

REST-API Shopware Connect Platform

support@bepado.com

26th August 2014



Contents

REST-API Shopware Connect Platform	1
Authentication	1
Ping / Test Authentication	2
Update Order Status	2
Register Import CSV Url	2
Register Event Hooks	3
Ensuring Message Delivery	3
Event “order_created”	4
Event: “order_status_updated”	6
Event: “order_payment_status_updated”	6
Request import report list	7
Request single import report	7

REST-API Shopware Connect Platform

Some of the SDK calls to Shopware Connect can be done without the SDK, using the REST-API. The SDK has convenience methods for these calls if you are using PHP anyways.

Authentication

Authentication to the REST API happens via HTTP headers using HMAC-SHA512. With every request you have to transmit:

X-Shopware-Connect-Shop: Id
X-Shopware-Connect-Key: HMAC-Key

The shop id can be found in the Shopware Connect “Exchange” (Synchronization) tab in your account. The Key is a computed value using the Shopware Connect API Key.

The HMAC Key is generated using SHA-512 as an algorithm, the secret is the Shopware Connect API Key and the payload is the JSON formatted body. The PHP code to build the secret is:

```
<?php
$data = array( 'pay' => 'load' );
$payload = json_encode($data);
$key = hash_hmac( 'sha512', $payload, $apiKey );
```

Ping / Test Authentication

You can test the authentication with the REST API by using the Ping endpoint:

Endpoint: `https://sn.bepado.de/sdk/ping` Method: POST

Every payload you send will be returned directly to you if everything was successful. A statuscode 200 will be set as well. If any error in authentication happens you will receive a status code in the range of 400.

Example:

POST `https://sn.bepado.de/sdk/ping`

X-Shopware-Connect-Shop: 1

X-Shopware-Connect-Key: abcdefg

```
{
  "Hello": "World"
}
```

Update Order Status

As a supplier (Lieferant) I can update the status of a Shopware Connect order so that the dealer (Händler) and end-customer can be informed about changes.

Endpoint: `https://sn.bepado.de/sdk/update-order-status` Method: POST Payload:

- id - The order ID of the transaction that was generated by the supplier
- status - One of open, in_process, shipped, delivered, canceled or error.
- tracking - A structure with keys id, url and vendor. The value is optional

Example:

POST `https://sn.bepado.de/sdk/update-order-status`

X-Shopware-Connect-Shop: 1

X-Shopware-Connect-Key: abcdefg

```
{
  "id": "1234",
  "status": "in_process",
  "tracking": {"id": "ABCDEFGH1234567890"}
}
```

Register Import CSV Url

For accounts that use the CSV shop you can register a feed url to download products from instead of using the FTP.

The following call has to be made to register the URL:

```
POST https://sn.bepado.de/sdk/provider/import-url
X-Shopware-Connect-ShopId: <ShopId>
X-Shopware-Connect-Key: <hmac-key>
```

```
{
  "url": "<import csv url>"
}
```

Register Event Hooks

Shopware Connect provides event hooks for events. Whenever something interesting happens inside Shopware Connect you may opt-in to get notified about these events via Webhook.

Currently the following events exist:

- `order_created` is triggered when an order was created through Shopware Connect. It will be published to both parties (supplier and dealer) when they have a hook registered. The dealer can receive multiple events for the same order when different suppliers took part in the transaction.
- `order_status_updated` is triggered when either supplier or merchant update the status of the order.
- `order_payment_status_updated` is triggered when information about the payment between merchant and supplier are updated.

Endpoint: `https://sn.bepado.de/sdk/hooks` Method: POST/DELETE Payload:

- `eventName` - Name of the event to register a hook url to or list of events separated by comma.
- `url` - URL where to send the hook to.

Example:

```
POST https://sn.bepado.de/sdk/hooks
X-Shopware-Connect-Shop: 1
X-Shopware-Connect-Key: abcdefg
```

```
{
  "eventName": "order_created , order_status_updated , order_payment_status_updated",
  "url": "http://example.com/my_hook.php"
}
```

Security of hooks: When Shopware Connect notifies your url of an event, it uses the `X-Shopware-Connect-Shop` and `X-Shopware-Connect-Key` headers as well. You can use them to verify that it was really Shopware Connect that sent you the event and not some malicious third party.

Ensuring Message Delivery

To ensure that events are really delivered we implement the webhook as replication using incremental revisions. This prevents loss of data through network outage or any problems on the receiving end of the communication.

Each registered webhook will receive a stream of events ordered by the revision number. Before any new events are sent to the server it is checked that the previous event(s) were received.

This is why your webhook implementation needs to implement two modes of operation:

1. A GET request that returns the last successfully processed event-revision.
2. a POST request that accepts the next event.

Lets take a hypothetical “http://example.org/hook” URL and look at the communication:

1. Shopware Connect asks the last processed event revision:

```
GET /hook
Host: example.org
X-Shopware-Connect-Shop: <ShopId>
X-Shopware-Connect-Key: <HMAC>
```

The X-Shopware-Connect-Key header is generated over an empty payload in this request.

