Nalios EDI Connector

Complete user guide — understand and configure automated exchanges with your partners, without writing a single line of code

Webhook | FTP / SFTP | AS2 | XML · JSON · EDIFACT · Fixed-width · CSV

Written for a business profile — no development knowledge required

1. What is this module for?

Today, when a customer or supplier sends an order, an invoice or a delivery note, they don't necessarily send it by typing it into Odoo by hand. Many large companies (retailers, industrial groups) exchange their business documents as **standardized computer files**, sent automatically from one system to another.

This is called **EDI** — Electronic Data Interchange. The **Nalios EDI Connector** module is the "translator" that receives these files, understands their content, and automatically creates or updates the corresponding records in Odoo (a customer, an order, an invoice...) — without a human needing to retype anything.

It also works the other way: it can build an EDI file from Odoo data (for example, confirming a delivery) and send it automatically to the partner.

> 💡 **In one sentence**
>
> The module watches a file arrive, understands its content using rules you configured (no code), and turns it into orders, customers, invoices... directly in Odoo. And it can do the same thing in reverse.

Why it's useful

- **Zero re-typing**: orders received from a major customer land directly in Odoo, with their product lines already filled in.
- **No code to write**: everything is configured in the Odoo interface (which field goes where), even if the partner's format changes over time.
- **Full traceability**: every file received or sent is logged, with its status (success, error, pending) and can be replayed if something goes wrong.
- **Several ways to transport files**: a web link (webhook), a file server (FTP/SFTP), or the secure protocol used across the retail sector (AS2).

2. A short EDI glossary

Before going further, a few words will come up constantly. Understanding them now will make the rest of the guide much easier.

File format: how the content is written

The same document (an order, for example) can be written in several different ways — a bit like the same information can be said in French or in English. The module understands five "languages":

Format	What it looks like	Used by
XML	`<Order><Ref>12345</Ref></Order>`	Many ERPs and e-commerce sites
JSON	`{"order": {"ref": "12345"}}`	Modern web applications, APIs
EDIFACT	`BGM+220+12345+9'`	Retail, industry (the sector's historic standard)
Fixed-width	`HH05410000000001...` (each field always occupies the same columns)	Legacy "mainframe" formats from some retailers, still used today
CSV	`ref;customer;product;qty`	Spreadsheet exports, product/stock imports

The module can read and write all five — you choose the right format based on what your partner uses.

> 💡 **What exactly is fixed-width?**
> Unlike XML or CSV, there is **no separator** between fields: each piece of information always occupies the same columns, whatever its length (padded with spaces if needed). Each line usually starts with a short code stating "this is a header line" or "this is a product line" — this is the code the profile uses to know how to read the line (see section 5).

Channel: the means of transport

The **channel** is how the file travels between your partner and Odoo. It's not the content of the file, it's the "truck" carrying it:

- **Webhook**: the partner (or their system) calls a web address to drop off the file, like submitting an online form.
- **FTP / SFTP**: a shared folder where files are dropped, which Odoo checks regularly (like a mailbox you check).

- **AS2**: a special protocol, heavily used by large retailers, which signs and encrypts files and guarantees proof of receipt (see section 8).
- **Manual**: the file is simply stored as an attachment in Odoo, without being sent anywhere.

Profile: the reading recipe

Once the file has arrived (whatever the channel), you need to know **what to do with its content**. That's the role of the **profile**: it says "this file is in EDIFACT format, it represents orders, and here's how each field in the file maps to a field in Odoo".

Mapping: the field-by-field correspondence

A **mapping** is a single line of correspondence: "in the file, the order reference is found here → put it in the Customer Reference field of the Odoo order". A profile contains as many mappings as there are fields to retrieve.

> 📌 **Key takeaway**
>
> **Channel** = how the file travels. **Profile** = how it's understood. The same profile can be used on several channels, and a channel can serve several profiles.

