the payment is an immediate or a deferred payment.

10. Retrieve the used amount and currency. To do this, retrieve the values of the following fields:

Field name	Description
vads_amount	Payment amount in the smallest currency unit.
vads_currency	Code of the currency used for the payment.
vads_change_rate	Exchange rate used to calculate the effective payment amount (see vads_effective_amount).
vads_effective_amount	Payment amount in the currency used for the capture in the bank.
vads_effective_currency	Currency used for the capture in the bank.

Table 7: Analysis of the payment amount and currency

11. Retrieve the value of the **vads_auth_result** field to identify the result of the authorization request.

    The complete list of returned codes can be viewed in the data dictionary.

    Here is a list of frequently returned codes that can help you understand the reason of the rejection:

Value	Description
03	**Invalid acceptor** This code is sent by the card issuer. It refers to a configuration problem on authorization servers. (e.g. closed contract, incorrect MCC declared, etc.). **To find out the specific reason of the rejection, the buyer must contact his or her bank.**
05	**Do not honor** This code is sent by the card issuer. This code is used in the following cases: • Invalid expiry date • Invalid CVV • Exceeded credit limit • Insufficient funds (etc.) **To find out the specific reason of the rejection, the buyer must contact his or her bank.**
51	**Insufficient balance or exceeded credit limit** This code is sent by the card issuer. This code appears if the funds on the buyer's account are insufficient for making the purchase. **To find out the specific reason of the rejection, the buyer must contact his or her bank.**
56	**Card absent from the file** This code is sent by the card issuer. The entered card number is incorrect or the card number + expiration date combination does not exist.
57	**Transaction not allowed for this cardholder** This code is sent by the card issuer. This code is used in the following cases: • The buyer attempts to make an online payment with a cash withdrawal card, • The authorized payment limit is exceeded. **To find out the specific reason of the rejection, the buyer must contact his or her bank.**
59	**Suspected fraud** This code is sent by the card issuer. This code appears when an incorrect CVV code or expiration date has been entered several times. **To find out the specific reason of the rejection, the buyer must contact his or her bank.**

Value	Description
60	**The acceptor of the card must contact the acquirer** This code is sent by the card issuer. It refers to a configuration problem on authorization servers. It is used when the merchant ID does not correspond to the used sales channel (e.g.: an e-commerce transaction with a distant sales MID with manual entry of contract data). **Contact the customer service to resolve the problem.**

Table 8: Values associated with the vads_auth_result field

12. Retrieve the 3D Secure authentication result. To do this:

    a. Retrieve the value of the **vads_threeds_enrolled** field to identify the status of the card enrollment.

Value	Description
Empty	Incomplete 3DS authentication process (3DS disabled in the request, the merchant is not enrolled or the payment method is not eligible for 3DS).
Y	Authentication available, cardholder enrolled.
N	Cardholder not enrolled.
U	Impossible to identify the cardholder or authentication is not available for the card (e.g. commercial or prepaid cards).

Table 9: Values of the vads_threeds_enrolled field

    b. Retrieve the result of 3D Secure authentication by retrieving the value of the **vads_threeds_status** field.

Value	Description
Empty	Incomplete 3DS authentication (3DS disabled in the request, the cardholder is not enrolled or the payment method is not eligible for 3DS).
Y	Cardholder authentication success.
N	Cardholder authentication error.
U	Authentication impossible.
A	Authentication attempted but not completed.

Table 10: Values of the vads_threeds_status field

13. Retrieve the result of fraud checks by identifying the value of the **vads_risk_control** field. This field is sent only if the merchant has:

- subscribed to the "**Risk management**" service
- enabled at least one verification process in the Expert Back Office (**Settings** > **Risk management** menu).

It is populated with the list of values separated by ";" with the following syntax: **vads_risk_control = control1=result1;control2=result2**

the possible values for **control** are:

Value	Description
CARD_FRAUD	Verifies whether the cardholder's card number is on the card greylist.
SUSPECT_COUNTRY	Checks whether the issuing country of the buyer's card is on the list of forbidden countries.
IP_FRAUD	Verifies whether the cardholder's IP address is on the IP greylist.
CREDIT_LIMIT	Checks the purchase frequency and amounts for the same card number, or the maximum amount of an order.
BIN_FRAUD	Checks whether the BIN code of the card is on the BIN code greylist.
ECB	Checks whether the buyer's card is of "e-carte bleue" type.
COMMERCIAL_CARD	Checks whether the buyer's card is a commercial card.
SYSTEMATIC_AUTO	Checks whether the buyer's card is a MAESTRO or VISA ELECTRON card.

