Medienwerft Punchout — User Guide

User Guide for setting up and operating Punchout Connections

Version 1.0.0 · Published 10 November 2026

Online: Developer Guide — api.punchout.medienwerft.de/api/guide

Contents

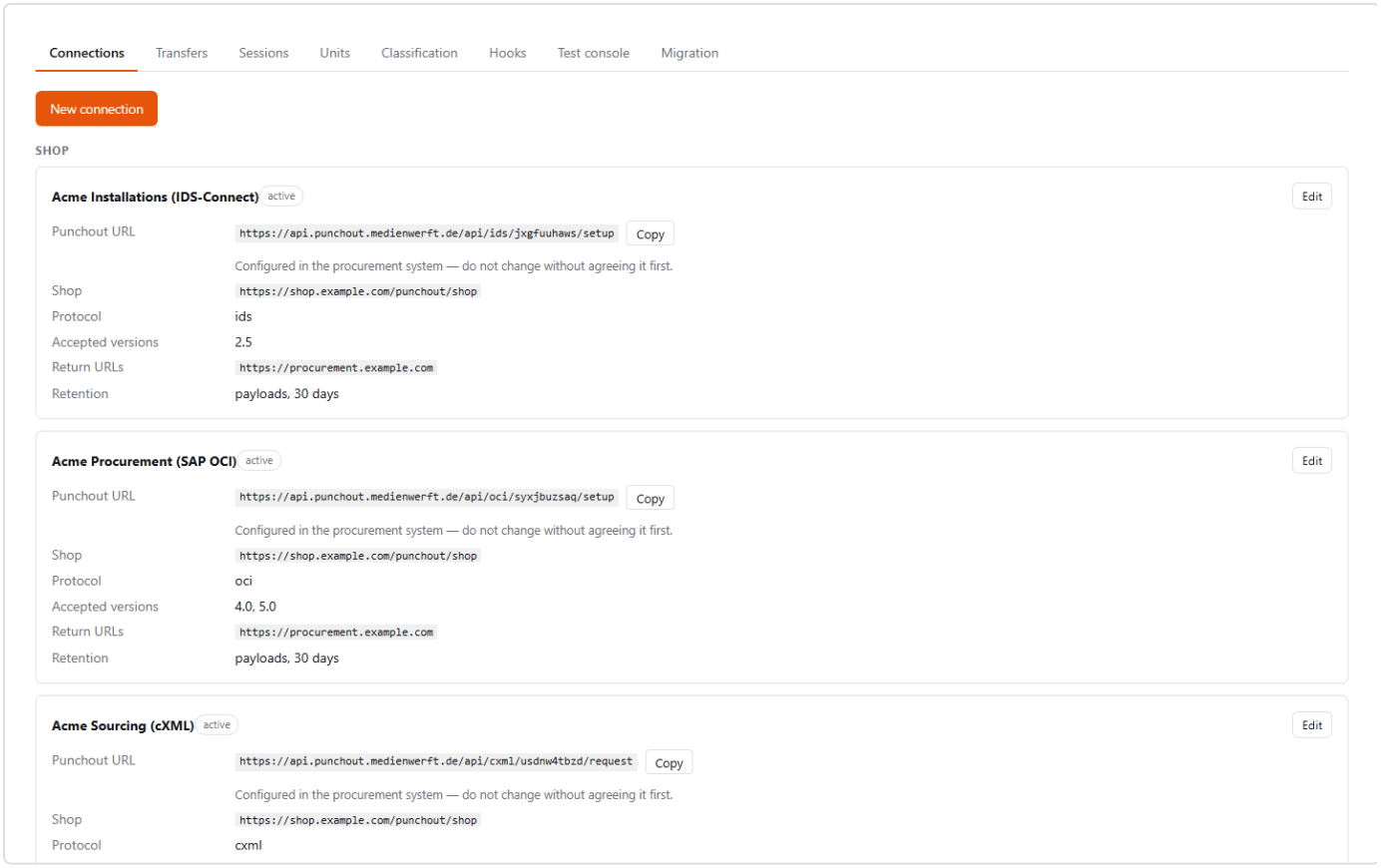
1. Create a connection
2. Allow a buyer’s return address
3. Add a customer
4. Map units and commodity codes
5. Write a hook
6. Run a test punchout
7. Read a transfer
8. Handle a failed order
9. A punchout from the shop’s side
10. Move to another instance

1. Create a connection

A connection binds one procurement system to your shop. It gets a permanent punchout URL, which the buyer enters in their own software once — so the settings around it are yours to change later, the URL is not.

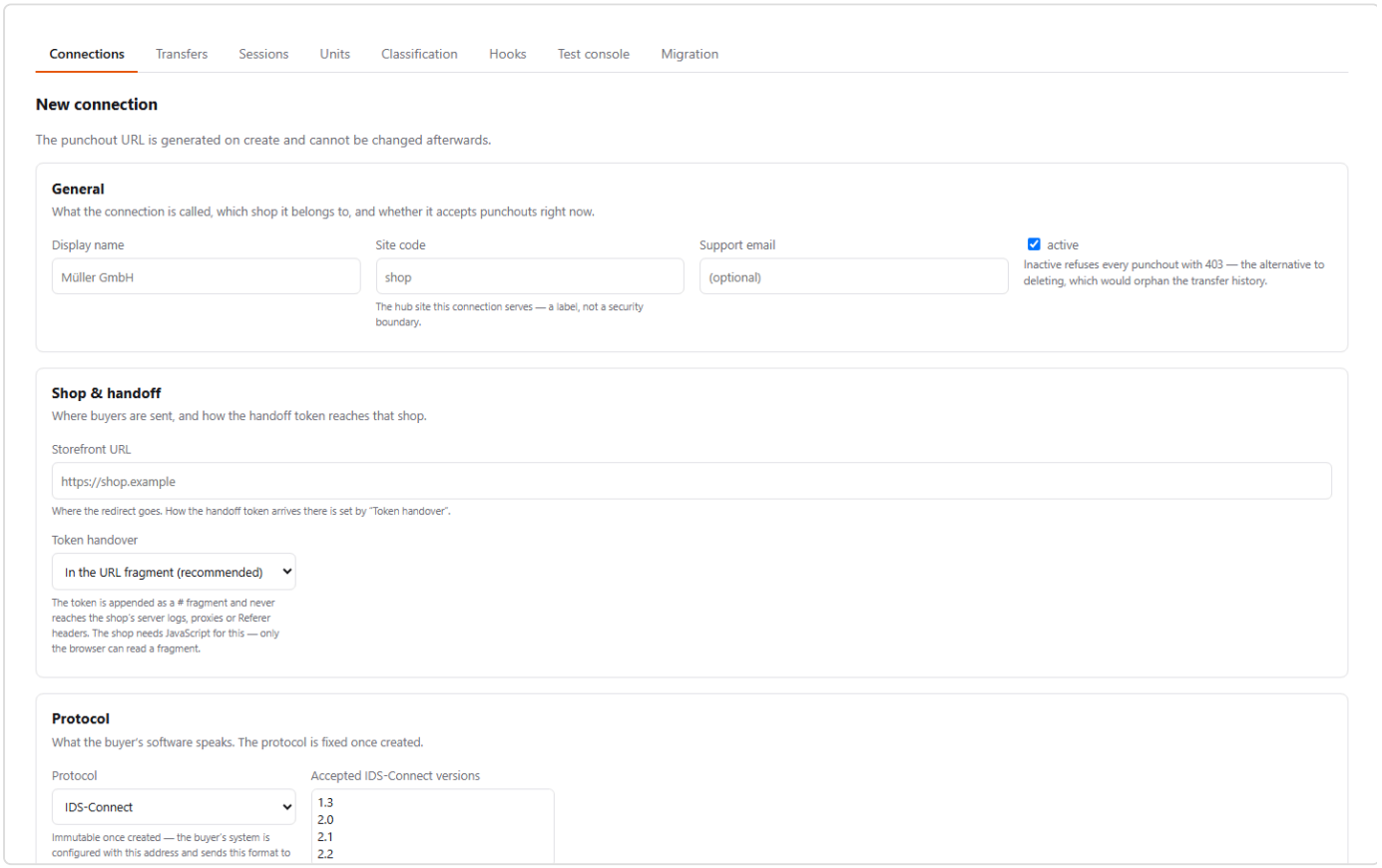
1.1 Open Connections

Every connection your tenant has, grouped by the site it serves, with the punchout URL ready to copy.



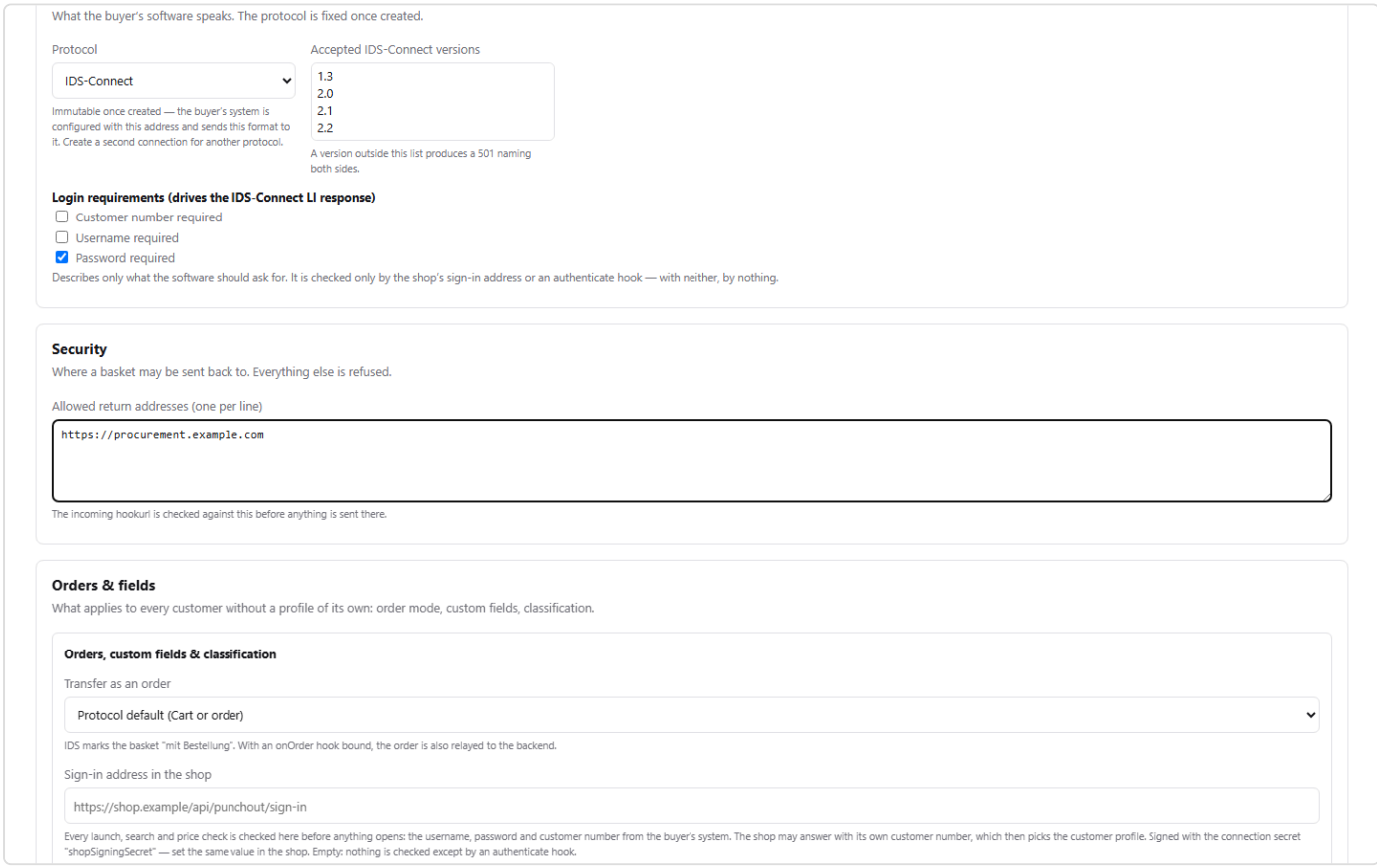
1.2 Start a new connection

The settings are grouped by what they decide. The protocol is fixed once the connection exists; everything else can change later.



1.3 Fill in the general and shop settings

The storefront URL is where buyers land. The return address is where their basket may be sent back — nothing else is ever accepted.



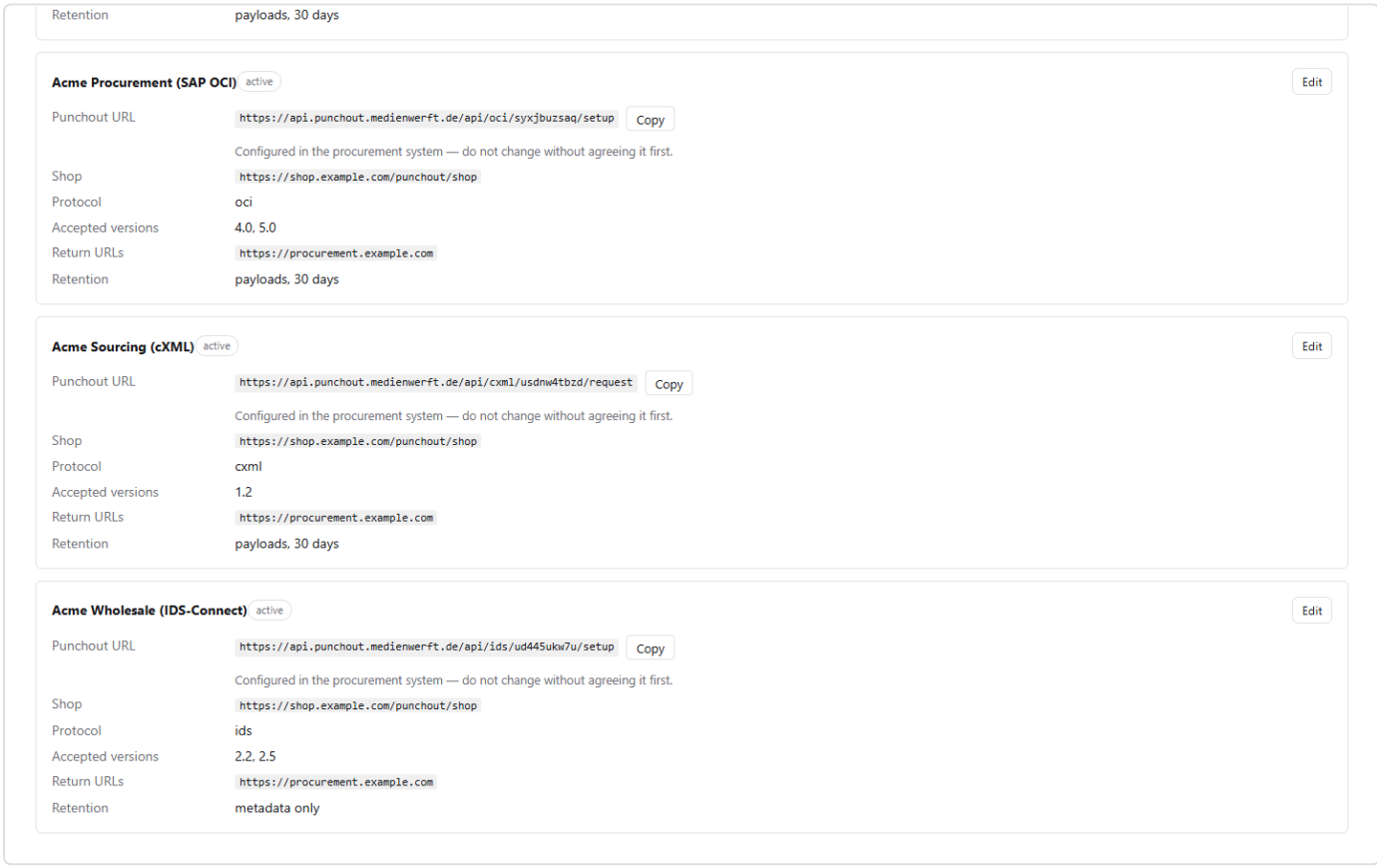
1.4 Choose the versions it accepts and what the login asks for

Select every protocol version the buyer's software may send; anything else is refused with a message naming both sides. For IDS, the login requirements tell the software what to ask the user for — customer number, username, password. They only describe it: the shop's sign-in address or an authenticate hook does the checking.