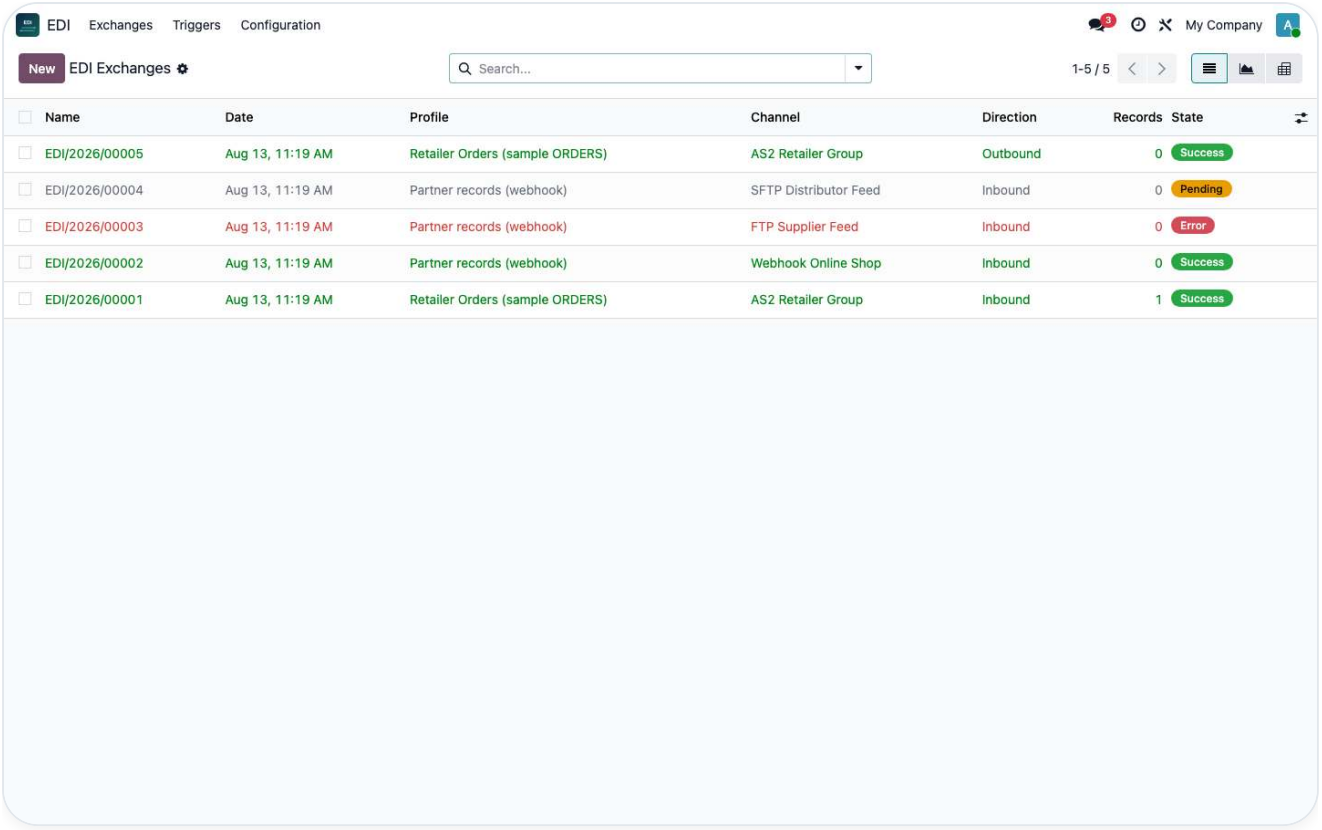
3. Finding your way around the module

The module appears as a separate app in Odoo, with its own icon.

The EDI app on the Odoo apps home screen

Once open, it offers three menus:

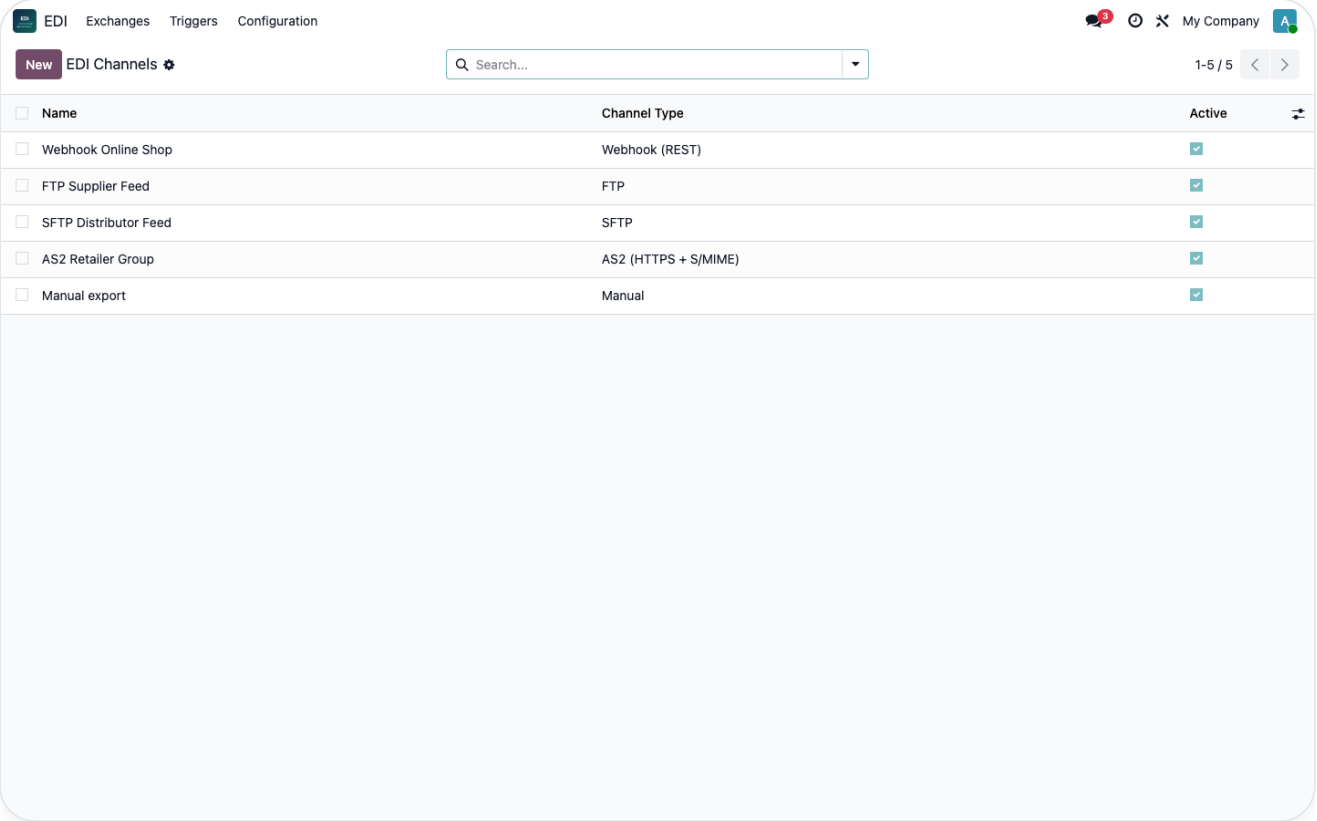
- **Exchanges** — the list of every file received or sent, with its status. This is the screen you'll consult most often.
- **Triggers** — automatically sends a file as soon as an Odoo record changes (for example, as soon as an order is confirmed).
- **Configuration** — contains **Channels** and **Profiles**, i.e. everything that needs to be set up before exchanges can start.



