REST-API Bepado Platform

support@bepado.com

26th August 2014

Contents

REST-API Bepado Platform

Some of the SDK calls to bepado 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-Bepado-Shop: Id
X-Bepado-Key: HMAC-Key
```

The shop id can be found in the bepado "Exchange" (Synchronization) tab in your account. The Key is a computed value using the bepado API Key.

The HMAC Key is generatd using SHA-512 as an algorihtm, the secret is the bepado 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 authenticatoin 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 receieve a status code in the range of 400.

Example:

```
POST https://sn.bepado.de/sdk/ping
X-Bepado-Shop: 1
X-Bepado-Key: abcdefg

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

Update Order Status

As a supplier (Lieferant) I can update the status of a bepado order so that the dealer (Hndler) 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-Bepado-Shop: 1
X-Bepado-Key: abcdefg

{
    "id": "1234",
    "status": "in_process",
    "tracking": {"id": "ABCDEFG1234567890"}
}
```

Register Event Hooks

bepado provides event hooks for events. Whenever something interesting happens inside bepado 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 bepado. It will be published to both parties (supplier and dealer) when they have a hook registered. The dealer can recieve 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 seperated by comma.
- url - URL where to send the hook to.

Example:

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

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

Security of hooks: When bepado notifies your url of an event, it uses the X−Bepado−Shop and X−Bepado−Key headers as well. You can use them to verify that it was really bepado that sent you the event and not some malicous 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 recieving end of the communication.

Each registered webhook will recieve 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 recieved.

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. bepado asks the last processed event revision:

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

   The X−Bepado−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. bepado replicates the next event with id higher than “1” to your server:

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

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

In this case X−Bepado−Shop and X−Bepado−Key are transmitted and allow you to verify that the event was really sent by bepado.

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 bepado 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 bepado 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/
 <revision>1</revision>
 <event>order_created</event>
 <order transaction-id="1" transaction-date="2014-05-06 12:15:00" supplier-shop="22" sup
  <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>Neustra e</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>Neustra e</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/
    <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 payed by invoice as instructed to his bank.
- "received" - When the payment provider executed the payment or the supplier marked the invoice as payed.

```
<?xml version="1.0" encoding="UTF-8"?>
<order-event xmlns="http://schema.bepado.de/order+v1" xmlns:xsi="http://www.w3.org/2001/
    <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>
```