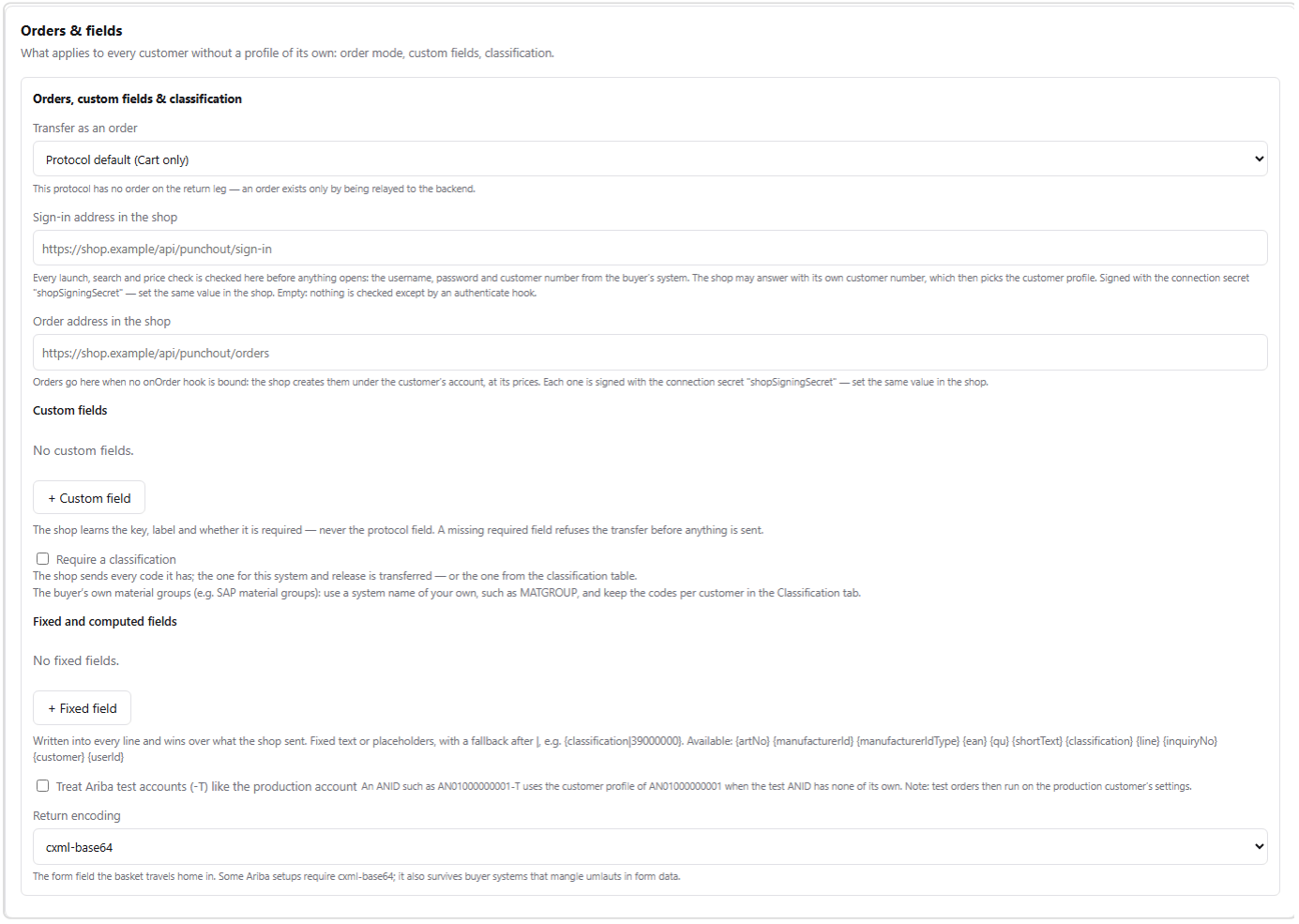
1.5 Create it and copy the punchout URL

This is the address the buyer configures in their procurement system. Send it to them once; it never changes.



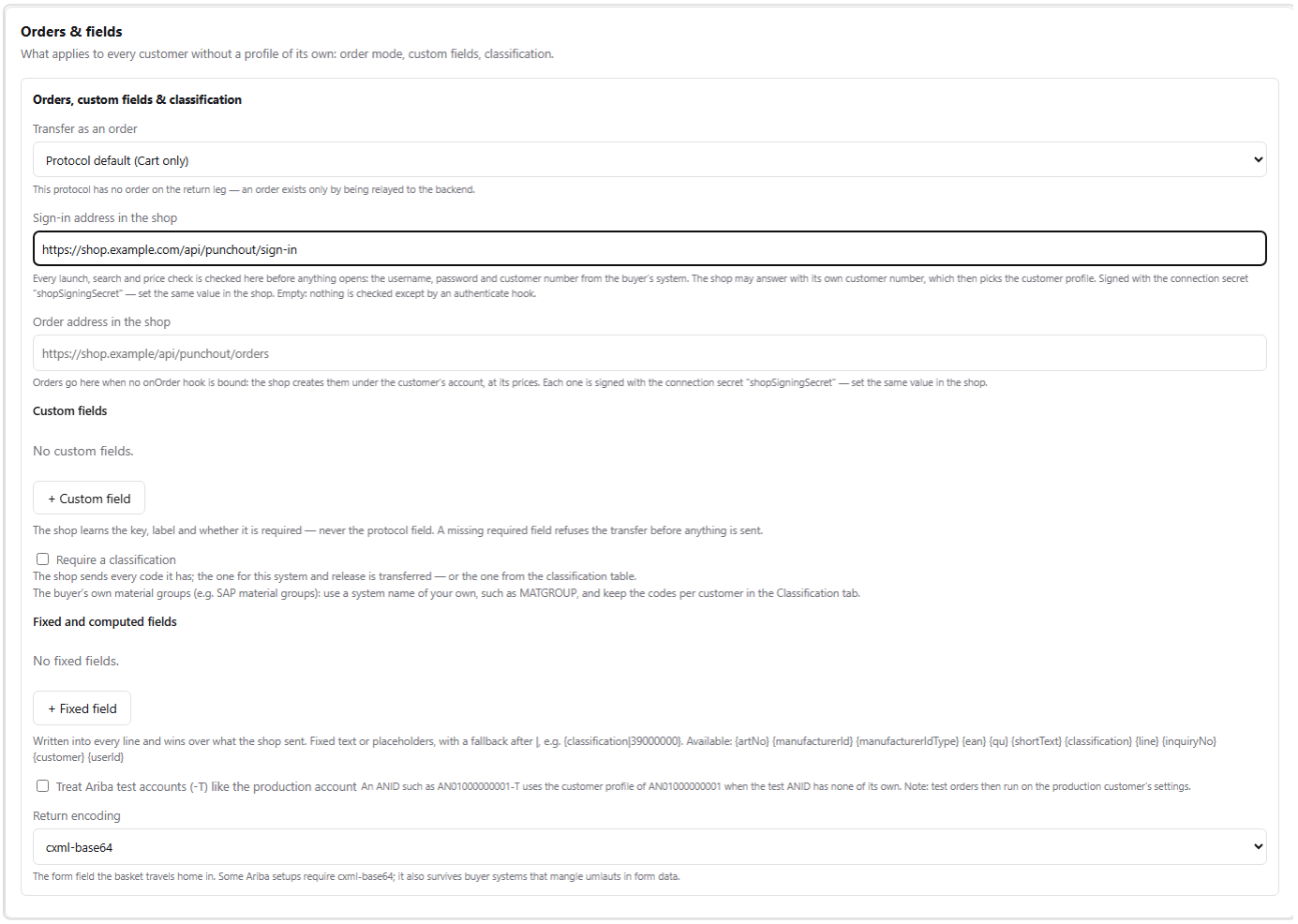
1.6 Settings that only one protocol has

The editor shows what the connection's protocol can use and nothing else. A cXML connection chooses how the basket travels home: cXML-urlencoded by default, or cxml-base64 for Ariba setups that require it — which also keeps umlauts intact.



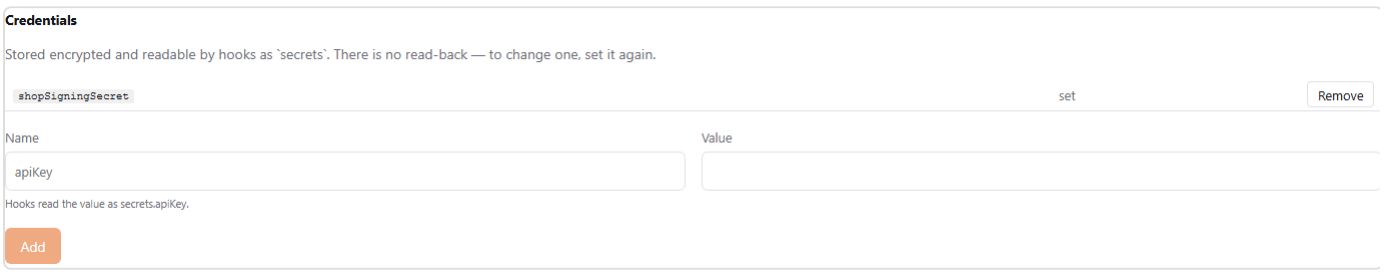
1.7 Let the shop check every login

With a sign-in address, every launch, search and price check is first sent to your shop: the customer number, username and password from the buyer's system. Only what the shop accepts opens. If the shop answers with its own customer number, that number picks the customer profile. The call is signed, so the shop answers only the punchout service — set the key under Credentials (next step).



1.8 Set the key the shop checks

Credentials appear once a connection exists. With a sign-in or order address, add shopSigningSecret with a long random value and set the same value in the shop as PUNCHOUT_SHOP_SECRET: every call to the shop is signed with it, so its public route answers only the punchout service and cannot be used to test passwords or collect customer numbers. Without it the service does not call the shop, and launches are refused. Without a sign-in or order address the section can stay empty — it adds no security on its own and otherwise only keeps values the tenant's hooks read. Only a Punchout Admin can set credentials; a value is never shown again.



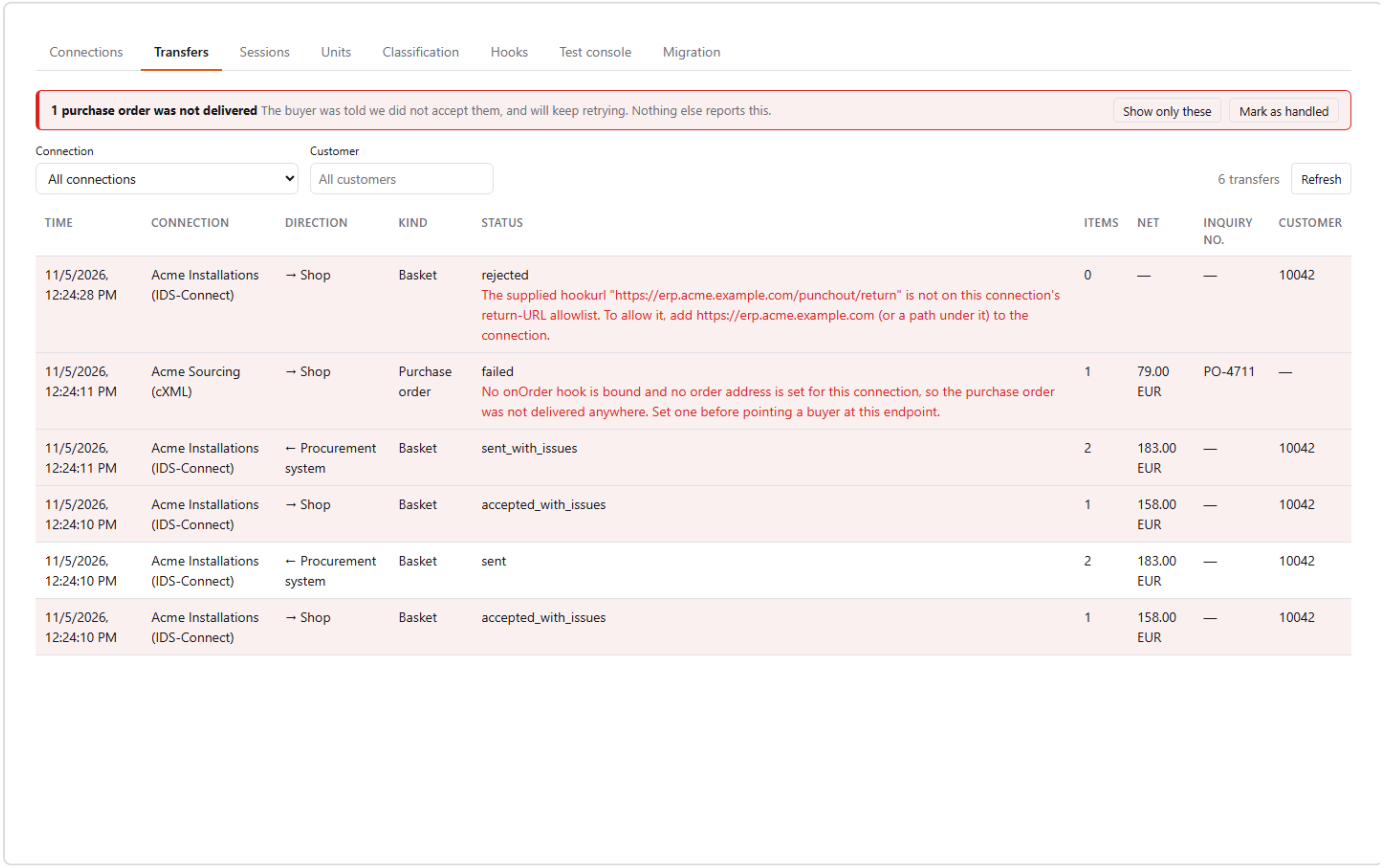
For developers: Developer Guide, 1. How a punchout reaches your shop · Developer Guide, 2. Picking up the basket — checking the login · Developer Guide, 7. Without JavaScript

2. Allow a buyer’s return address

A basket only ever goes back to an address on the connection’s allowlist. When a buyer’s system uses one that is not listed, the launch is refused — and the refusal names the address, so you know exactly what to add.

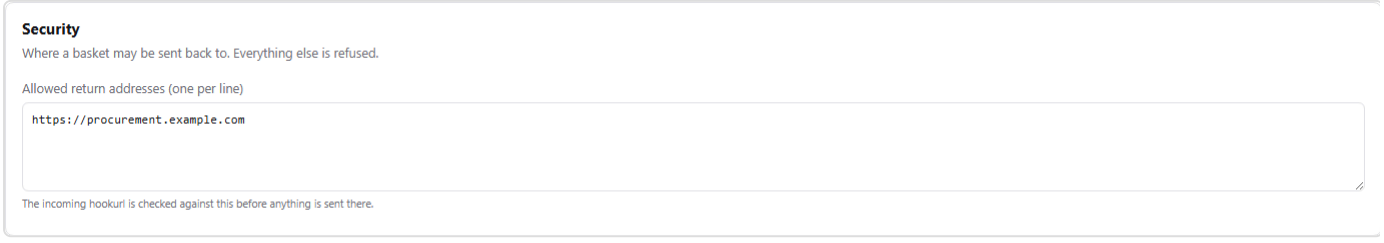
2.1 A refused launch in Transfers

The refused launch is recorded as a rejected transfer, and its message names the address and the origin to add.