The three menus of the EDI app

4. Channels: how files travel

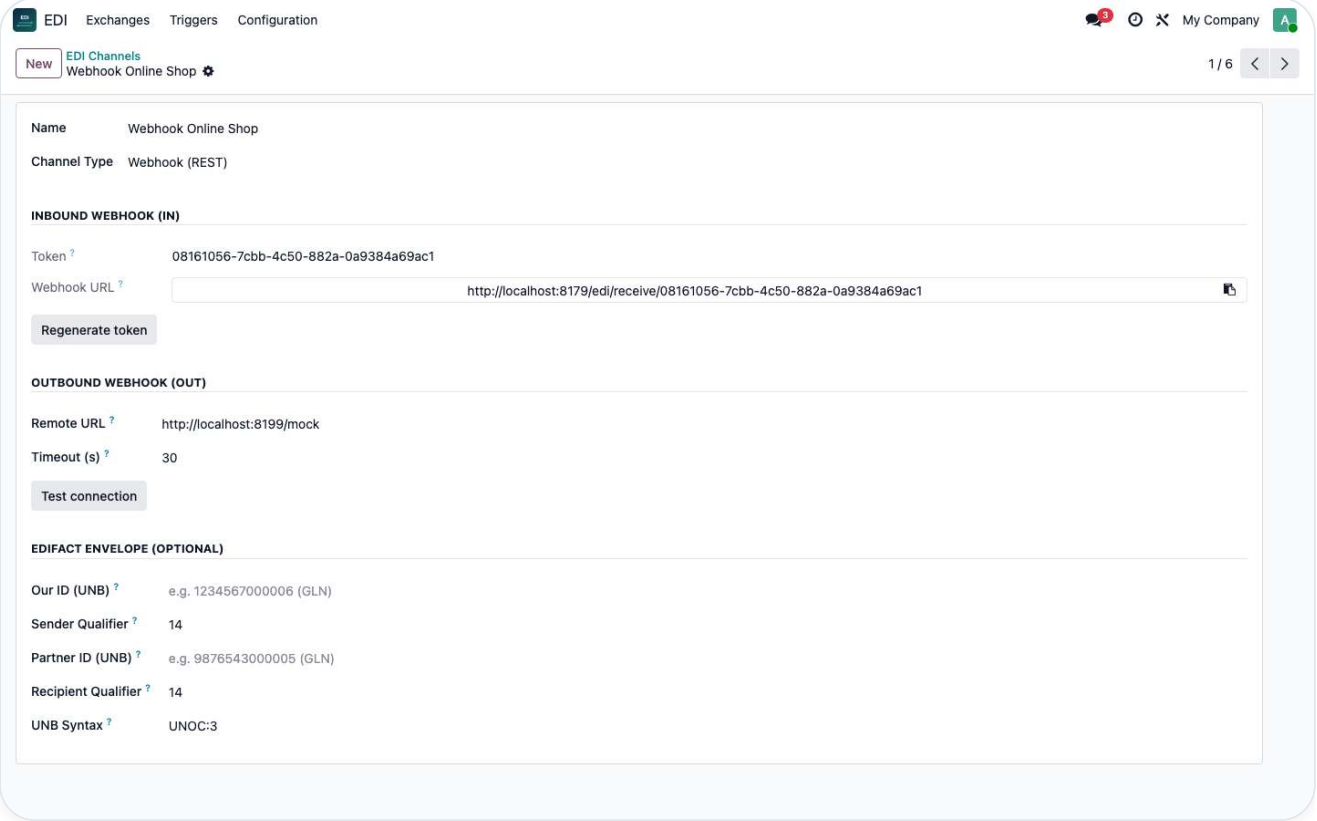
You create one channel per partner (or per type of exchange with a partner). The channel list looks like this:



Channel list — one per partner or means of transport

Webhook channel

The webhook works both ways: Odoo can **receive** a file on a unique web address (auto-generated, to give to the partner), and can **send** a file to the partner's address.



Webhook channel configuration

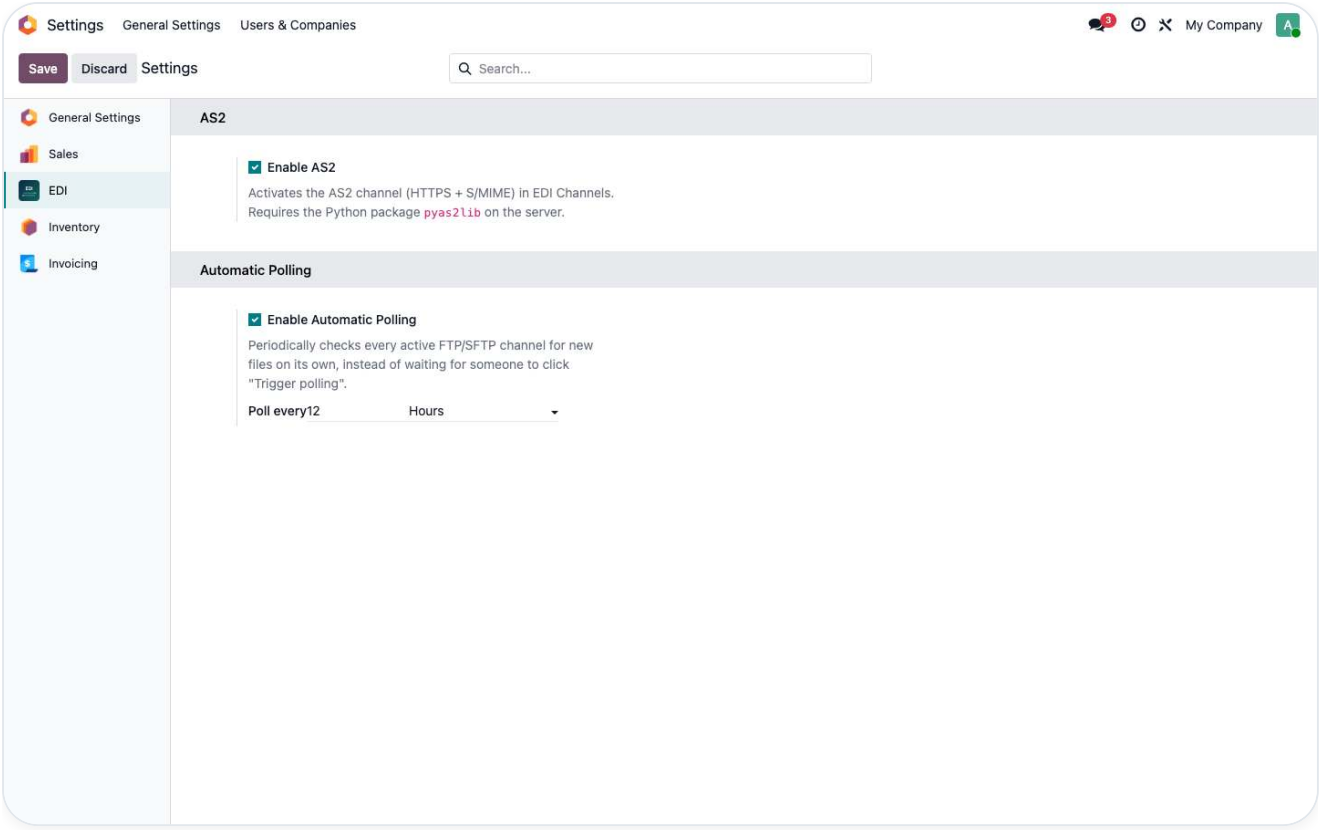
1 Inbound Webhook (IN) — to RECEIVE files: the Token is auto-generated, never fill it in yourself. The resulting Webhook URL is the exact address to give your partner so they can send you their files. "Regenerate token" changes this address — only use it if it has leaked, and you'll then need to give the new one to the partner.

