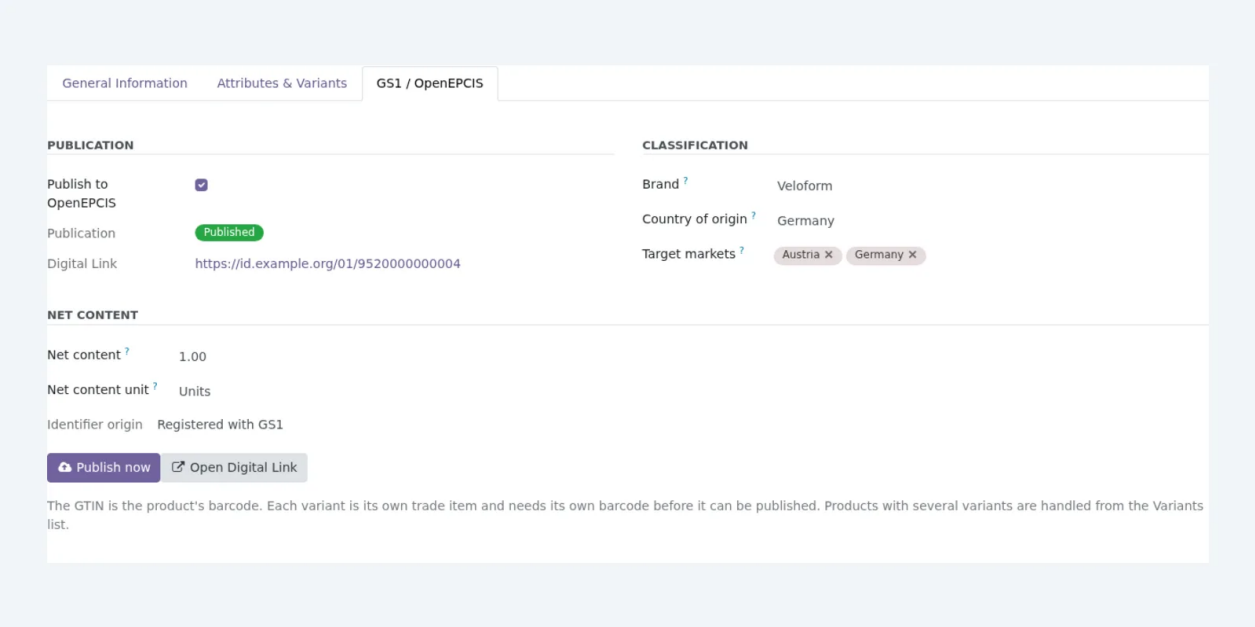
OpenEPCIS



Your ERP Already Knows: OpenEPCIS for Odoo

Seven open-source addons publish Odoo's products, contacts, lots and serial numbers to a GS1 Digital Link and report what happens to them as EPCIS 2.0 events. Nobody types a passport field twice, and the warehouse never waits for a repository.

Sven Böckelmann · 9 October 2026 · 8 min read

odoo • erp • connectors • epcis • digital-link • dpp • gs1 • open-source

openepcis.io/blog/odoo-connector

Contents

Your ERP Already Knows: OpenEPCIS for Odoo **3**
What it does, in one paragraph . . . 3
Seven addons, because Odoo is modular and so are we . . . 3
The decisions worth explaining . . . 4
What it looks like . . . 4
And if your system is not Odoo . . . 5
Try it . . . 5

 © benelog GmbH & Co. KG

Your ERP Already Knows: OpenEPCIS for Odoo

Every Digital Product Passport conversation reaches the same moment. The data model is agreed, the resolver is up, the passport page renders, and then somebody asks who is going to fill in the brand, the net weight, the country of origin and the manufacturer for eleven thousand products. The answer, usually, is a spreadsheet and an intern. Both are wrong, because the data is already in the ERP. It has been there for years. It is what the company orders, picks and invoices with.

So we wrote a connector that lives inside the ERP, and today it is public: openepcis/openepcis-odoo on GitHub, seven addons for Odoo 18 and 19, LGPL-3.

What it does, in one paragraph

Tick *Publish to OpenEPCIS* on a product, save, and within five minutes the product is a document in the catalog behind a GS1-conformant Digital Link resolver, with its Digital Link and a QR code back on the Odoo form and on a printable label. Company contacts go out the same way, as GS1 organizations. Lots and serial numbers follow their product down to the instance level — one document per batch, one per unit. And once the goods start moving, every validated transfer, every pallet packed, every return and every manufacturing order becomes an EPCIS 2.0 event in the repository. A scanned serial number can then answer both questions a passport exists to answer: what is this, and what became of it.

Seven addons, because Odoo is modular and so are we

The connector is cut along Odoo's own module boundaries, and that is a deliberate choice rather than an accident of history.

The base addon, `openepcis_connector`, depends on `product` and nothing else. It installs on an Odoo that has no warehouse at all. Everything that needs another Odoo app is a *bridge* that installs itself when that app is present:

Addon	Needs	Adds
`openepcis_connector`	`product`	Products and contacts as GS1 master data, *Draw GTIN*, the field mapping, a first-load wizard
`openepcis_connector_stock`	Inventory	Lots as `/10/<lot>`, serial numbers as `/21/<serial>`
`openepcis_connector_product_expiry`	Expiration Dates	The expiry dates on a lot under their GS1 names
`openepcis_connector_events`	the two above	Transfers, packing, returns and withdrawals as EPCIS events; an inbox for partners' events
`openepcis_connector_events_mrp`	Manufacturing	A manufacturing order as one `TransformationEvent`
`openepcis_connector_events_pos`	Point of Sale	A till's orders read as sales, not shipments
`auth_oauth_end_session`	`auth_oauth`	Logging out of Odoo also logs out of Keycloak

A database never carries code for an app it does not run, and nobody has to decide which bridges to install. Each addon has a page of its own in the documentation, and the *Learn More* button on its tile in Odoo's app list opens exactly that page.