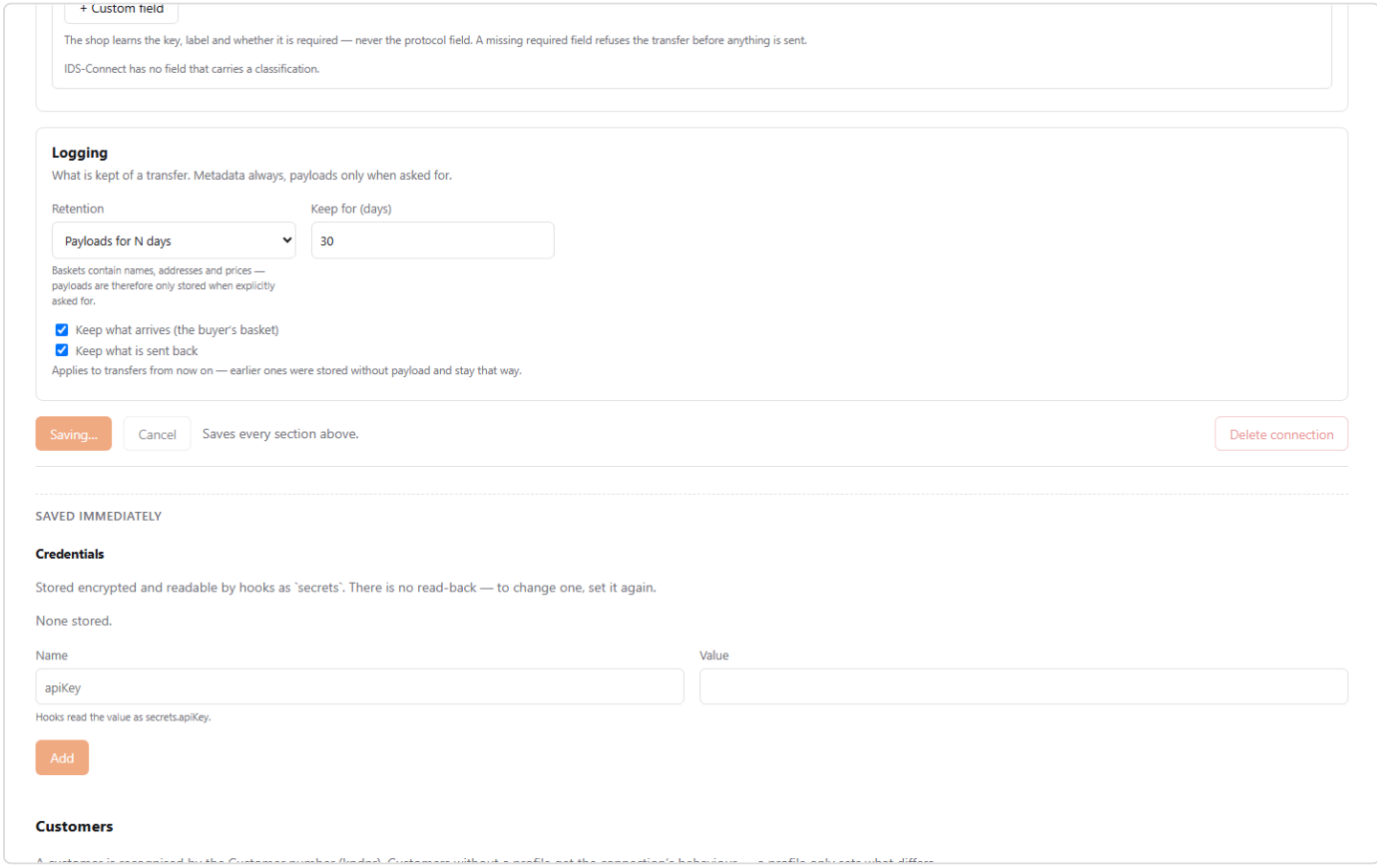
2.2 Open the connection’s Security section

One address per line. An origin allows every path under it; a path narrows it.



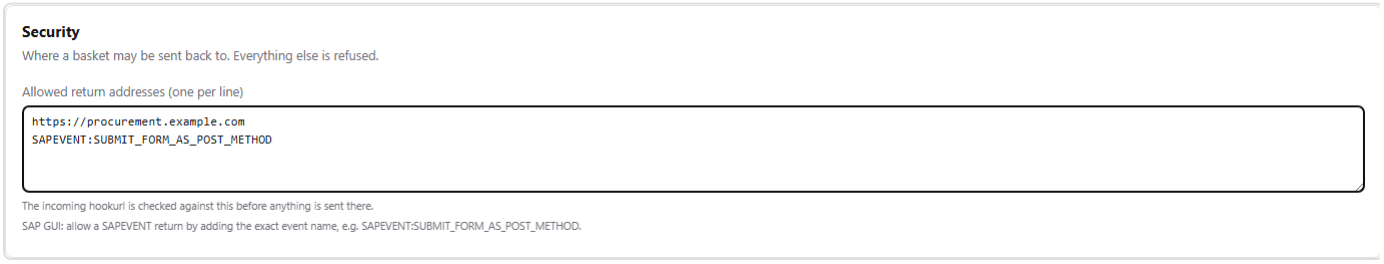
2.3 Add the buyer’s origin and save

The next launch from that address is accepted. Nothing else about the connection changes.



2.4 For SAP GUI buyers: allow the SAP event

A buyer working in the SAP GUI rather than a browser gets the basket back through an SAP event. Add it by its exact name — there is no pattern matching, and a web address never allows one.

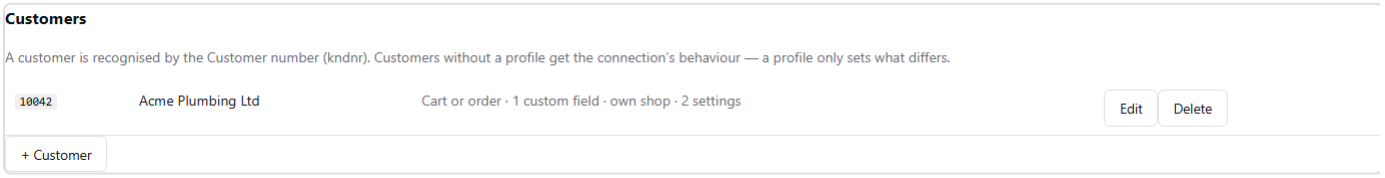


3. Add a customer

One connection serves every buyer organisation arriving over it. A customer profile changes only what differs for that customer — its own shop address, the fields it requires, its settings — and everyone else keeps the connection's behaviour.

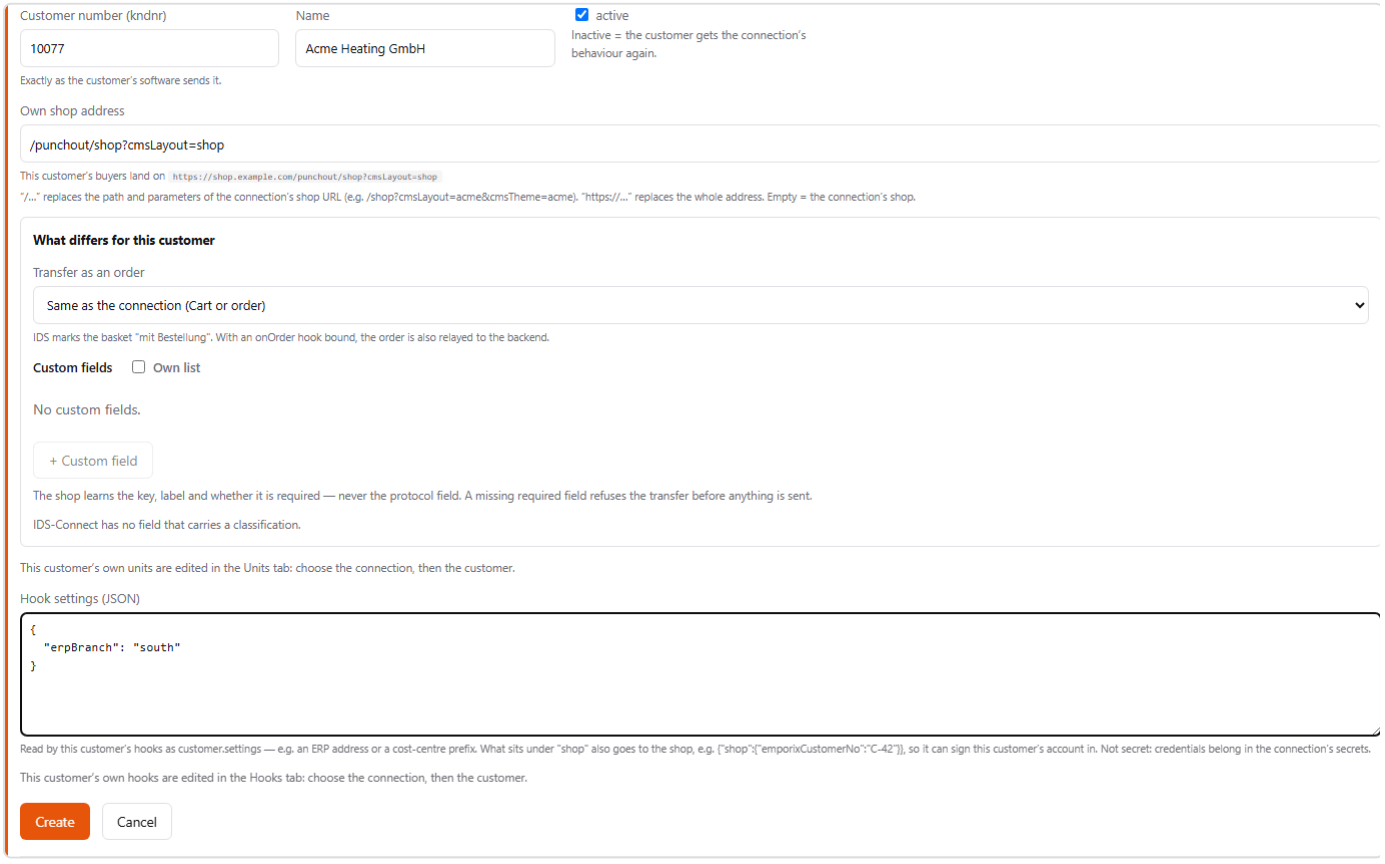
3.1 Open the connection's customers

Customers are recognised by what their protocol sends — for IDS-Connect, the customer number.



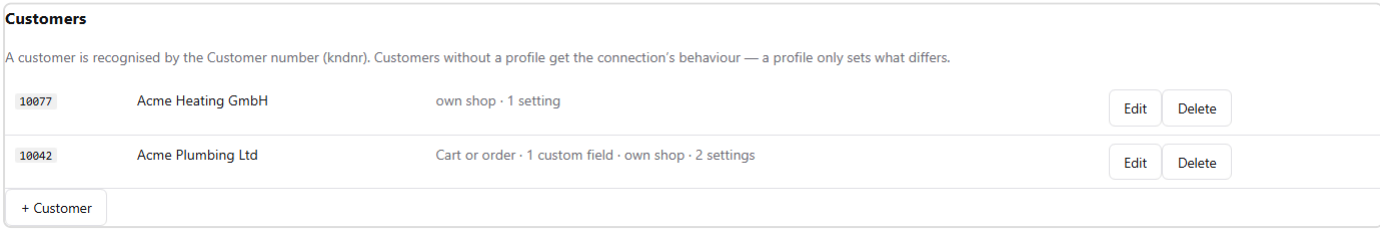
3.2 Create a customer with its own shop address

The preview shows where this customer's buyers will land. Settings are read by your hooks as customer.settings.



3.3 Save the customer

The list says in one line what is special about each customer.



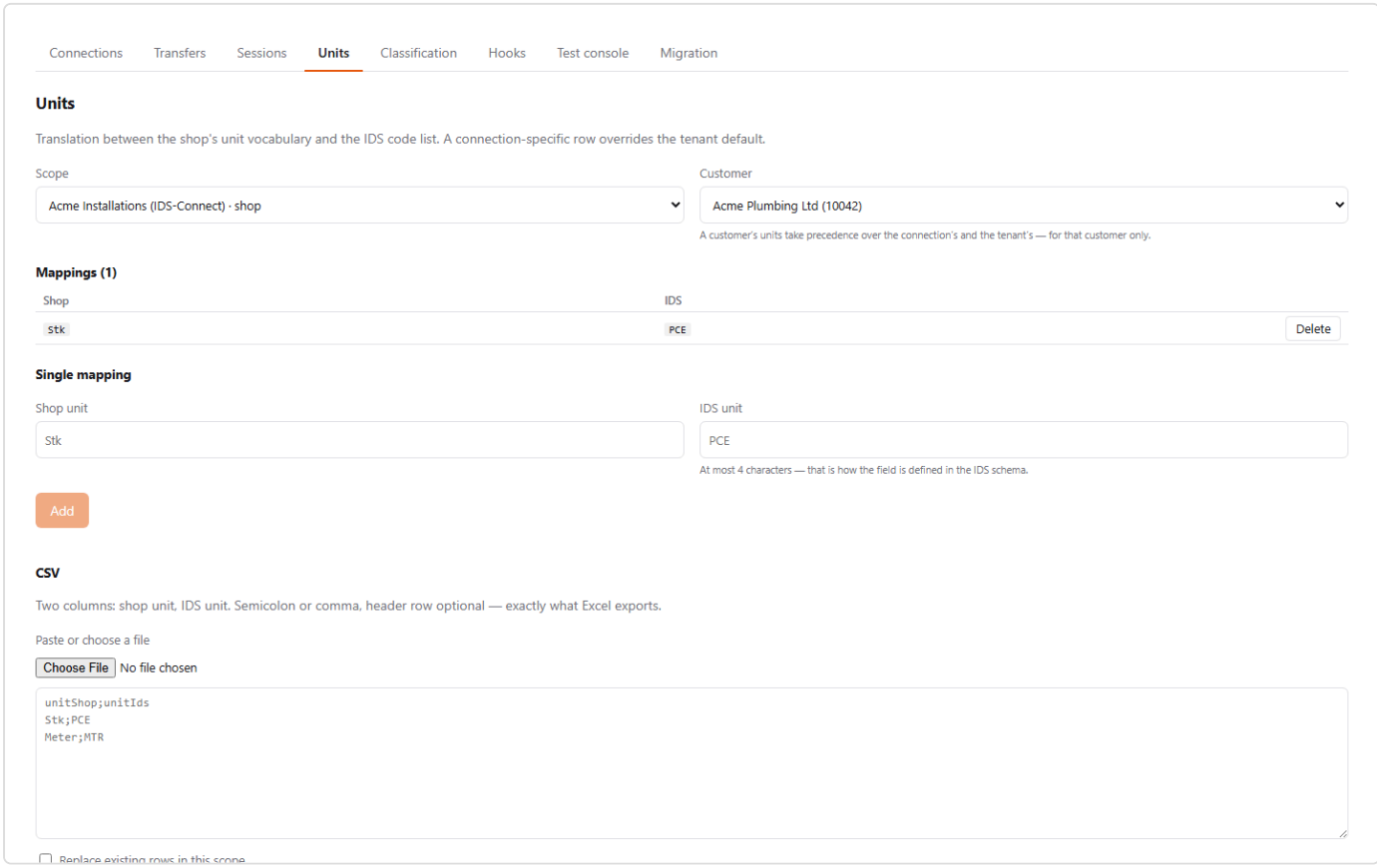
For developers: Developer Guide, 4. Reading the requirements · Developer Guide, 8. A customer's own look

4. Map units and commodity codes

Buyers and shops rarely spell units the same way, and many buyers want an eCl@ss or ETIM code on every line. Both are tables you maintain here — per tenant, per connection or per customer, the most specific one winning.

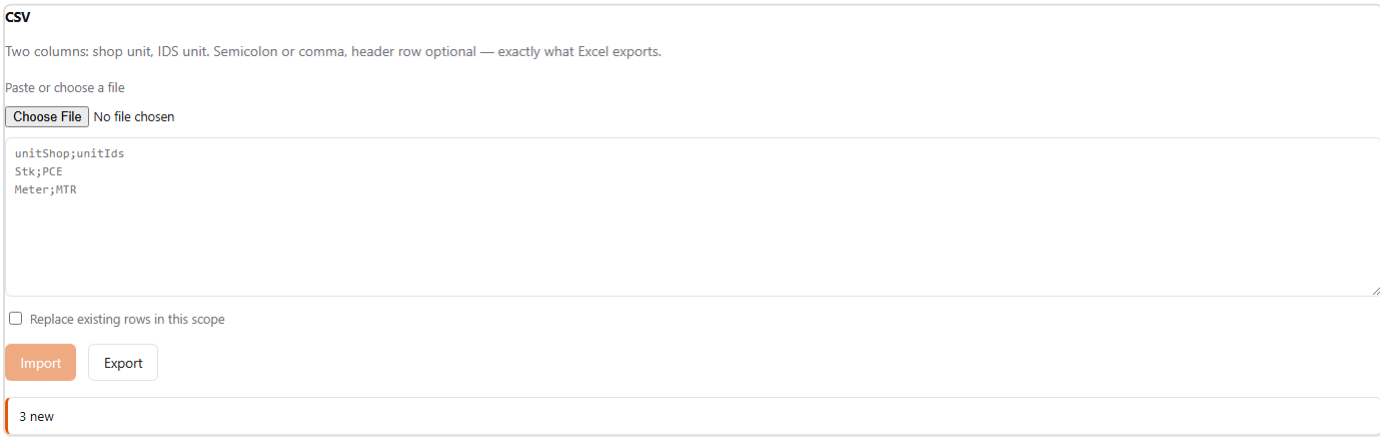
4.1 Add a unit for one customer

A customer's own units take precedence over the connection's and the tenant's, for that customer only.