2 Outbound Webhook (OUT) — to SEND files: these two fields only matter if Odoo needs to send files, not receive them. Remote URL = the address on the partner's side (they provide it, it's not for you to invent). Timeout = how long to wait for a response before giving up (30 seconds works in most cases).

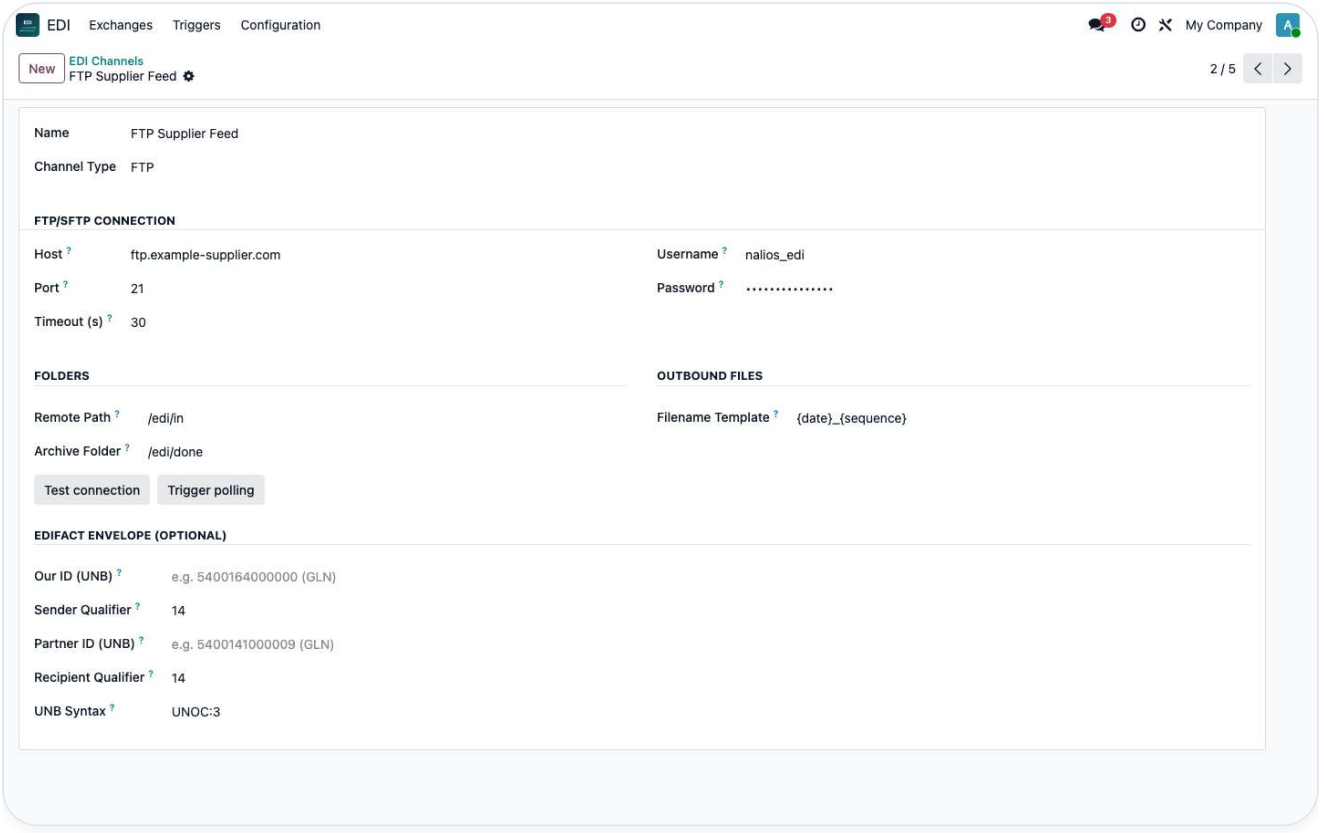
> 📌 **Nothing to poll, nothing to fetch**
>
> Unlike FTP/SFTP, a webhook has no schedule and no "Trigger polling" button — there's simply nothing to go and check. The moment the partner posts a file to the Webhook URL, Odoo processes it there and then, in that same request, and the result is already sitting in **Exchanges** by the time the partner's request completes. The only thing to prepare on your side is making sure that URL is actually reachable from the internet (behind a firewall or reverse proxy, it needs to be exposed).