2. Your server responds with an XML snippet containing the last processed event revision, for example “1”:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=UTF-8

<last-revision>1</last-revision>
```

3. Shopware Connect replicates the next event with id higher than “1” to your server:

```
POST /hook
Host: example.org
X-Shopware-Connect-Shop: <ShopId>
X-Shopware-Connect-Key: <HMAC>
X-Shopware-Connect-Event: <EventName>

<order-event>
  <revision>2</revision>
  <!-- ... -->
</order-event>
```

In this case X-Shopware-Connect-Shop and X-Shopware-Connect-Key are transmitted and allow you to verify that the event was really sent by Shopware Connect.

4. You save the Revision 2 (in this case) from the <revision>-tag in your database as the last processed revision when you can guarantee that the event was stored on your side. If your server fails to save the revision then Shopware Connect will attempt to send the event again some time later. Using a database transaction to save both the events data and update the last revision is the best way to achieve this task.

Event “order_created”

When an order is created through Shopware Connect with two or more parties you can get notified of the details of this order with the “order_created” hook. A sample XML request looks like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<order-event xmlns="http://schema.bepado.de/order+v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://schema.bepado.de/order+v1 http://schema.bepado.de/order+v1.xsd">
  <revision>1</revision>
  <event>order_created</event>
  <order transaction-id="1" transaction-date="2014-05-06_12:15:00" supplier-shop="22" supplier-id="1">
    <shipping-costs net="10" gross="11.9"/>
    <customer-total net="190" gross="214.1"/>
    <intershop-total net="110" gross="124.9"/>
    <delivery-address>
      <company>Acme Corp.</company>
      <name>Max Mustermann</name>
      <first-name>Max</first-name>
      <middle-name/>
      <sur-name>Mustermann</sur-name>
      <street>Neustrasse</street>
      <street-number>22</street-number>
      <door-code>a</door-code>
      <additional-address-line>foo</additional-address-line>
      <zip>12345</zip>
      <city>Neustadt</city>
      <country>DEU</country>
    </delivery-address>
    <billing-address>
      <company>Acme Corp.</company>
      <first-name>Max</first-name>
      <sur-name>Mustermann</sur-name>
      <street>Neustrasse</street>
      <street-number>12-14a</street-number>
      <door-code/>
      <additional-address-line/>
      <zip>12345</zip>
      <city>Neustadt</city>
      <country>DEU</country>
      <phone>+1234</phone>
      <email>max@mustermann.de</email>
    </billing-address>
    <order-items>
      <item id="100-1" source-id="1" count="1">
        <customer-price net="100" gross="107"/>
        <intershop-price net="50" gross="53.5"/>
      </item>
      <item id="100-2" source-id="2" count="3">
        <customer-price net="90" gross="107.1"/>
        <intershop-price net="60" gross="71.4"/>
      </item>
    </order-items>
  </order>
</order-event>

```

Event: “order_status_updated”

When the status of an order is updated an event is fired for this change.

The following order status values are allowed:

- open - When the transaction/order is created
- in_process - When the supplier started processing the order
- shipped - When the supplier sent out the order to his shipping company
- delivered - When the shipping company marked the order as delivered and the supplier marks the order as delivered
- canceled - When either merchant or supplier mark the order as canceled for various reasons (end customer canceled, untrustworthy customer, payment not accepted, ...)
- error - When automatic processing failed

Please note that not all shop systems can transition to all states.

```
<?xml version="1.0" encoding="UTF-8"?>
<order-event xmlns="http://schema.bepado.de/order+v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <revision>2</revision>
  <event>order_status_updated</event>
  <order transaction-id="1" transaction-date="2014-05-06_12:15:00" supplier-shop="22"
    <status>in_process</status>
  </order>
</order>
```

Event: “order_payment_status_updated”

When the payment status of an order is updated an event is fired for this change. A payment here is the payment of the merchant shop to the supplier. No information is exchanged about the payment status of the end customer.

The interesting payment states are:

- “instructed” - When the supplier marks an order paid by invoice as instructed to his bank.
- “received” - When the payment provider executed the payment or the supplier marked the invoice as paid.

```
<?xml version="1.0" encoding="UTF-8"?>
<order-event xmlns="http://schema.bepado.de/order+v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <revision>3</revision>
  <event>order_payment_status_updated</event>
  <order transaction-id="1" transaction-date="2014-05-06_12:15:00" supplier-shop="22"
    <payment-status>received</payment-status>
  </order>
</order>
```

Request import report list

You can request list off all import reports for certain shop.

Example:

```
GET https://sn.bepado.de/api/provider/reports
X-Shopware-Connect-ShopId: <ShopId>
X-Shopware-Connect-Key: <hmac-key>
```

```
{ "id": "2", "date": "1421336826", "modified": "1421336787", "md5": "624d37d0097d265e71e4ca34ec"
```

Request single import report

You can request single import report by id.

Example:

```
GET http://sn.bepado.de/api/provider/report/1
X-Shopware-Connect-ShopId: <ShopId>
X-Shopware-Connect-Key: <hmac-key>
```

```
{ "issues": [], "id": null, "date": "1420794200", "modified": "1420793322", "md5": "66fae22027976d"
```

Example:

```
GET http://sn.bepado.de/api/provider/report/2
X-Shopware-Connect-ShopId: <ShopId>
X-Shopware-Connect-Key: <hmac-key>
```

```
{ "issues": [ { "severity": "error", "message": "Field \"%field%\" must not be empty.", "paramet
```