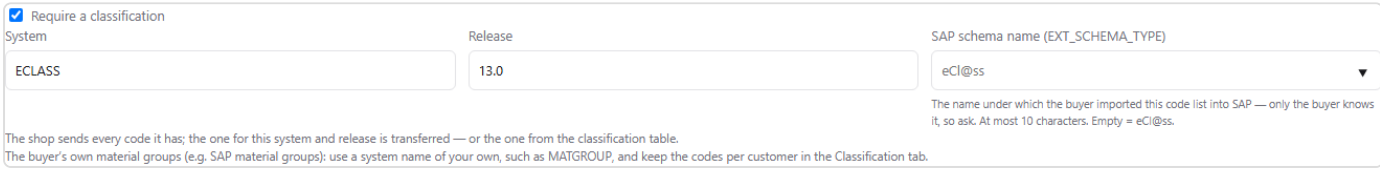
4.2 Import a whole unit list from a spreadsheet

Paste the two columns or choose the file Excel exported; Export writes the current list back out. Importing adds to the list — tick "Replace existing rows" to start over. When several shop units point at one buyer unit, the list marks them as ambiguous, and a resolveUnit hook decides.



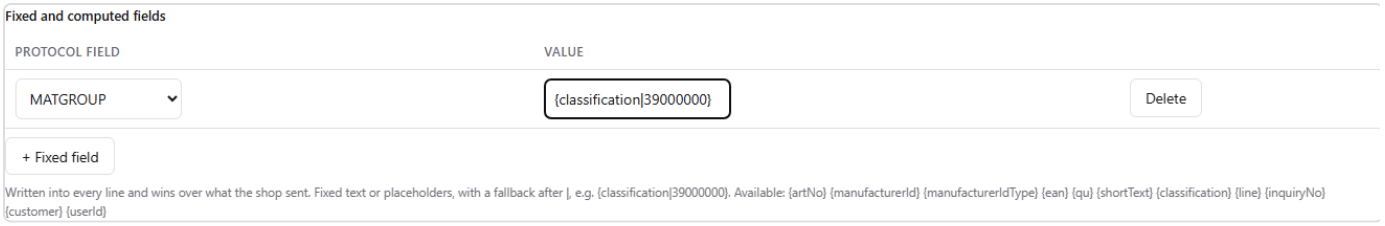
4.3 Choose which classification the buyer expects

Under Orders & fields, a connection — or one customer of it — names the system and the release its buyer wants, such as eCl@ss 13.0. The shop sends every code it has; this one is transferred, or else the one from the code table below.



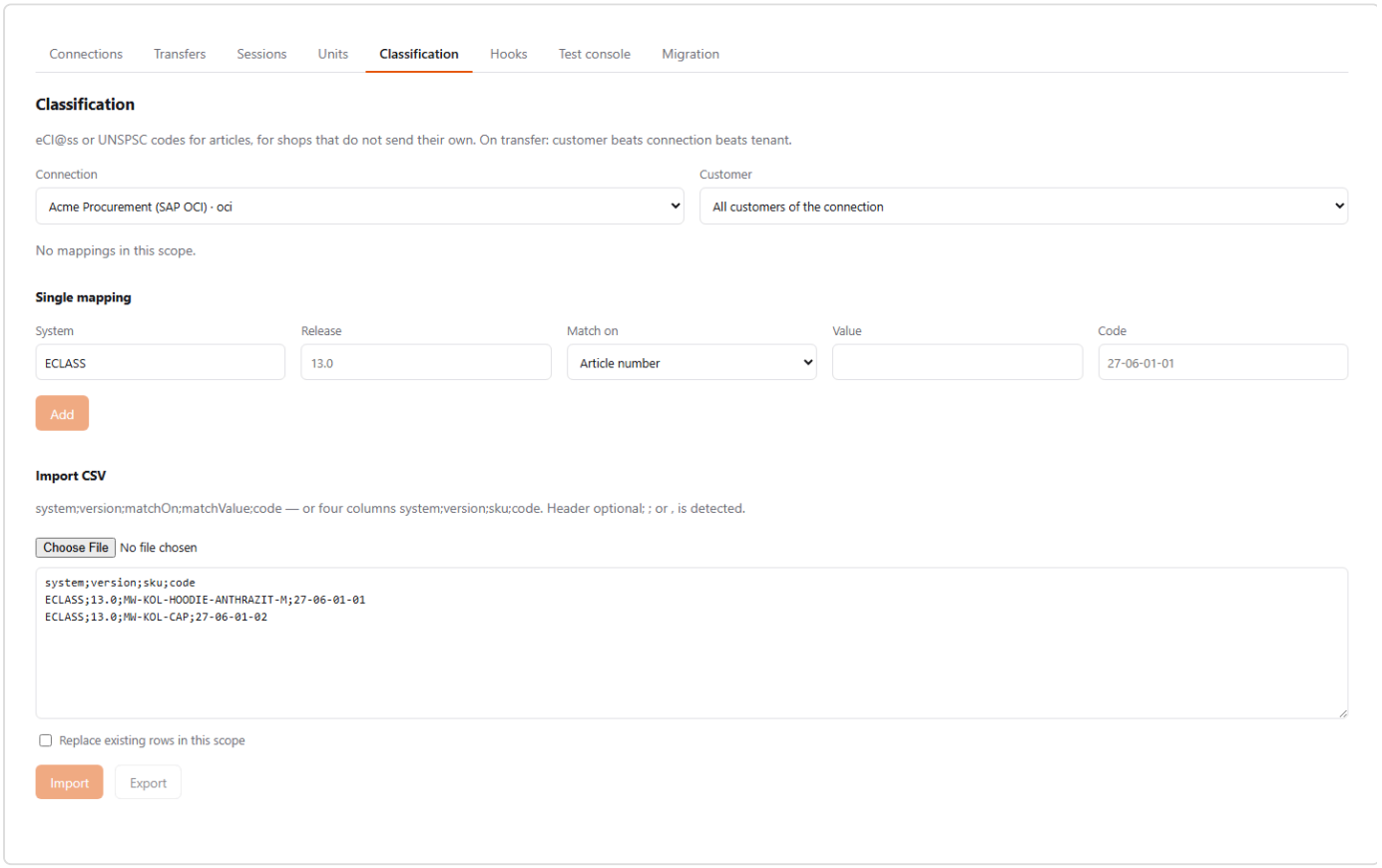
4.4 Write a fixed or computed value into every line

Some buyers need a value the shop does not have, such as a material group. A fixed field is written into every line and wins over what the shop sent — plain text, or a placeholder with a fallback after the bar: here the line's classification, or 39000000 where it has none. For OCI, SAP's lengths apply: text that is too long is shortened before it is sent (description and article number 40 characters, material group 10), and the transfer shows as sent with issues. The long text has no limit.



4.5 Import commodity codes from a spreadsheet

Lines without a code from the shop get one from this table, for the system and release the customer asked for.

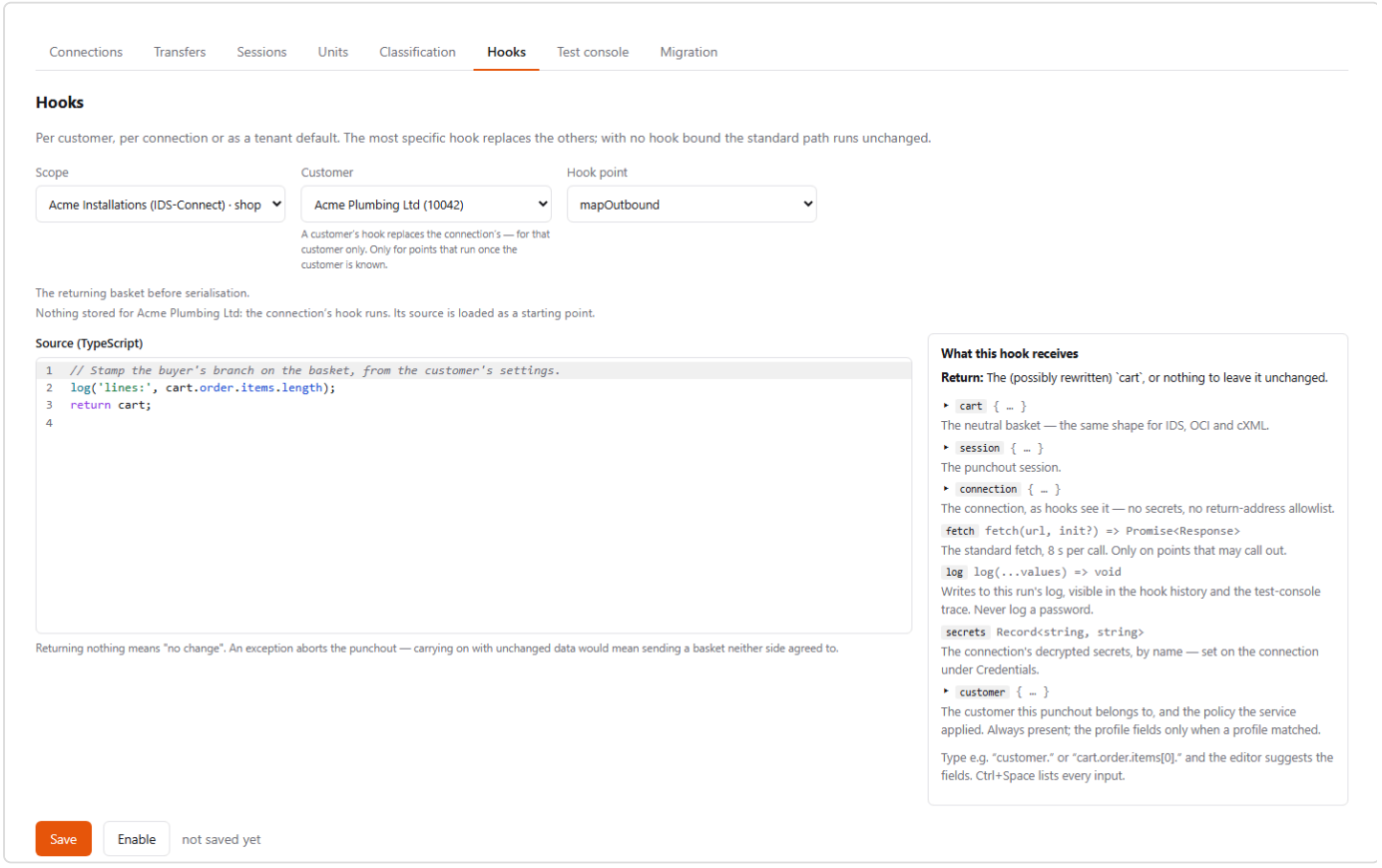


5. Write a hook

Hooks change what the punchout does at a defined point — map article numbers on the way in, enrich the basket on the way out, relay an order to your backend. They are small functions you write here; no release is needed, and every saved version is kept.

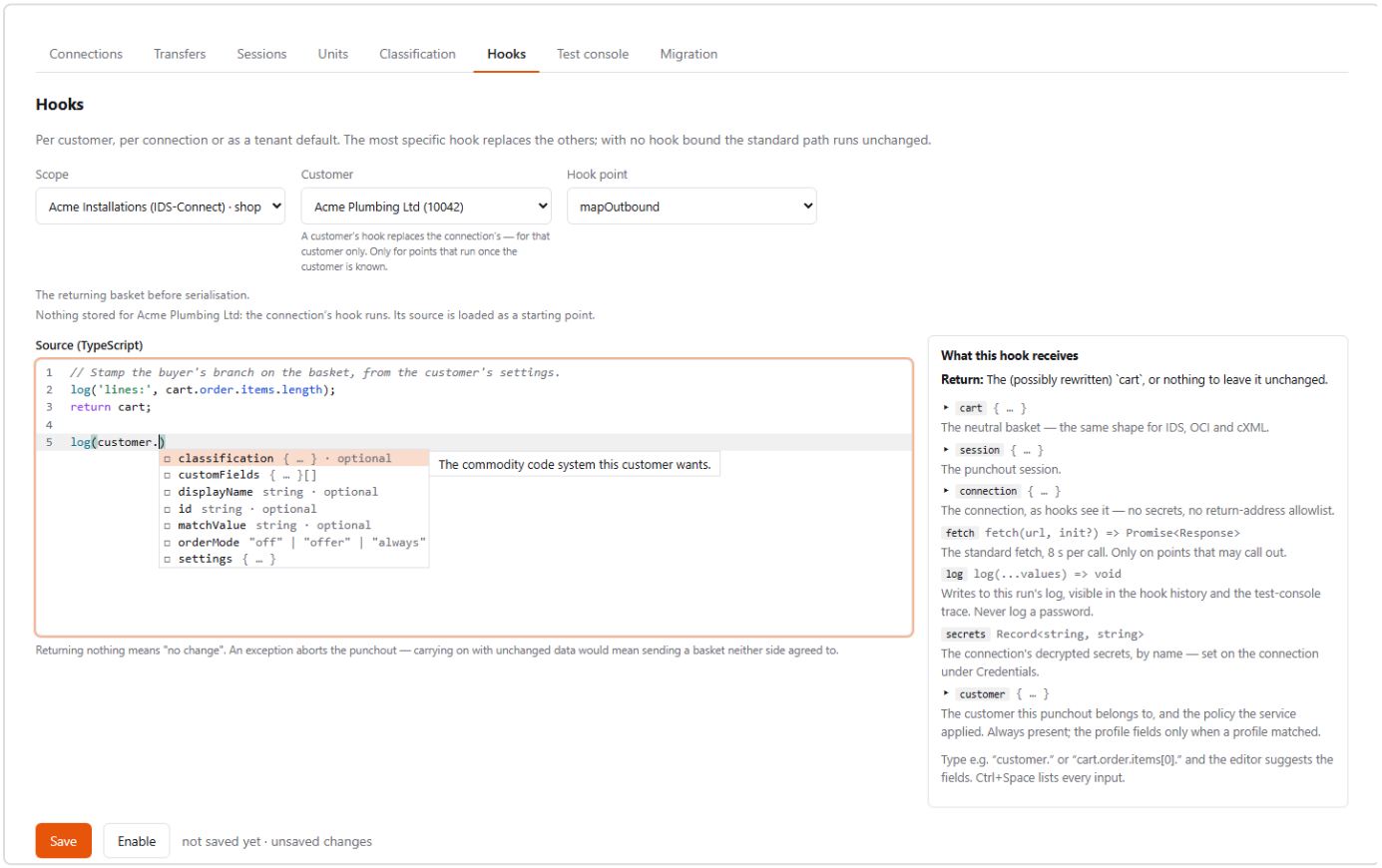
5.1 Choose where the hook runs

A hook belongs to the tenant, one connection, or one customer of it. The most specific one replaces the others.