FTP / SFTP channel

Here, the partner drops their files into a folder on a server. Odoo checks that folder at a regular interval and picks up new files — but this **automatic polling is off by default** and covers every active FTP/SFTP channel at once. Turn it on and set the frequency in **Settings** → **EDI**.



Settings → EDI: turning on automatic polling, checking every 12 hours — twice a day



FTP channel configuration — drop folder and archive folder

Once a file is processed, it is **automatically moved** to the archive folder (instead of being deleted), to also keep a trace on the partner's server side.

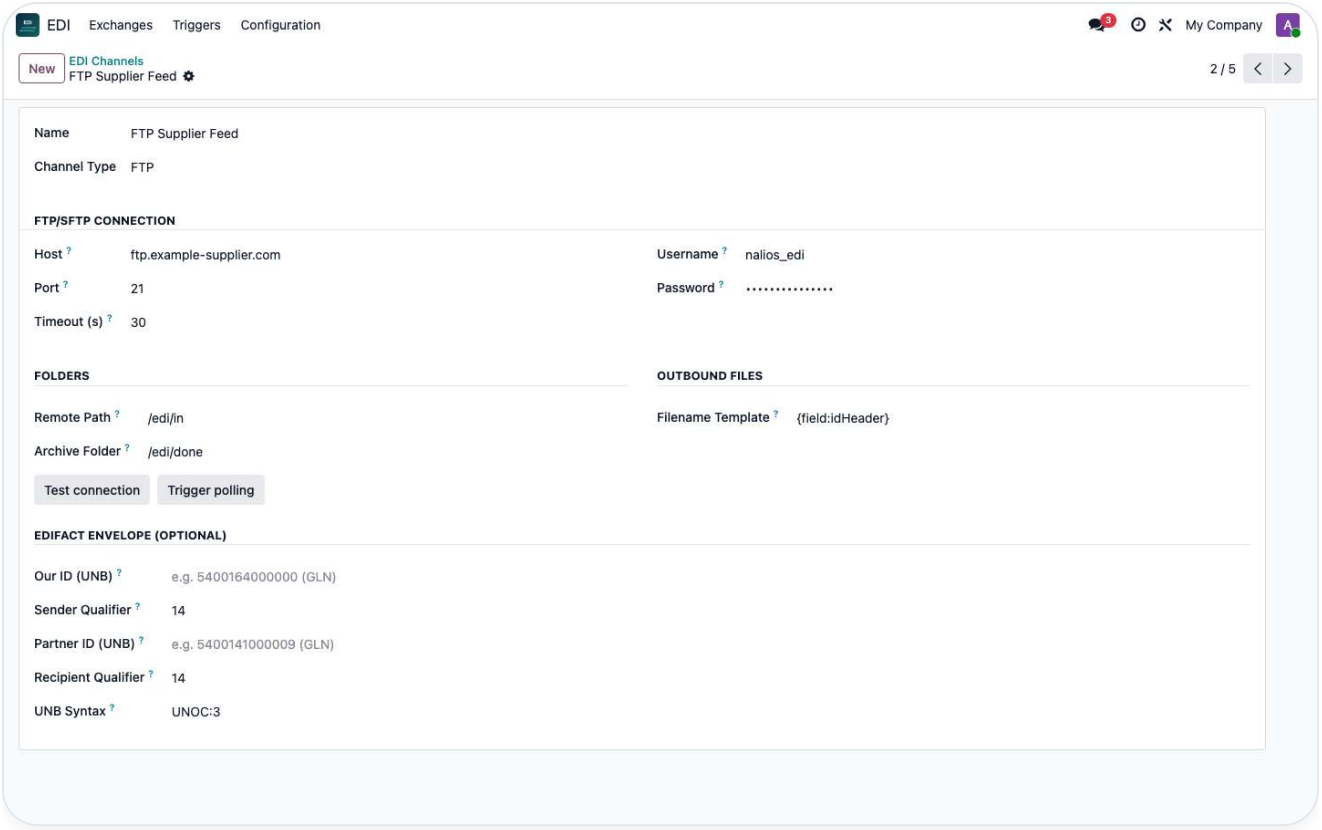
> 💡 **How do we make sure the same file is never processed twice?**
>
> The principle is simple: as soon as a file is picked up and processed, it immediately leaves the drop folder (moved to the archive, or deleted if the move fails). The next pass therefore never sees it again — no need to remember anything. One special case: if the partner ever resends a file with exactly the same name, it will be processed as a new file (the module doesn't compare content) — which is why every file should carry a unique name or reference, as is standard practice in EDI.

> 📌 **One setting, every channel**
>
> There's a single automatic polling schedule for the whole module — it checks *every* active FTP/SFTP channel on the same run, not one schedule per partner. If a partner's server is only reachable during the day, set the interval accordingly (e.g. every 12 hours) rather than the default 5 minutes. Not an FTP/SFTP partner? Nothing to do — Webhook and AS2 are both push-based (the partner sends straight to Odoo, or Odoo sends straight to them), so there's no polling involved on either side.

> 📌 **Not urgent? "Trigger polling" still works**
>
> Automatic polling is optional. Whether it's on or off, the **"Trigger polling"** button on a channel always checks it immediately, on demand.