Value	Description
INCONSISTENT_COUNTRIES	Checks whether the country of the IP address, the country of the payment card and the buyer's country of residence match.
NON_WARRANTY_PAYMENT	Liability shift.
SUSPECT_IP_COUNTRY	Checks whether the buyer's country, identified by their IP address, is on the list of forbidden countries.

Table 11: List of fraud verification processes

The possible values for **result** are:

Value	Description
OK	OK.
WARNING	Informational control failed.
ERROR	Blocking control failed.

Table 12: List of fraud verification processes

14. Retrieve the card type used for the payment.

Two scenarios are possible:

- For a payment processed with **only one card**. The fields to process are:

Field name	Description
vads_card_brand	Brand of the card used for the payment, e.g.: CB, VISA, VISA_ELECTRON, MASTERCARD, MAESTRO, VPAY
vads_card_number	Card number used for the payment.
vads_expiry_month	Expiry month between 1 and 12 (e.g.: 3 for March, 10 for October).
vads_expiry_year	Expiry year in 4 digits (e.g.: 2023).
vads_bank_code	Code of the issuing bank
vads_bank_product	Product code of the card
vads_card_country	Country code of the country where the card was issued (alpha ISO 3166-2 code, e.g.: "FR" for France, "PF" for French Polynesia, "NC" for New Caledonia, "US" for the United States).

Table 13: Analysis of the card used for the payment

- For a **split payment** (i.e. a transaction using several payment methods), the following fields must be processed:

Field name	Value	Description
vads_card_brand	MULTI	Several types of payment card are used for the payment.
vads_payment_seq	In Json format, see details below.	Details of performed transactions.

The **vads_payment_seq** field (json format) describes the split payment sequence. It contains:

1. "trans_id": transaction identifier used for the entire payment sequence.
2. "transaction": transaction table of the sequence. It contains the following elements:

Field name	Description
amount	Amount of the payment sequence.
operation_type	Debit transaction.
auth_number	Authorization number. Will not be returned if not applicable to the used payment method. Example: 949478
auth_result	Return code of the authorization request.
capture_delay	Delay before the capture (in days).

<table>
<tr><th>Field name</th><th>Description</th></tr>
<tr><td></td><td>• For a payment by card, this parameter is the requested capture date (ISO 8601 format). If not sent in the payment form, the value defined in the Expert Back Office will be used.</td></tr>
<tr><td>card_brand</td><td>Used payment method.
For a payment by card (e.g. CB or Visa or MasterCard co-branded CB cards), this parameter is set to "CB".
See the Payment Gateway Implementation Guide available in our online documentation archive to see the complete list of card types.</td></tr>
<tr><td>card_number</td><td>Payment method number</td></tr>
<tr><td>expiry_month</td><td>Expiry month of the payment method.</td></tr>
<tr><td>expiry_year</td><td>Expiry year of the payment method.</td></tr>
<tr><td>payment_certificate</td><td>Payment certificate.</td></tr>
<tr><td>contract_used</td><td>Contract used for the payment</td></tr>
<tr><td>identifier</td><td>Unique identifier (token) associated with a payment method.</td></tr>
<tr><td>identifier_status</td><td>Only present if the requested action is a token creation or update.
Possible values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>CREATED</td><td>The authorization request has been accepted.
The token has been successfully created.</td></tr>
<tr><td>NOT_CREATED</td><td>The authorization request has been declined.
The token has not been created, and therefore cannot be viewed in the Expert Back Office.</td></tr>
<tr><td>UPDATED</td><td>The token has been successfully updated.</td></tr>
<tr><td>NOT_UPDATED</td><td>The token has not been updated.</td></tr>
<tr><td>ABANDONED</td><td>The action has been abandoned by the buyer (debtor).
The token has not been created, and therefore cannot be viewed in the Expert Back Office.</td></tr>
</table></td></tr>
<tr><td>presentation_date</td><td>For a payments by card, this parameter is the requested capture date (ISO 8601 format).</td></tr>
<tr><td>trans_id</td><td>Transaction number.</td></tr>
<tr><td>ext_trans_id</td><td>This field is not sent for credit card payments.</td></tr>
<tr><td>trans_uuid</td><td>Unique reference generated by the payment gateway after the creation of a payment transaction.
Guarantees that each transaction is unique.</td></tr>
<tr><td>extra_result</td><td>Numeric code of the risk assessment result.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>Empty</td><td>No verification completed.</td></tr>
<tr><td>00</td><td>All the verification processes have been successfully completed.</td></tr>
<tr><td>02</td><td>Credit card velocity exceeded.</td></tr>
<tr><td>03</td><td>The card is on the Merchant’s greylist.</td></tr>
<tr><td>04</td><td>The country of origin of the card is on the Merchant’s greylist.</td></tr>
<tr><td>05</td><td>The IP address is on the Merchant’s greylist.</td></tr>
<tr><td>06</td><td>The BIN code is on the Merchant’s greylist.</td></tr>
<tr><td>07</td><td>Detection of an e-carte bleue.</td></tr>
<tr><td>08</td><td>Detection of a national commercial card.</td></tr>
<tr><td>09</td><td>Detection of a foreign commercial card.</td></tr>
<tr><td>14</td><td>Detection of a card that requires systematic authorization.</td></tr>
<tr><td>20</td><td>Relevance verification: countries do not match (country IP address, card country, buyer's country).</td></tr>
<tr><td>30</td><td>The country of the this IP address belongs to the greylist.</td></tr>
<tr><td>99</td><td>Technical issue encountered by the server during a local verification process.</td></tr>
</table></td></tr>
<tr><td>sequence_number</td><td>Sequence number.</td></tr>
<tr><td>trans_status</td><td>Transaction status.</td></tr>
</table>