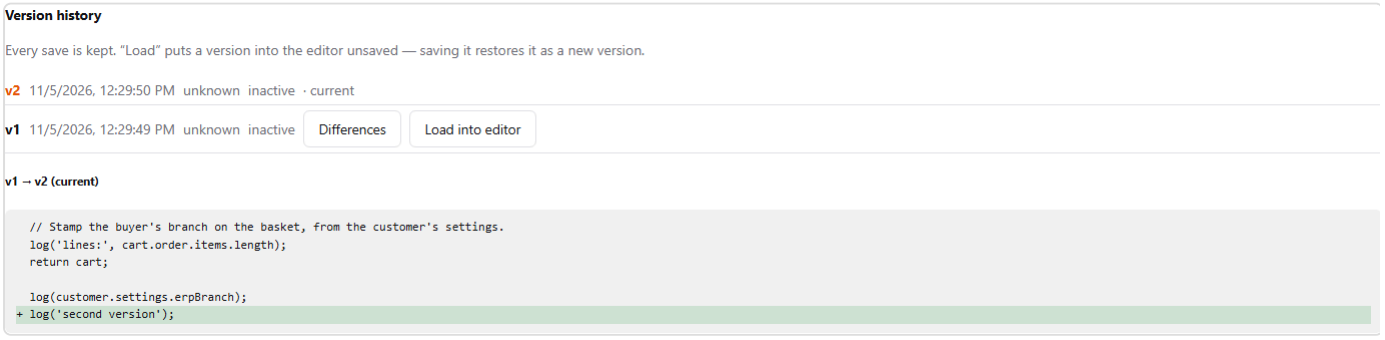
5.2 Let the editor show what the inputs hold

Type a dot after any input and the editor lists its fields with their types — including this customer's own settings. The reference beside it says what the hook must return.



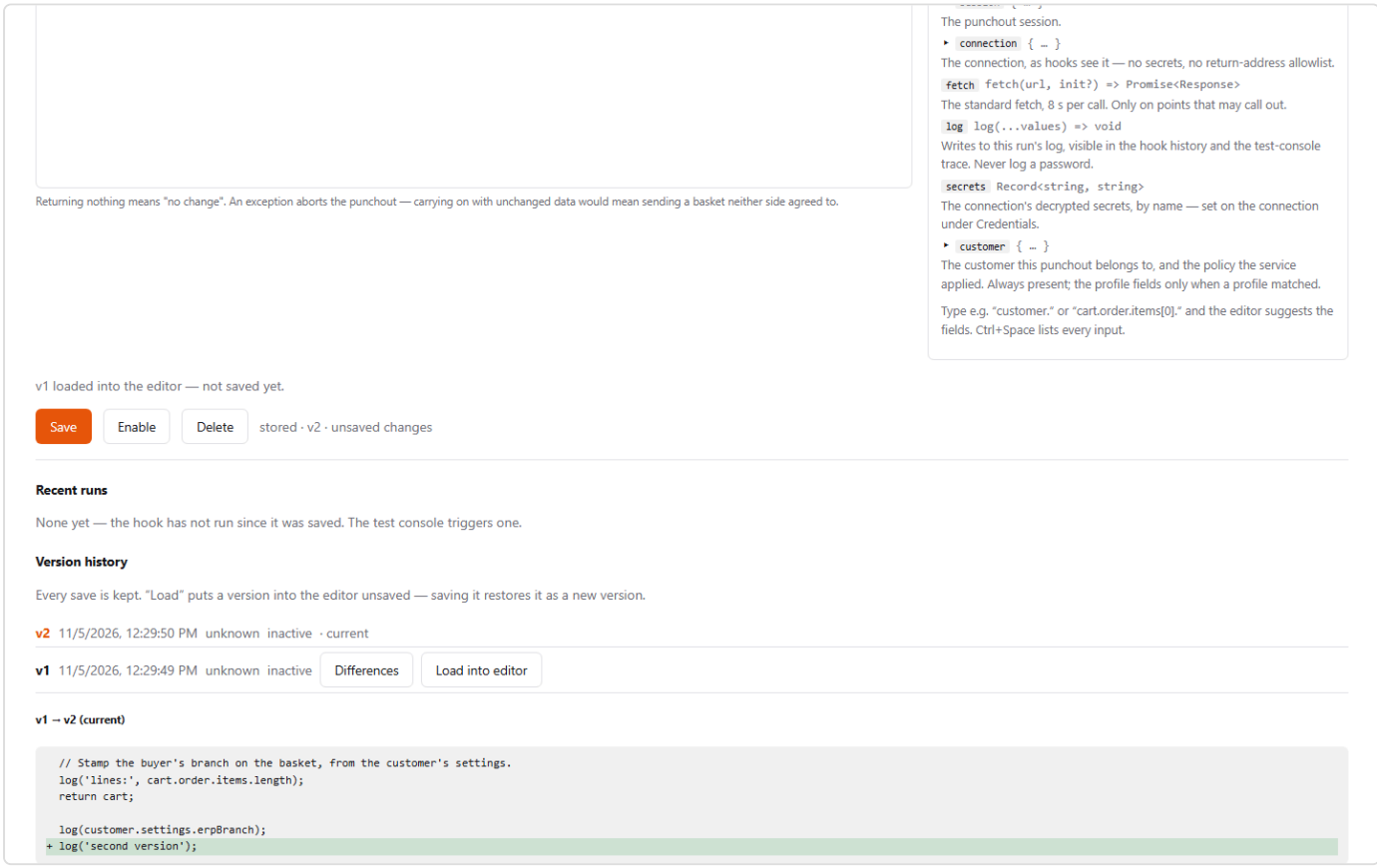
5.3 Compare with an earlier version

Every save is kept with its author and time. The difference to the current version is one click away.



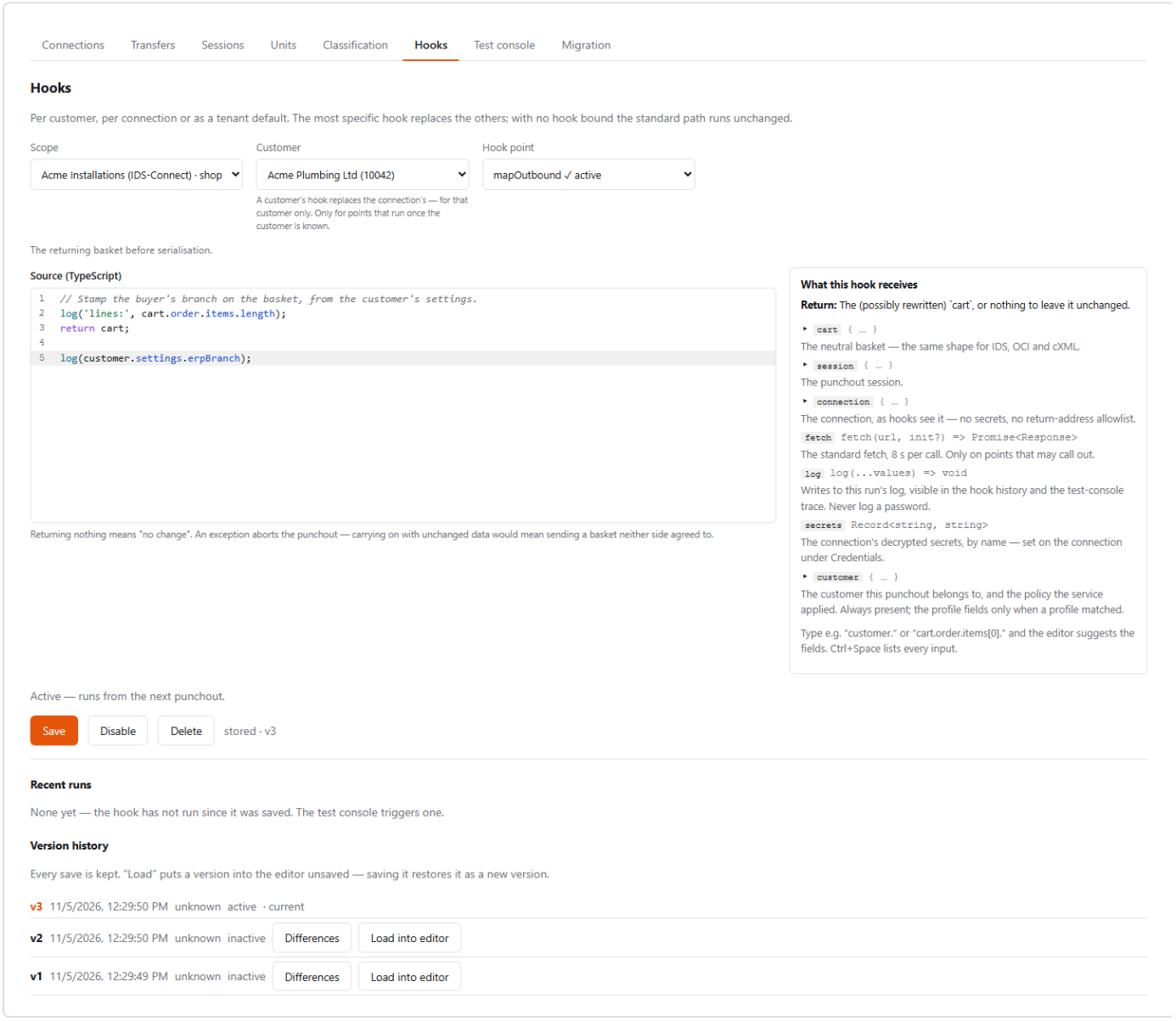
5.4 Load an old version back into the editor

The old code comes back unsaved, so it can be read before it goes live. Saving it restores it as a new version — history is never rewritten.



5.5 Switch it on

Saving never puts a hook into live punchouts, so a half-written function cannot reach a buyer. Enable does — and saves what is in the editor first, so what goes live is what you read. Disable takes it out again and keeps it; Delete removes it, and the connection's or the tenant's hook applies again.

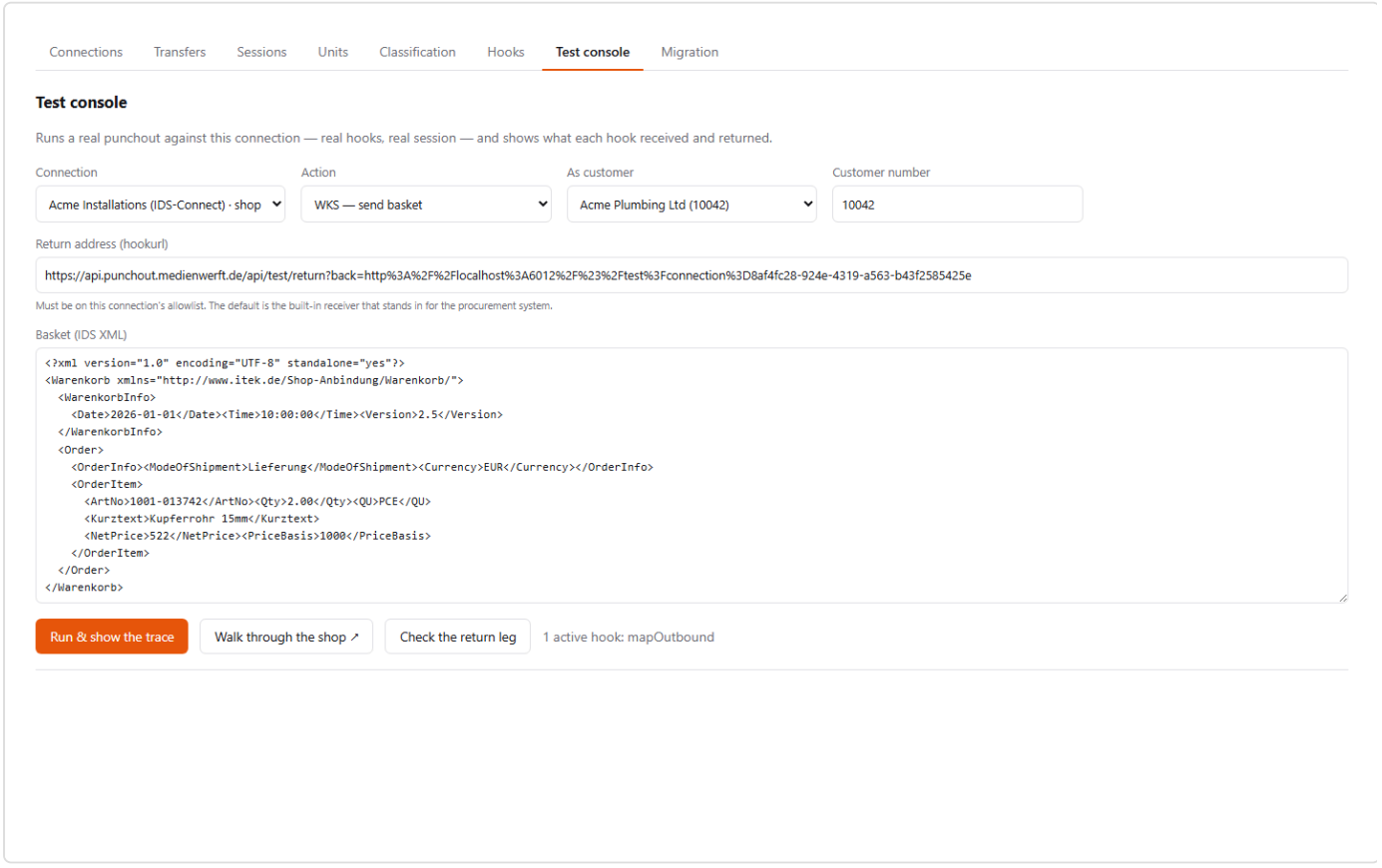


6. Run a test punchout

The test console runs a real punchout against a connection — real hooks, a real session — without a buyer's system. It is how a new connection, a new customer or a changed hook is tried before a buyer finds out.

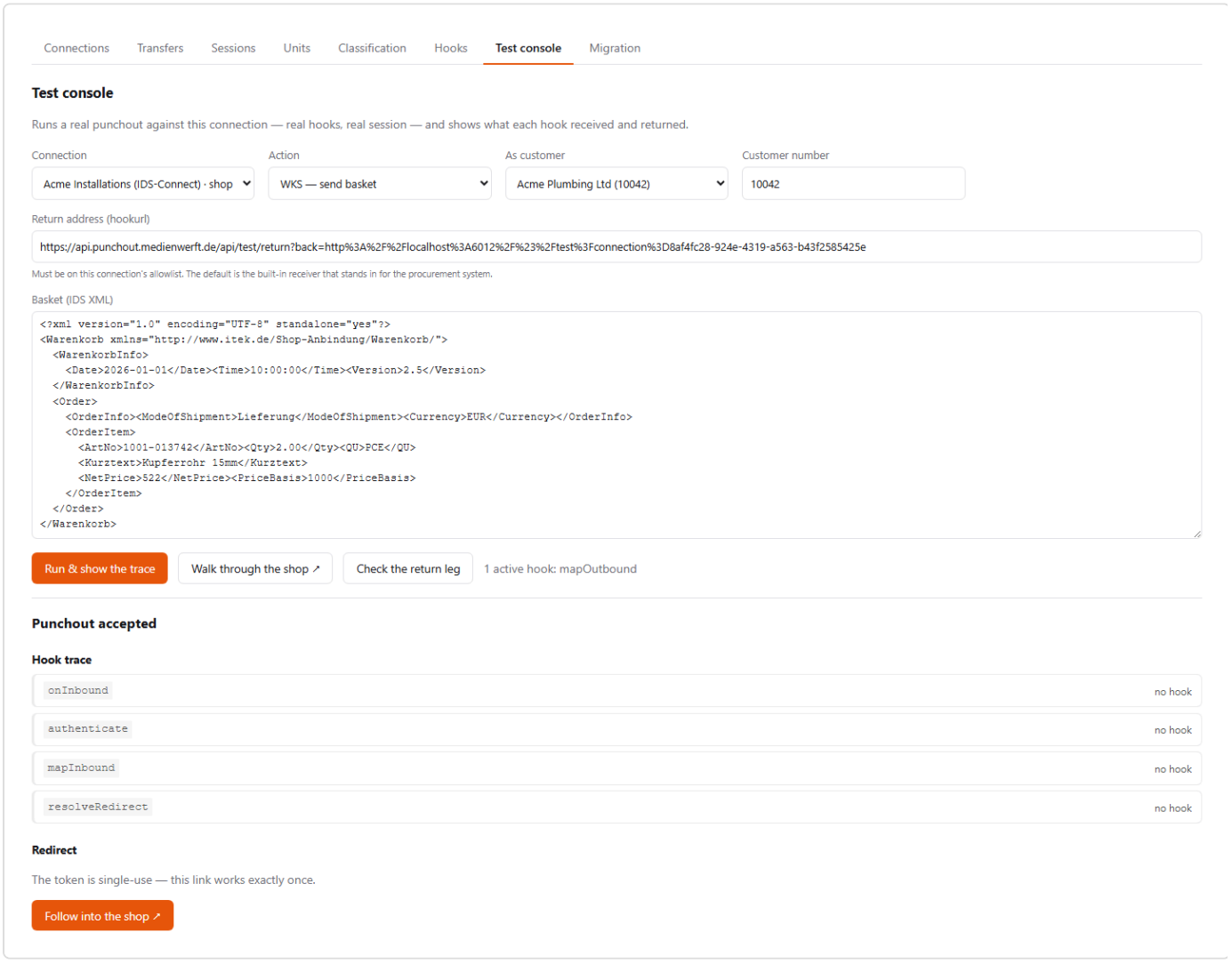
6.1 Choose the connection and the customer

Choosing a customer fills in what its software would send — here the customer number — so the run gets that customer's rules.