Naming outbound files

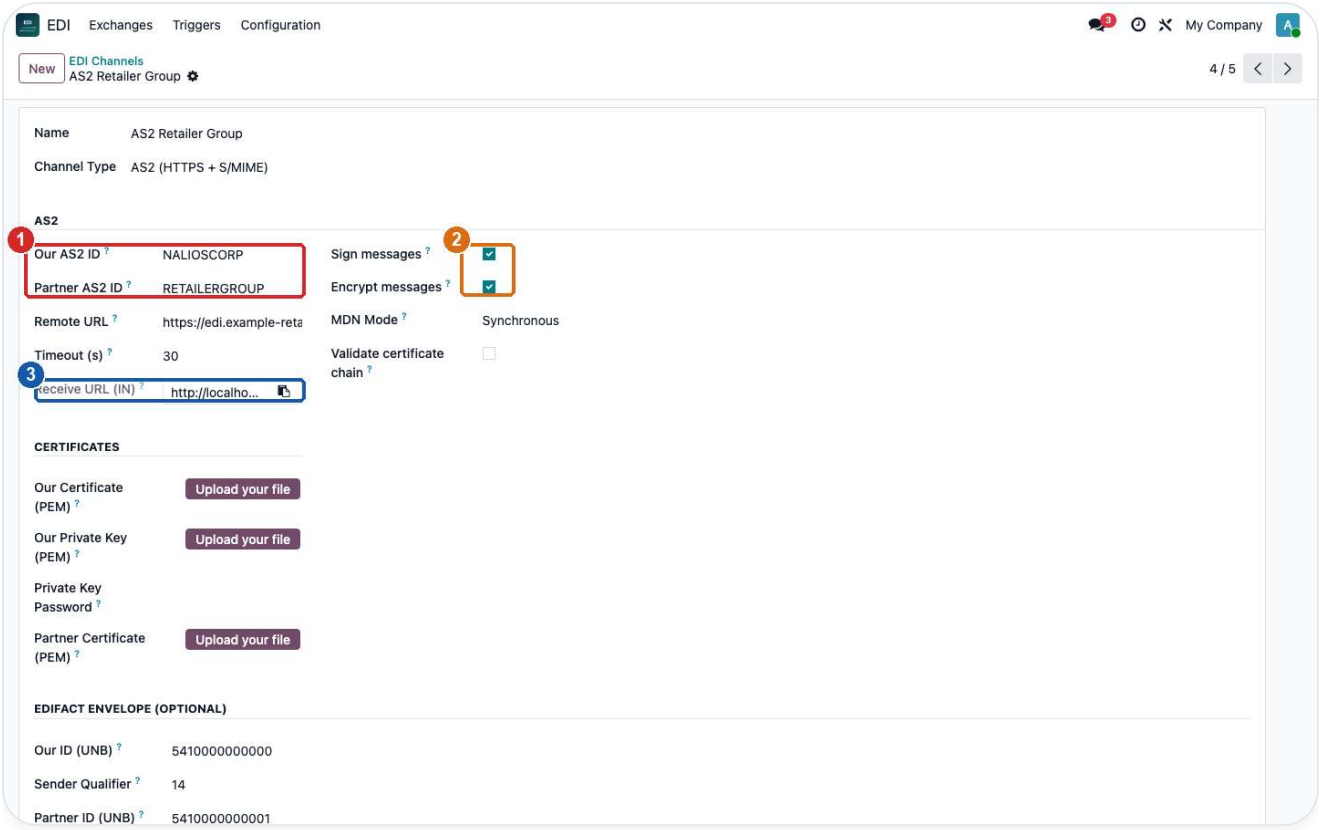
For FTP and SFTP, the **"Filename Template"** field (under "Outbound Files") controls how generated files are named. Besides the generic `{sequence}`, `{date}`, `{model}` and `{direction}` variables, `{field:xxx}` reuses the value of whichever header mapping has `xxx` as its Source Path — useful when the partner expects a specific naming convention built from the document's own reference.



Filename Template using {field:idHeader} — reusing the value mapped to "idHeader" as the file name

AS2 channel

AS2 is richer than the others: digital signature, encryption, delivery receipt. Section 8 is entirely dedicated to it. Here's an overview of the fields nonetheless:



AS2 channel configuration

1. The AS2 identifiers: a code you choose to represent yourself, and the partner's code — to be agreed together before starting.
2. Sign / encrypt messages: check these to secure exchanges (recommended, often required by the partner).
3. The address to give the partner so they can send you files back.