Table 14: JSON object content

Note: canceled transactions are also displayed in the table.

15. Store the value of the **vads_trans_uuid** field. It will allow you to assign unique identification to the transaction if you use the Web Service APIs.

16. Retrieve all the order, buyer and shipping details.

    These details will be provided in the response only of they have been transmitted in the payment form.

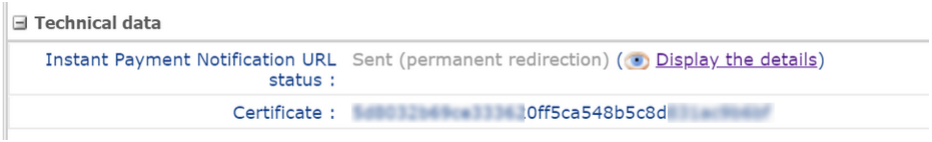
    Their values are identical to the ones submitted in the form.

17. Proceed to order update.

7.7. Running tests and troubleshooting

In order to test the notifications, follow the the steps below:

1. Make a payment (in TEST mode or in PRODUCTION mode).
2. Once the payment is complete, look for the transaction in your Back Office (**Management** > **Transactions** or **TEST Transactions** menu if you made the payment in TEST mode).
3. Double-click the transaction to view the **transaction details**.
4. In the transaction details, search for the section entitled **Technical data**.
5. Check the status of the Instant Payment Notification URL:



The list of possible statuses is provided below:

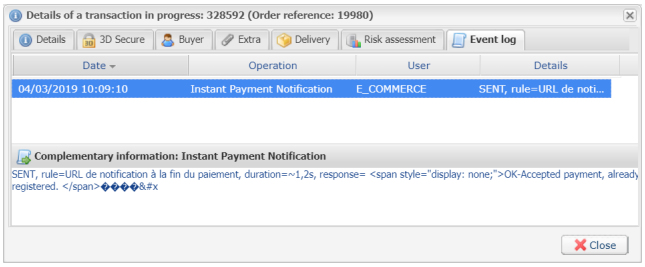
Status	Description
N/A	The transaction did not result in a notification or no notification rules have been enabled.
Undefined URL	An event has triggered the notification rule for end of payment but the URL is not configured.
Call in progress	The notification is in progress. This status is temporary.
Sent	The notification has been successfully sent and a remote device returned a HTTP 200, 201, 202, 203, 204, 205 or 206 response status code.
Sent (permanent redirection)	The merchant website has returned a HTTP 301 or 308 response status code with a new URL to contact. A new call in POST mode has been made to the new URL.
Sent (temporary redirection)	The merchant website has returned a HTTP 302 or 307 response status code with a new URL to contact. A new call in POST mode has been made to the new URL.
Sent (redirection to another page)	The merchant website has returned a HTTP 303 response status code with a new URL to contact. A new call in GET mode has been made to the new URL.
Failed	Generic error different from the codes described below.
Server unavailable	The notification has lasted more than 35s.
SSL handshake failure	Your server is incorrectly configured. Run a test on the Qualys website (*https://www.ssllabs.com/ssltest/*) and correct the errors.
Connection interrupted	Communication error.
Connection refused	Communication error.
Server error 300	Case of redirection not supported by the gateway.
Server error 304	Case of redirection not supported by the gateway.
Server error 305	Case of redirection not supported by the gateway.
Server error 400	The merchant website has returned a HTTP 400 Bad Request response status code.
Server error 401	The merchant website has returned a HTTP 401 Unauthorized response status code. Make sure that the resource is not protected by an .htaccess file.
Server error 402	The merchant website has returned a HTTP 402 Payment Required response status code.
Server error 403	The merchant website has returned a HTTP 403 Forbidden response status code. Make sure that the resource is not protected by an .htaccess file.
Server error 404	The merchant website has returned a HTTP 404 Not Found response status code. Make sure that the URL is correctly specified in the rule configuration. Make sure that the file is present on your server.
Server error 405	The merchant website has returned a HTTP 405 Method Not Allowed response status code.