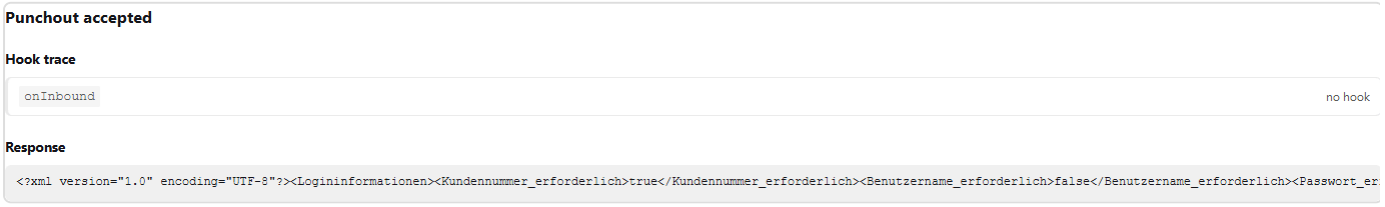
6.2 Run it and read the trace

Each hook point in order: whether a hook ran, which scope it came from, what it received and what it returned.



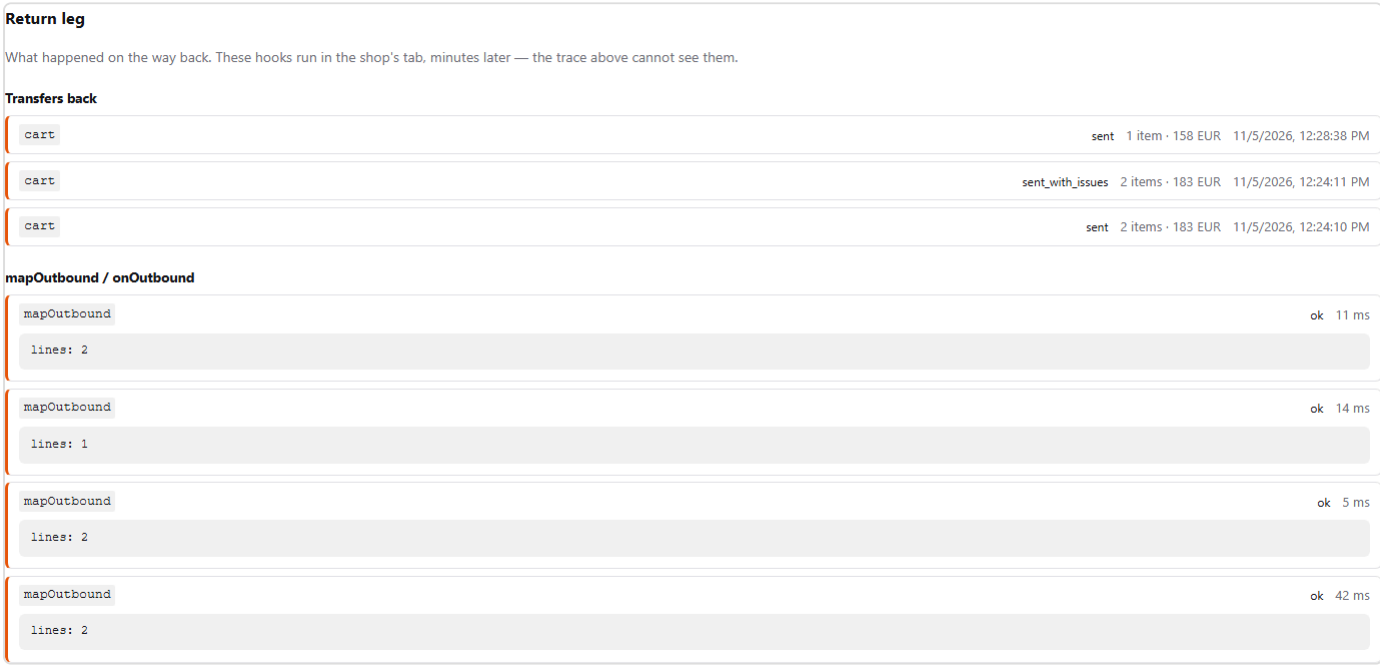
6.3 Try what else the buyer's software can ask

Action lists every request the protocol knows besides a basket — an article search, a product deeplink, the interface version, or, as here, the login information, which answers from the connection's login requirements. An action that needs a hook says so in the list.



6.4 Walk through the shop, then check the way back

Walk through the shop opens your shop in a new tab, exactly as the buyer would arrive. Once the basket was sent back, Check the return leg shows what went home and the hooks that built it — they run in the shop's tab, so the trace above cannot see them.



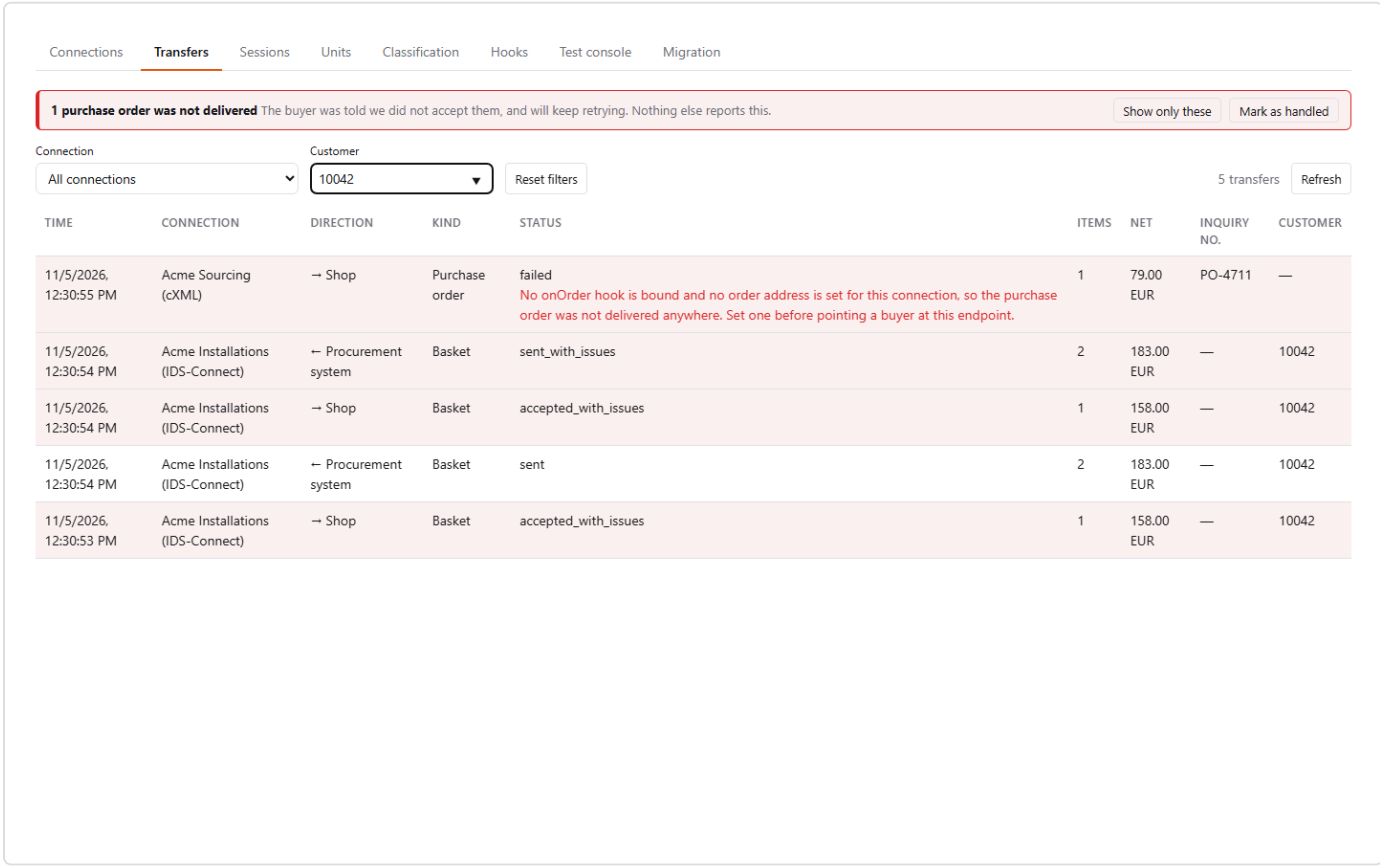
For developers: Developer Guide, 10. Testing your integration

7. Read a transfer

Every basket that arrives or goes back is a transfer. Its detail shows the document itself (when the connection keeps payloads), the hooks that ran for it in the version that ran, and lets you recalculate it against today's hooks.

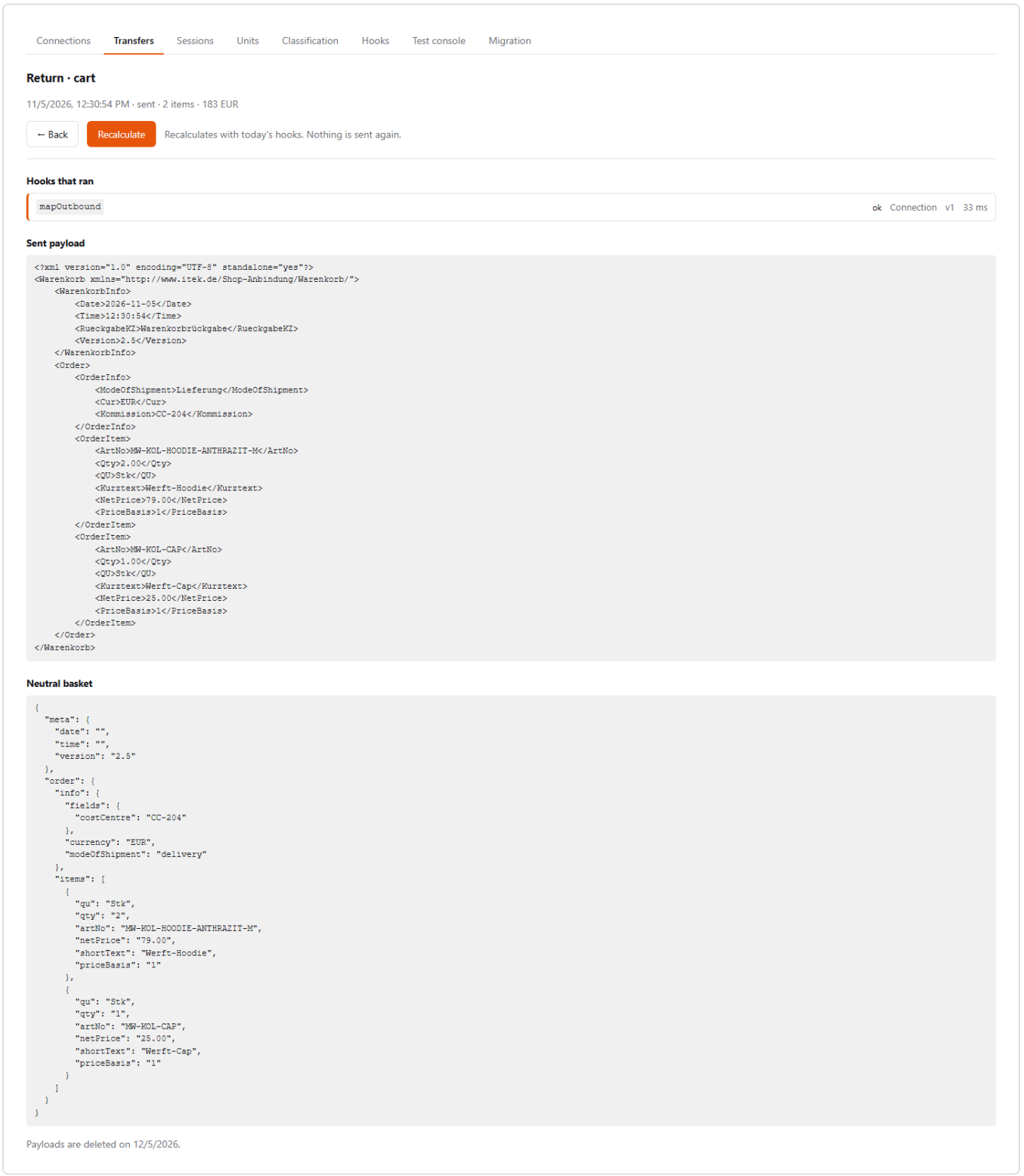
7.1 Filter the list to one customer

Filter by connection, by customer, or both. Without a connection filter, each row says which connection it came through.



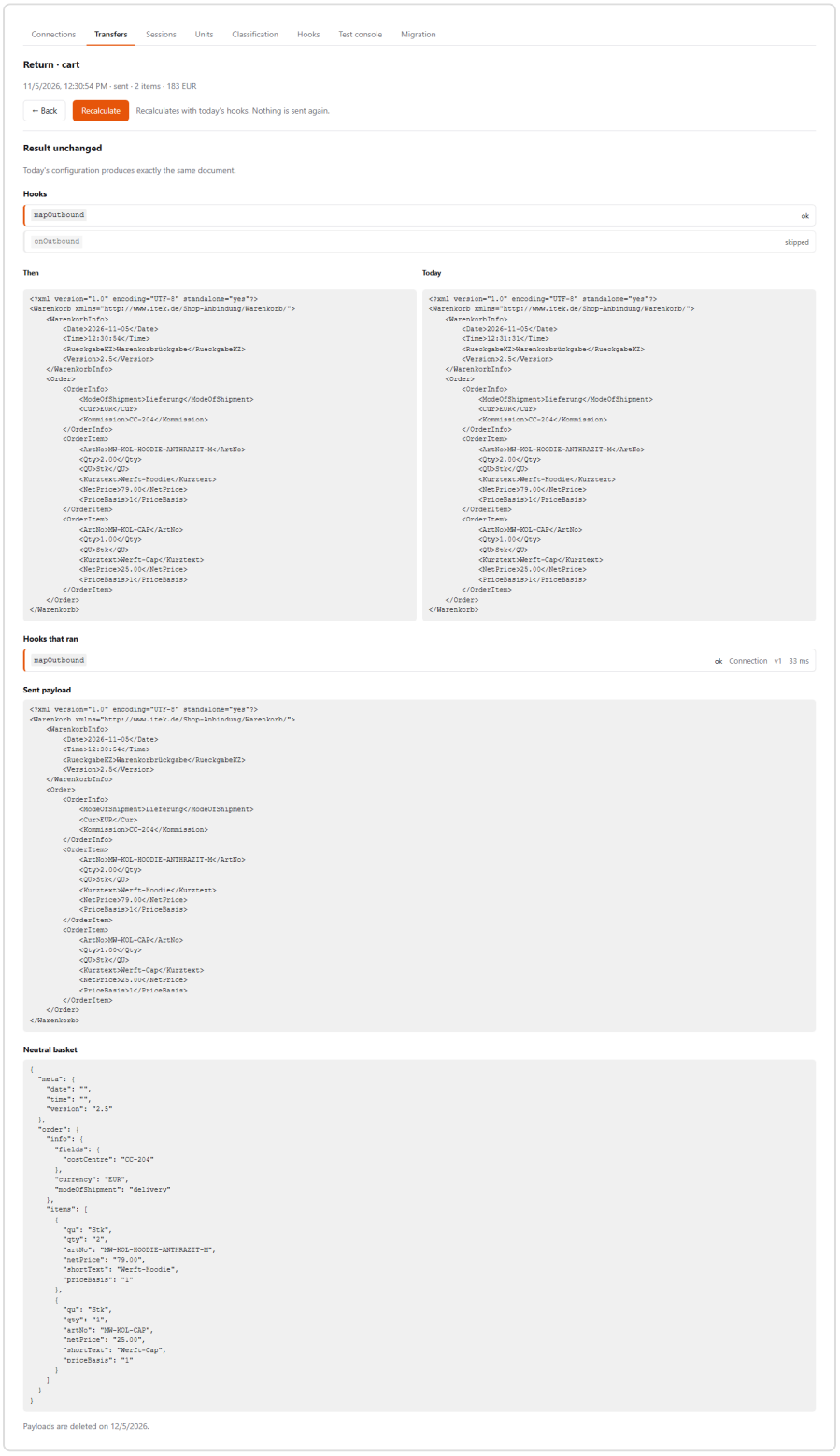
7.2 Open a transfer and see the hooks that ran

The hooks are listed in the order they ran, each with its scope, version, duration and log lines.