© benelog GmbH & Co. KG

The decisions worth explaining

A connector is a thin thing to describe and a thick thing to get right. These are the places where the right answer was not the obvious one.

Saving never waits for the network. A product is queued on save and published in the background by a scheduled action. Validating a transfer writes a row into an outbox and returns. The person pressing *Validate* can do nothing about a repository that is down, and blocking them on it turns somebody else's outage into a stopped loading bay.

The outbox does not believe a 202. EPCIS capture is asynchronous: the repository answers *accepted*, validates afterwards, and may still refuse. A queue that deletes its row on the 202 reports success for events that were thrown away minutes later. So a row stays until the repository confirms it *captured* the event, and only then says what really happened.

The field mapping is data, not code. Which Odoo field feeds which GS1 term is a list of records an administrator edits, because no two Odoo databases keep a brand in the same place. The shipped rows are starting points, and an upgrade does not undo your edits.

Drawing a GTIN and registering it are two acts. Registration with GS1 cannot be undone — an identifier with no product data behind it can be neither deleted nor deactivated. So *Draw GTIN* holds a number from your company prefix and registers it only when the product is saved. A number drawn for a form somebody then abandons is handed back instead of burnt.

The form tells you what a registry wants before you publish, not after. The resolver reports which terms each publishing channel insists on, and the product form keeps that list current as you type. It informs; it does not block. A passport is filled in over time and by several people, and an unmet requirement must never hold data hostage.

No password in the ERP. Odoo holds an OIDC offline token, mints a short-lived access token from it for every call, and discovers the Keycloak realm from the resolver's own metadata (RFC 9728). Access is withdrawn in Keycloak, not in Odoo.

A lot is not a serial number. A serial identifies one unit and goes into an event's `epcList`. A lot identifies a *set* and goes into the `quantityList`. Untracked goods are a set too — the trade item itself — which is why a warehouse that tracks nothing still produces useful events.

The read point is not optional. EPCIS allows an event without one; we do not. An event that says something happened without saying where is half an answer, and half answers are worse than none because they look complete. A GLN goes on the warehouse or the loading bay, every location underneath inherits it, and a transfer between locations with no GLN is reported in the chatter instead of the repository.

The inbox never moves stock. Partners' events about our goods are shown on the lot and the package; they are observations, not documents, and posting an observation into a valued stock would make the closing depend on a third party's data quality. The one exception is double-locked: an incoming event may validate a transfer that is already open, reserved and waiting for exactly that confirmation, and only if the operation type allows it *and* the partner is trusted *and* somebody has turned off observe-only after reading a week of the rehearsal log.

What it looks like

The documentation walks through one warehouse day: sixty kilos of coffee arrive, forty-eight go into two cases, the cases onto a pallet. One case comes off and ships; two kilos come back a week later. The other case is emptied, ten kilos are ground into forty bags of a different article, and the delivery turns out never to have happened. Nine screens, twelve events, each copied from the outbox of the instance in the screenshots — including the withdrawal, because an event cannot be pulled out of a repository and EPCIS has a proper way to say "this did not occur".

© benelog GmbH & Co. KG

The screenshots are not taken by hand. A script drives a headless browser through the same screens, so they can be renewed when the addon changes instead of quietly going stale. Every identifier is in the GS1 952 test range, every host is a placeholder, and the instance publishes to nobody.

And if your system is not Odoo

The line that matters is not PIM versus ERP. It is where the connector runs. Odoo is the one system where the connector lives *inside* the host: the addons are installed in Odoo, queue on save, and publish from there. For every other system we support — the UnoPim PIM, and the ERPNext and metasfresh ERPs — the connector is a service of ours that polls the system's REST API, maps what it finds onto GS1 terms and publishes it. Nothing has to be installed in the host at all.

What each route publishes follows from what the host knows. A PIM knows the model: UnoPim gives you products and batches. An ERP knows the individual goods as well: ERPNext gives products, batches, serial numbers and parties, metasfresh products and parties. Odoo goes one step further because it also runs the warehouse — it is the only route today that reports movements, packing, production and sales as events.

For UnoPim there is an optional PHP package that puts a panel above the product form — what the platform currently says about this product, draw a number, deposit one you already have. That package is open source: openepcis/openepcis-unopim, MIT. What the routes share, and where they differ, is in how a connector works.

Try it

```
git clone https://github.com/openepcis/openepcis-odoo.git
cd openepcis-odoo
docker compose up -d
# http://localhost:8069 — create a database, install "OpenEPCIS Connector"
```

Then *Settings* → *General Settings* → *OpenEPCIS*: the resolver's URL, and an offline token issued in the OpenEPCIS web interface.

One warning that the README repeats in bold, and so will this post. **There is no GS1 sandbox.** The platform's GS1 credentials are production credentials, and a registered identifier is registered for good. Test against a development deployment, with the operator's dry-run flag on, and with identifiers in the 952 range GS1 reserves for exactly that.

The code is LGPL-3, the issues are open, and the two branches follow Odoo's convention — `18.0` for the long-term release, `19.0` for the newest. If your Odoo keeps a brand somewhere we did not expect, the mapping is yours to edit. If it keeps something we did not think of, tell us.

© benelog GmbH & Co. KG