Status	Description
Server error 406	The merchant website has returned a HTTP 406 Not Acceptable response status code.
Server error 407	The merchant website has returned a HTTP 407 Proxy Authentication Required response status code.
Server error 408	The merchant website has returned a HTTP 408 Request Time-out response status code.
Server error 409	The merchant website has returned a HTTP 409 Conflict response status code.
Server error 410	The merchant website has returned a HTTP 410 Gone response status code.
Server error 411	The merchant website has returned a HTTP 411 Length Required response status code.
Server error 412	The merchant website has returned a HTTP 412 Precondition Failed response status code.
Server error 413	The merchant website has returned a HTTP 413 Request Too Large response status code.
Server error 414	The merchant website has returned a HTTP 414 Request-URI Too Long response status code.
Server error 415	The merchant website has returned a HTTP 415 Unsupported Media Type response status code.
Server error 500	The merchant website has returned a HTTP 500 Internal Server Error response status code. An application error has occurred on the level of the server hosting your shop. See the logs of your HTTP server (usually apache). The issue can only be corrected with an intervention on your server.
Server error 501	The merchant website has returned a HTTP 501 Not Implemented response status code.
Server error 502	The merchant website has returned a HTTP 502 Bad Gateway / Proxy Error response status code.
Server error 503	The merchant website has returned a HTTP 503 Service Unavailable response status code.
Server error 504	The merchant website has returned a HTTP 504 Gateway Time-out response status code. The merchant server has not accepted the call within the time limit of 10s.
Server error 505	The merchant website has returned a HTTP 505 HTTP Version Not Supported response status code.

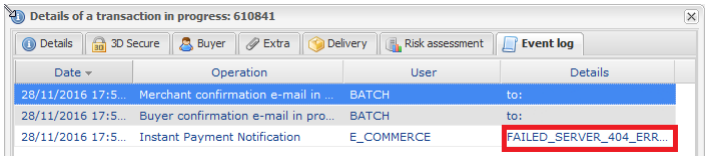
For more information on a notification, click the link **Display the details** or click the **Event log** tab and search for the line **Notification URL call**.

In order to help the merchant identify the source of the error, the gateway systematically analyses the 512 first characters returned by the merchant website and displays them in the **Details** column.

- Example of a successfully processed notification:



- Example of a failed notification:



If the payment gateway is unable to access the URL of your page, an e-mail alert will be sent to the shop administrator.

It contains:

- The HTTP code of the encountered error
- Parts of error analysis
- Its consequences
- Instructions to follow via the Expert Back Office for resending the request to the URL specified in step 4.

8. RETURNING TO THE SHOP

By default, when the buyer returns to the merchant website, no parameters will be transmitted by their browser.

However, if the **vads_return_mode** field has been transmitted in the payment form (see chapter **Managing the return to the merchant website**of the Hosted Payment Page Implementation Guide available in our online document archive) it will be possible to retrieve the data:

- either via GET, the data is presented in the URL as follows: ?field1=value1&field2=value2
- or via POST: the data is sent in a POST form.

The data transmitted to the browser is the same as for notifications (IPN).

The **vads_url_check_src** and **vads_hash** fields will be sent only in the instant notification.

To analyze this data, see chapter **Analyzing the payment result**.

Note: the return to the shop will allow you to show only the visual context to the buyer. Do not use the received data for processing in the database.