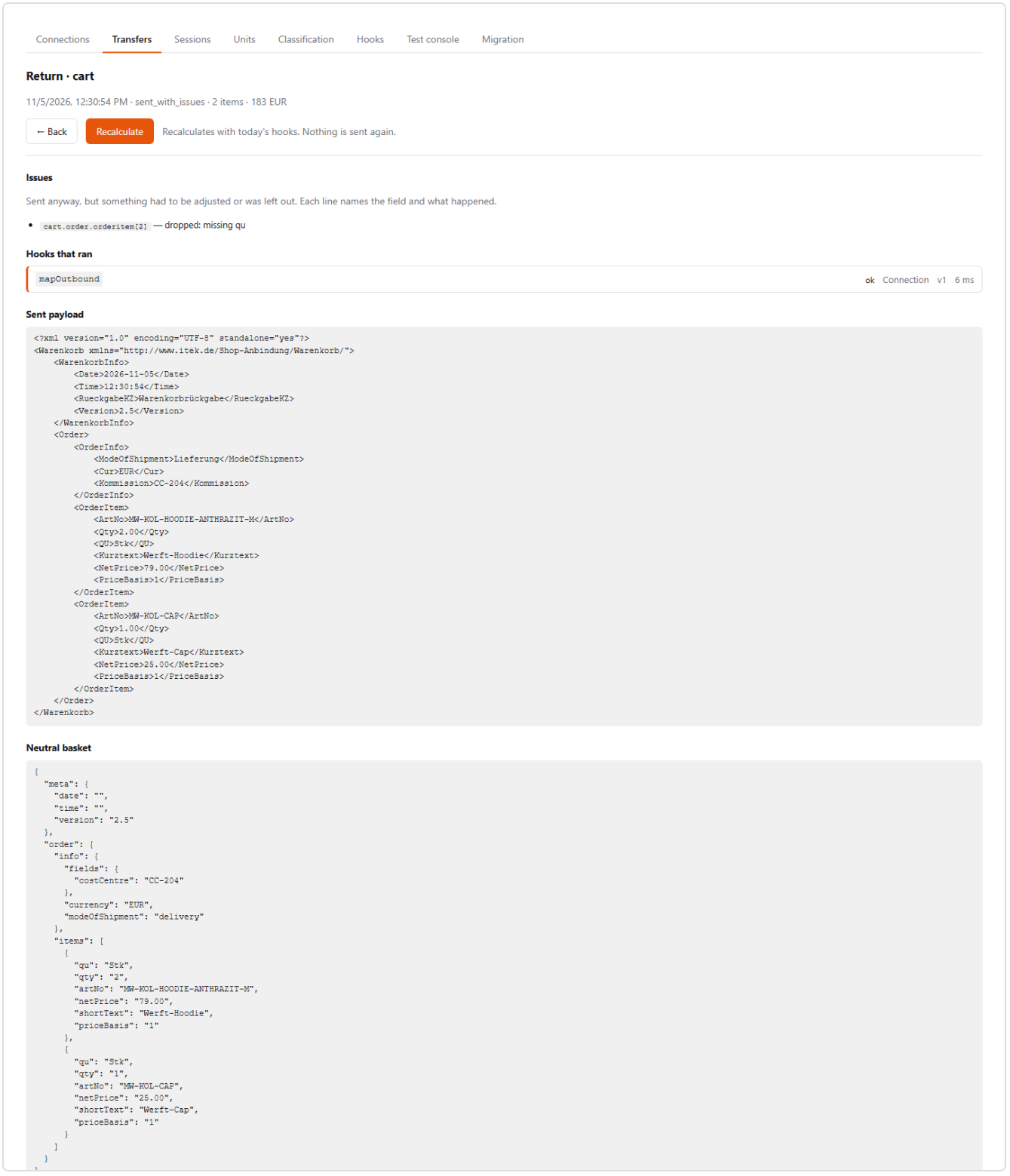
7.3 Recalculate against today’s hooks

Nothing is sent again. The result says whether today’s hooks and settings would produce the same document — here they would, so a fix can be checked before real traffic arrives.



7.4 See why a transfer went back with issues

A transfer listed as accepted_with_issues or sent_with_issues went through, but something had to be adjusted or left out; its Issues section names the field and what happened. Here the shop sent a line without a unit, so it was left out. Other typical causes on the way back: a custom field cut to its maximum length, or, for IDS-Connect, a basket sent as an order while no onOrder hook or order address receives it. Transfers from before this section existed show only the status.

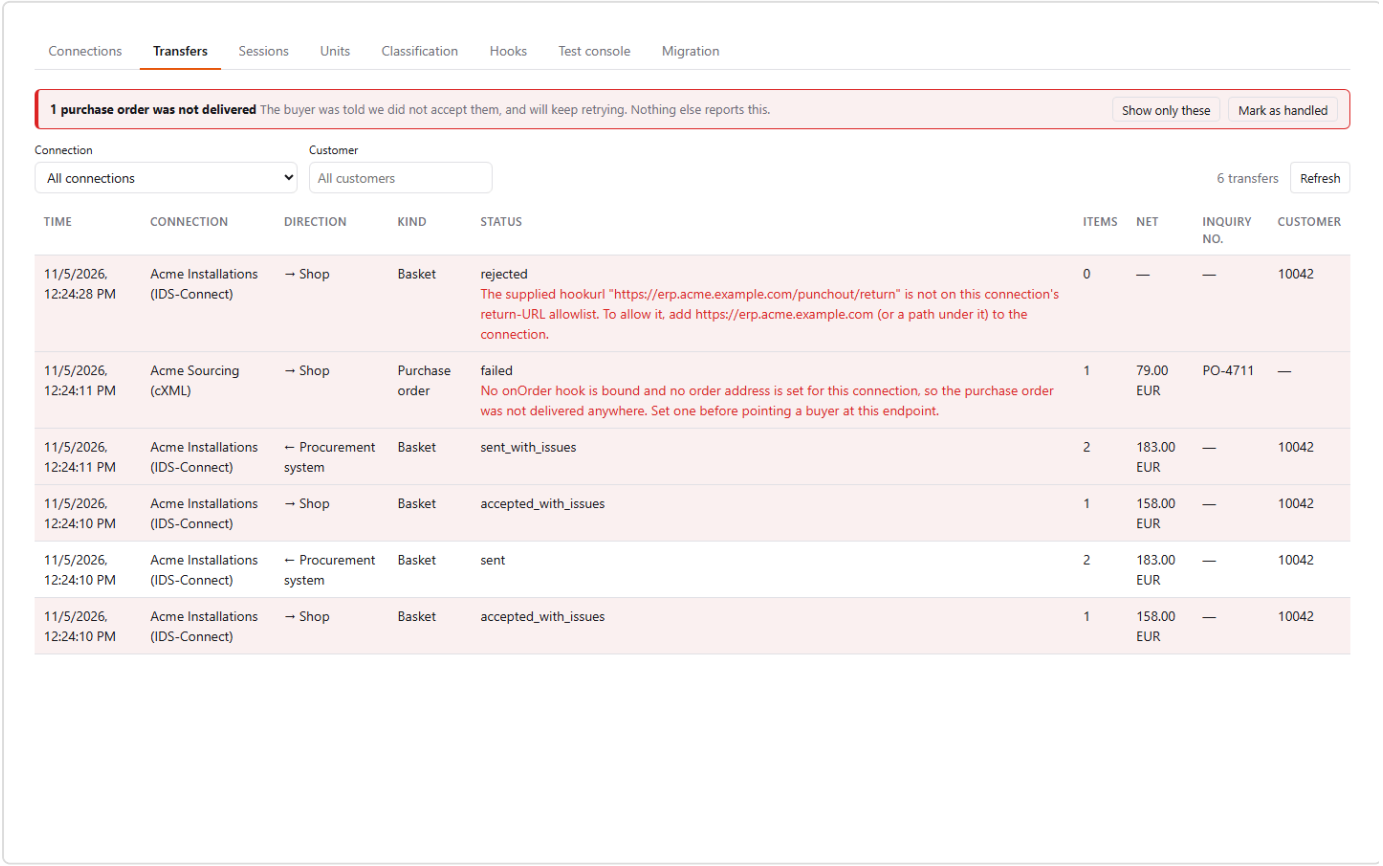


8. Handle a failed order

When a buyer's purchase order cannot be delivered to your backend, the buyer is told and will retry — but nobody on your side hears about it. That is what the red strip above the transfer list is for.

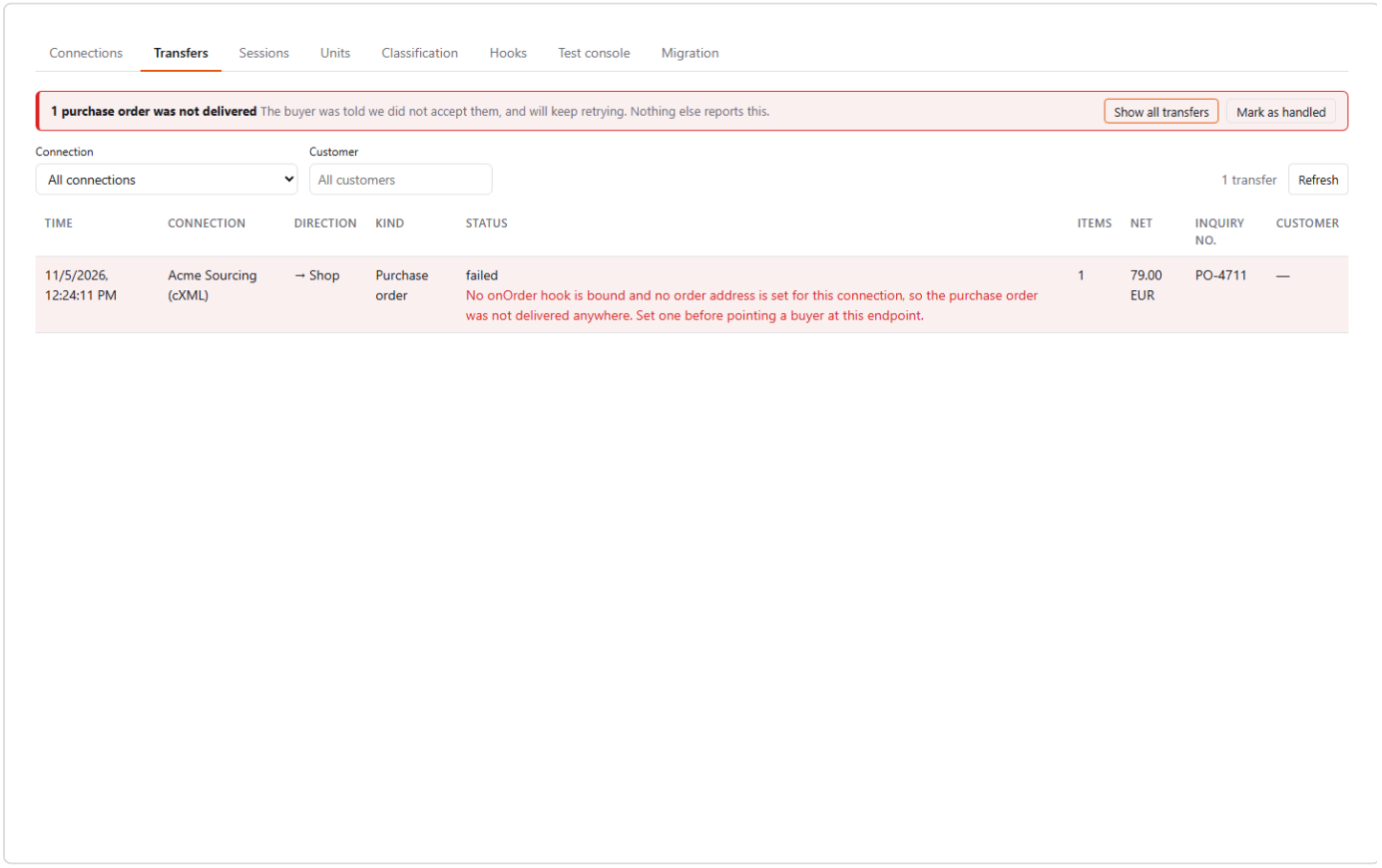
8.1 See the strip

It counts undelivered orders of the last fourteen days, whether or not they are on the current page.



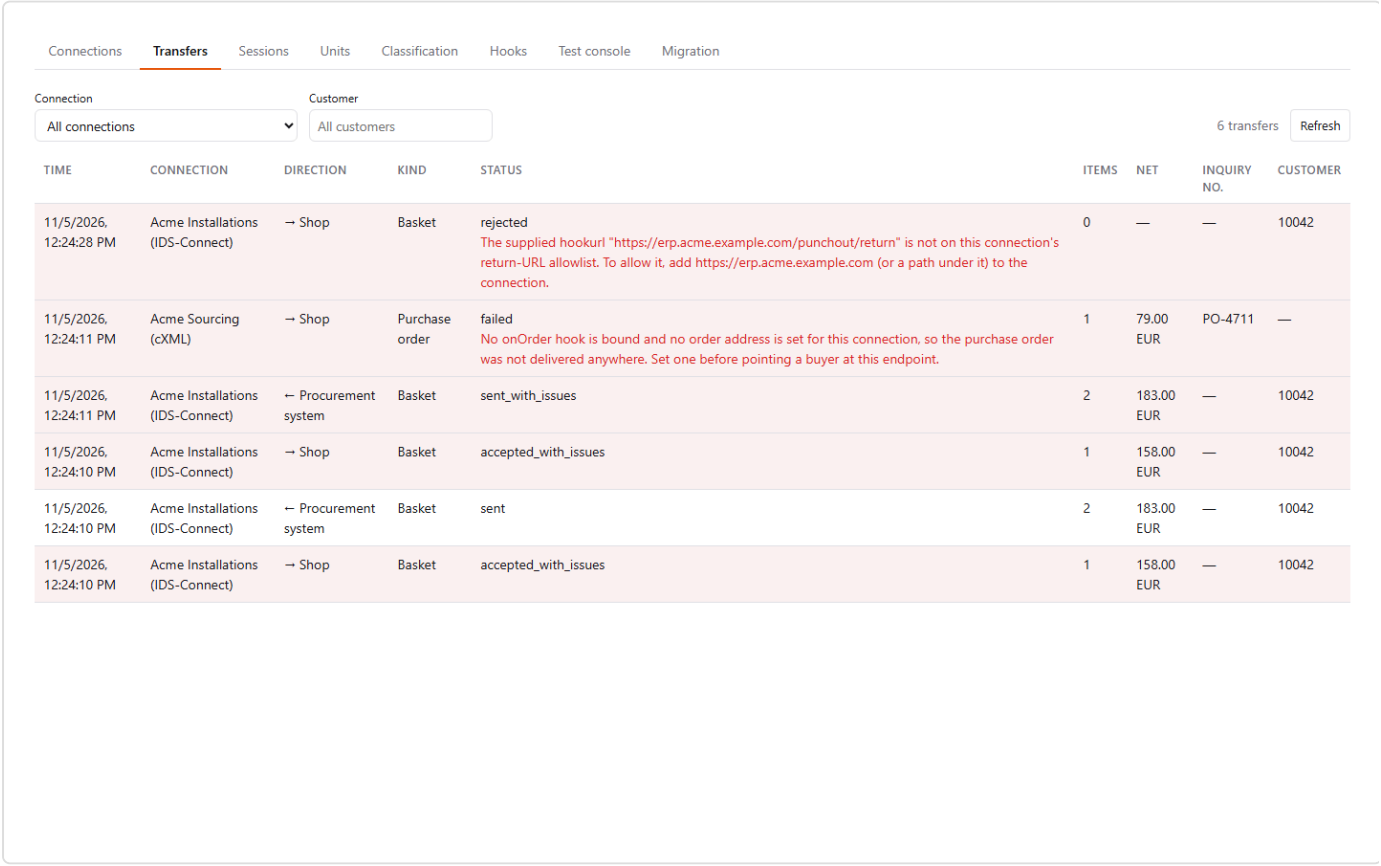
8.2 Show only those orders

Open one to see what the buyer sent and why it was refused.