"EDIFACT Envelope (optional)" block

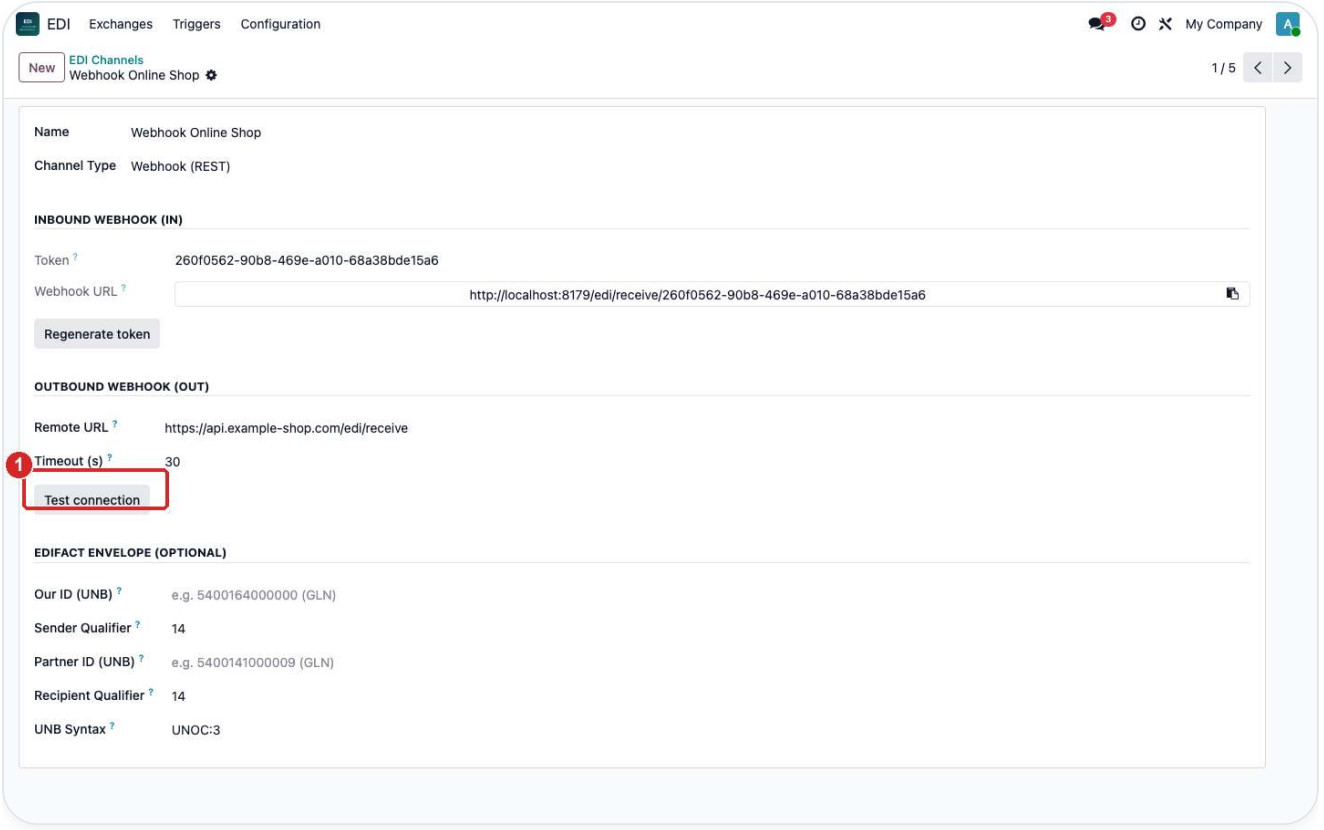
On every channel (whatever its type), an **"EDIFACT Envelope"** block appears at the bottom of the form. It only concerns exchanges in EDIFACT format, and only if your partner requires an **interchange envelope** — the equivalent of a postal envelope around the document itself, carrying the sender's and recipient's identity.

- **Our ID / Partner ID**: usually a GLN (the international identifier of each company), to be agreed with the partner.
- **Qualifiers** and **Syntax**: technical codes specifying the type of identifier used — the default value works in the vast majority of cases.

> 📌 **Key takeaway**
>
> If this block is left empty, outbound EDIFACT files are generated **without an envelope** (just the document itself) — which works for many partners. Only fill it in if the partner explicitly requires a UNB/UNZ envelope.

Testing a connection

For FTP, SFTP and Webhook channels, a **"Test connection"** button lets you immediately check that the credentials and address are correct, without waiting for the first real file.

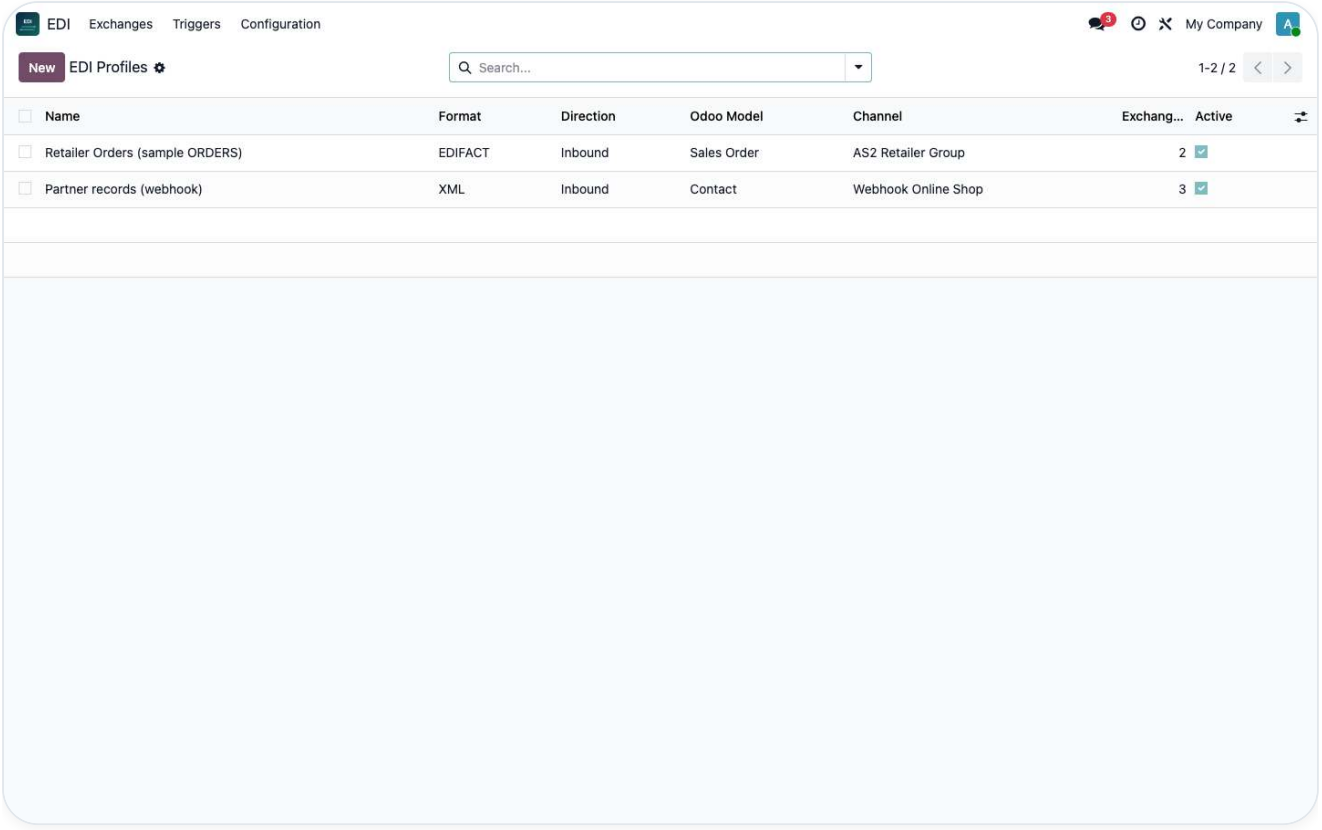


The "Test connection" button, shown here on a Webhook channel — also available on FTP and SFTP