8.3 Mark them as handled

For the whole team: the strip stays away until another order fails. The transfers themselves are unchanged.



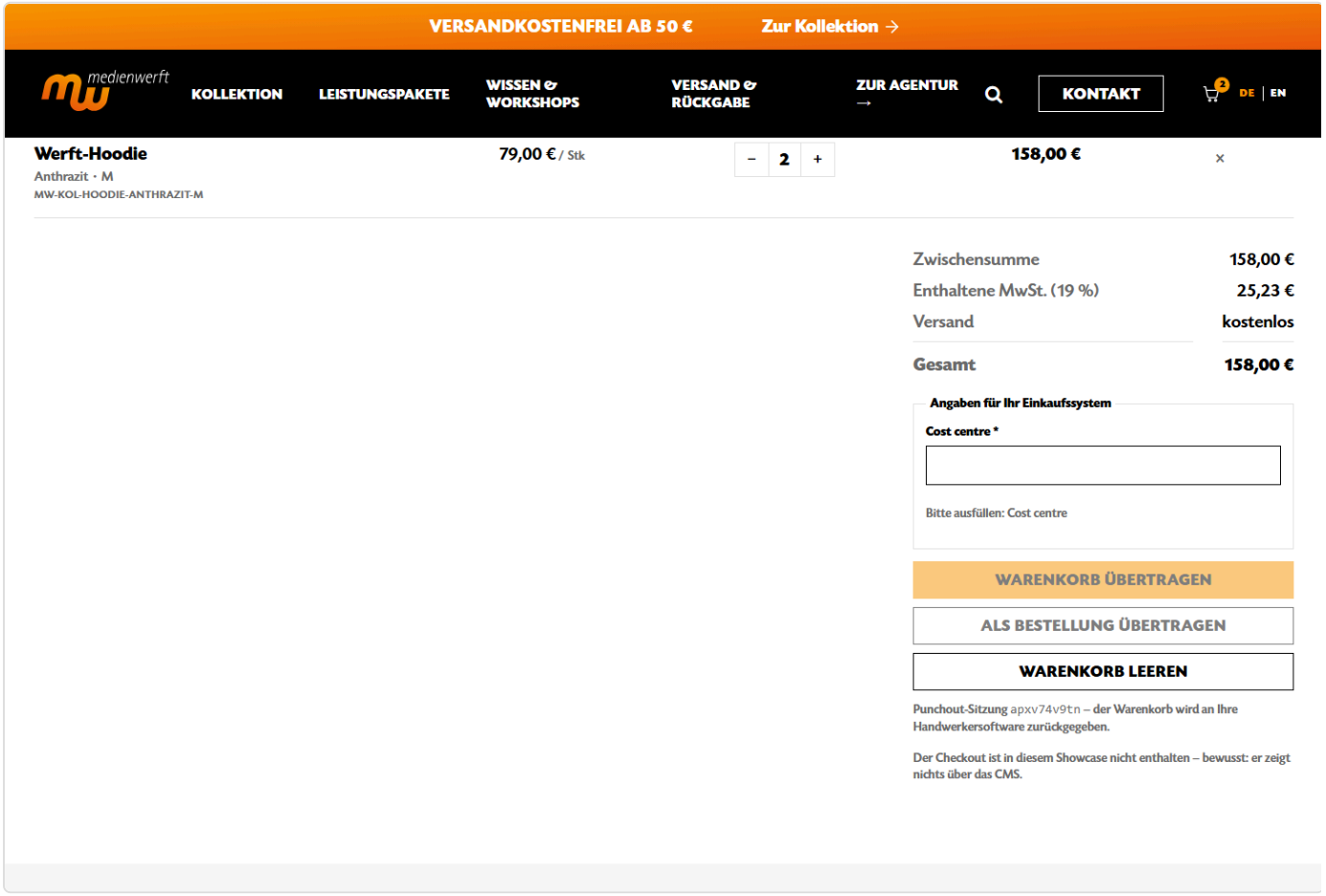
For developers: Developer Guide, 6. Orders and refusals

9. A punchout from the shop's side

What a buyer sees: they arrive from their procurement system in your shop, their basket already loaded, and the checkout is replaced by a transfer back. The shop shows exactly the buttons and fields this customer needs.

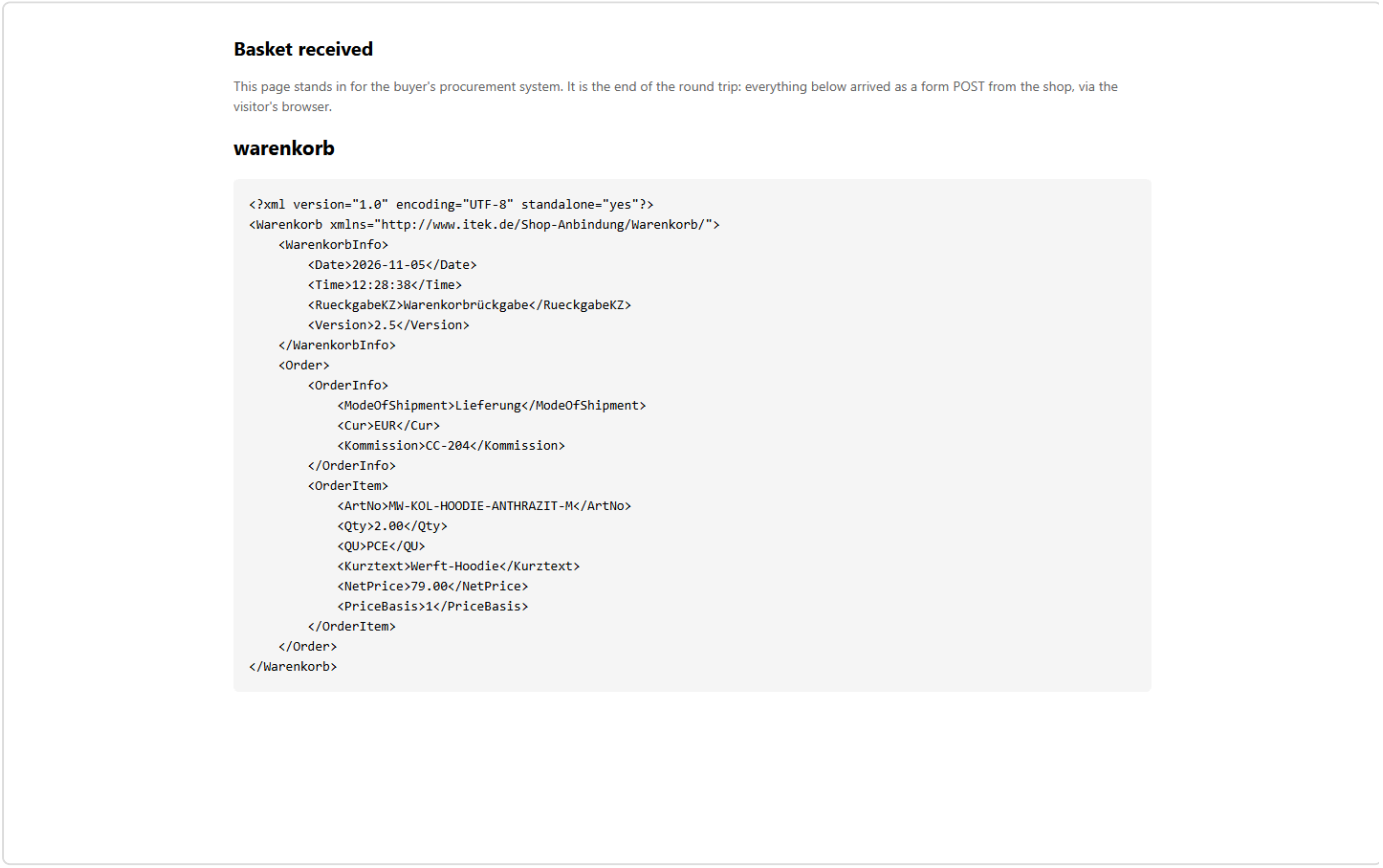
9.1 Arrive in the shop with the basket

The checkout is replaced by the transfer. This customer may transfer as a basket or as an order, and must give a cost centre first.



9.2 Transfer the basket back

The basket goes back in the buyer's protocol — here the return inspector stands in for the buyer's software and shows what arrived.



Basket received

This page stands in for the buyer's procurement system. It is the end of the round trip: everything below arrived as a form POST from the shop, via the visitor's browser.

warenkorb


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Warenkorb xmlns="http://www.itek.de/Shop-Anbindung/Warenkorb/">
    <WarenkorbInfo>
        <Date>2026-11-05</Date>
        <Time>12:28:38</Time>
        <RueckgabeKZ>Warenkorbrückgabe</RueckgabeKZ>
        <Version>2.5</Version>
    </WarenkorbInfo>
    <Order>
        <OrderInfo>
            <ModeOfShipment>Lieferung</ModeOfShipment>
            <Cur>EUR</Cur>
            <Kommission>CC-204</Kommission>
        </OrderInfo>
        <OrderItem>
            <ArtNo>MW-KOL-HOODIE-ANTHRAZIT-M</ArtNo>
            <Qty>2.00</Qty>
            <QU>PCE</QU>
            <Kurztext>Werft-Hoodie</Kurztext>
            <NetPrice>79.00</NetPrice>
            <PriceBasis>1</PriceBasis>
        </OrderItem>
    </Order>
</Warenkorb>
```

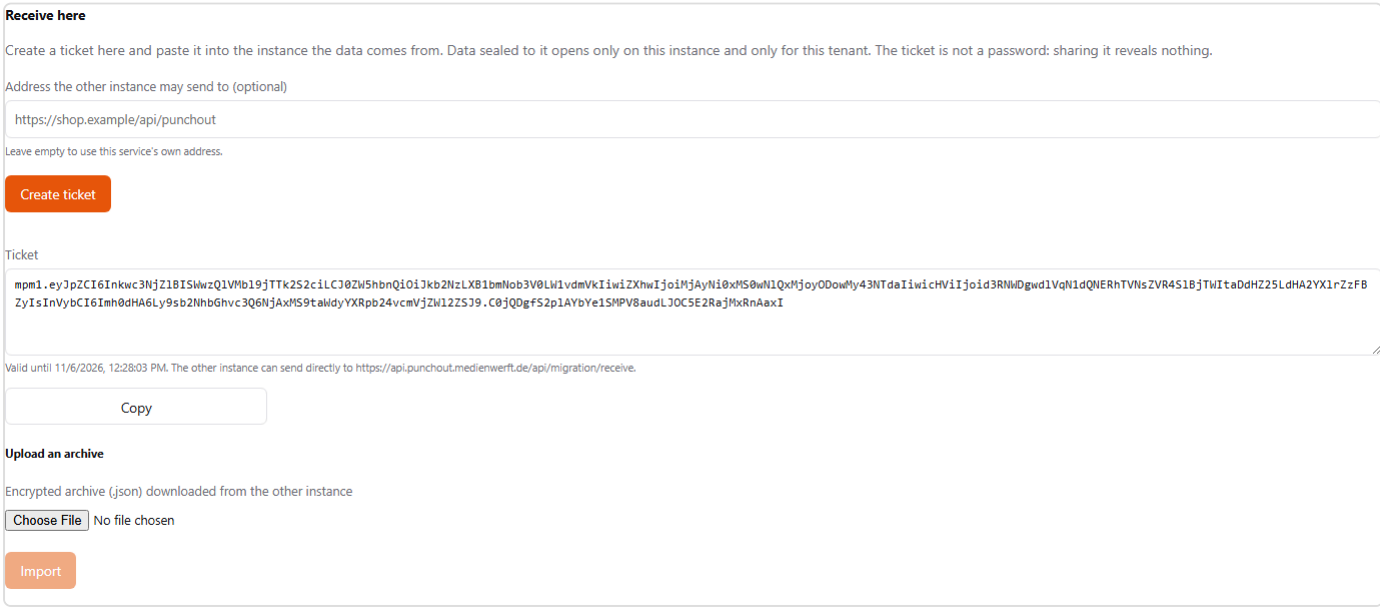
For developers: Developer Guide, 2. Picking up the basket · Developer Guide, 5. Sending the basket back

10. Move to another instance

Moving from the hosted service into Emporix, or back, takes the whole configuration along: connections with their credentials, customers, hooks, units and commodity codes. The connection addresses stay the same, so buyers keep their setup.

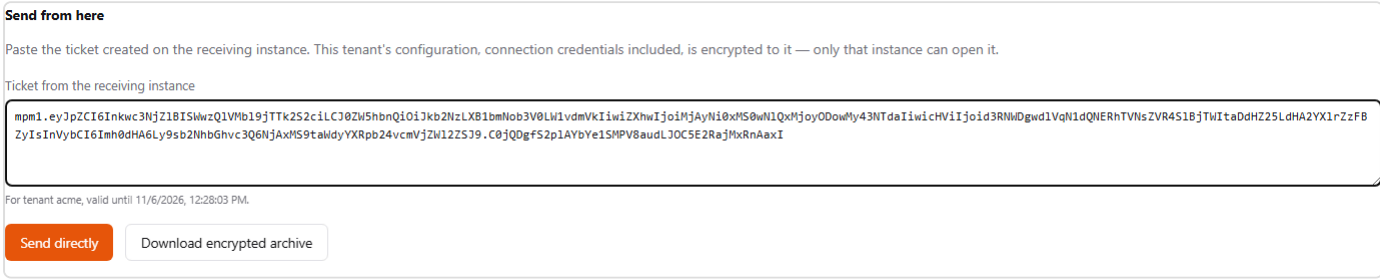
10.1 Create a ticket where the configuration should go

On the instance that receives, in the tenant that should hold the configuration — it must be empty. The ticket is valid for 24 hours and is not a password: it can only lock data for this instance, never open it. Inside Emporix, enter the shop's punchout relay as the address to allow a direct transfer.



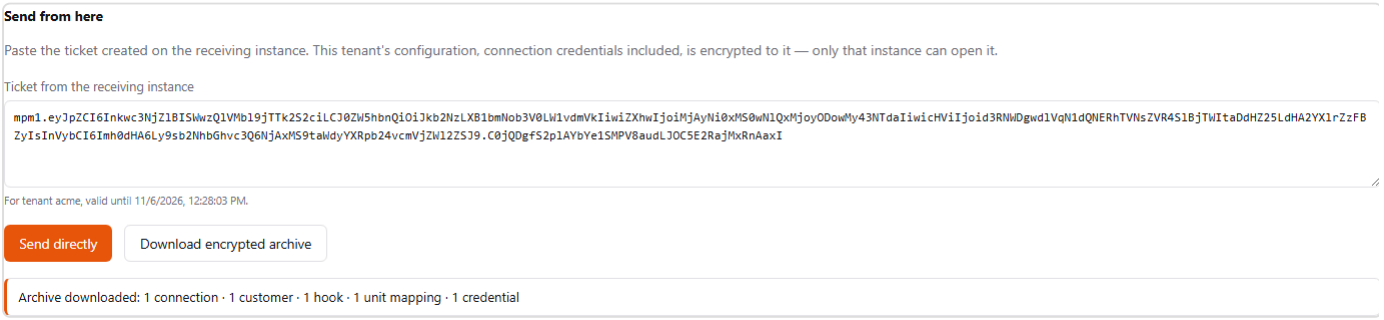
10.2 Paste it where the configuration is now

The dashboard shows which tenant the ticket is for and until when. Send directly when the ticket names an address; otherwise download the archive. Either way it is encrypted for the receiving instance, connection credentials included.



10.3 Download the encrypted archive

The file opens only on the instance that issued the ticket. Keep it as you would the credentials it carries.



10.4 Upload and import it

Everything is checked as if entered by hand. If anything is refused, nothing stays behind and the import can be repeated. Transfers, sessions and the hook history stay on the old instance; switch the old connections off once buyers arrive here.