> 📌 **Not sure about a field? The little "?" has the answer**
>
> Every somewhat technical field on these screens (Token, URL, EDIFACT identifiers, qualifiers...) has a full explanation available by hovering over the small question mark next to its name — no need to guess what to put there.

5. Profiles: the reading recipe

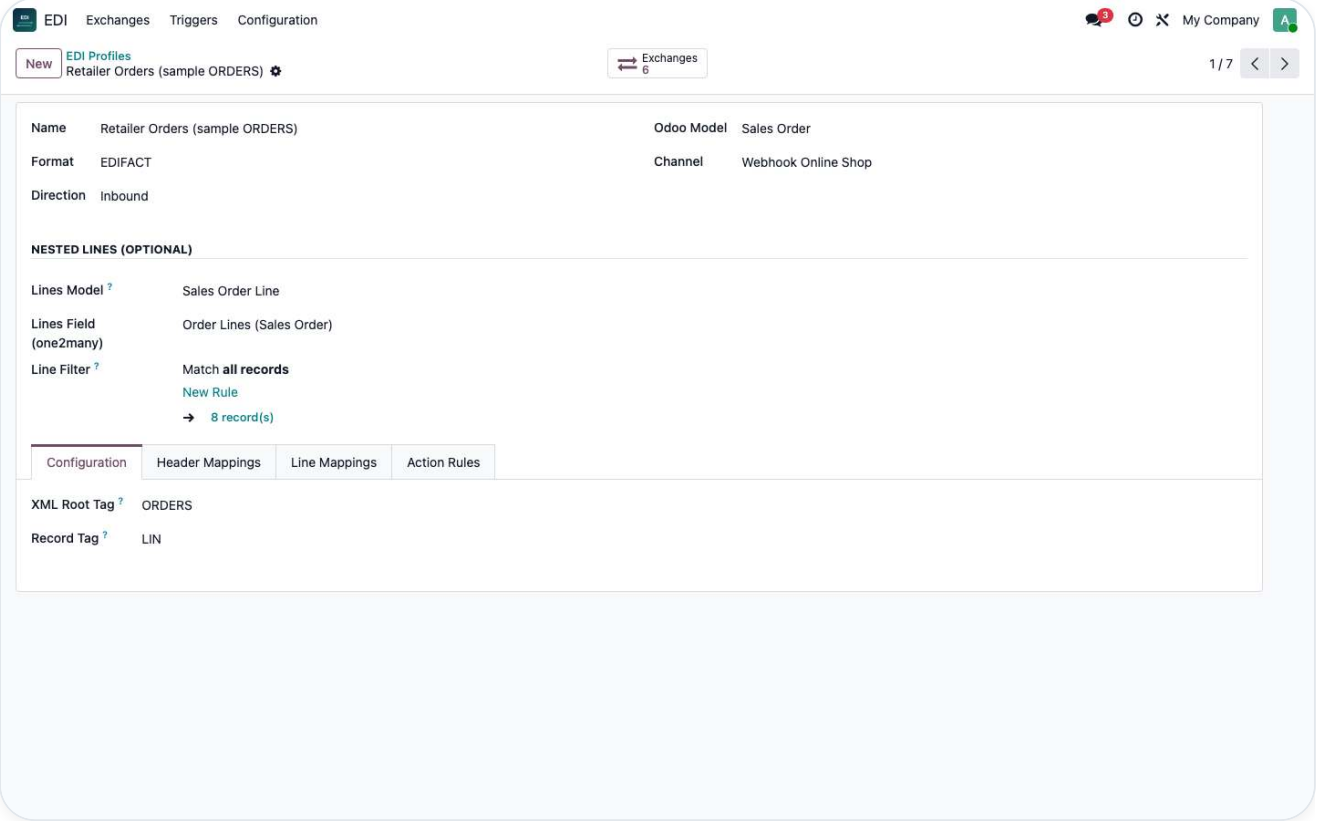
Each profile describes a type of document (an order, a customer record...) in a given format, and how to turn it into Odoo data.



List of configured profiles

Configuration tab

Here you define the **format** (XML, JSON, EDIFACT, fixed-width or CSV), the target **Odoo model** (what type of record to create: customer, order...), the direction (inbound/outbound), and two technical markers: the **root tag** and the **record tag**. For CSV, a **"CSV Options"** block also appears to choose the delimiter (comma, semicolon...) and whether the first line contains column headers.



Configuration tab of a profile — here for reading EDIFACT orders and creating sales orders with their lines

💡 Root tag and record tag: two different questions

Root tag answers "what TYPE of document is this, as a whole?". It's only needed if the same channel can receive several different types of documents (sometimes orders, sometimes invoices) — the module uses it to automatically pick the right profile. If this channel only ever receives one type of document, this field can stay empty.

Record tag answers "where does ONE record start, inside the file?". This one is almost always needed: it's what tells the module "a new record starts here".

Format	Root tag (example)	Record tag (example)
XML	`Orders` (tag wrapping the whole file)	`Order` (tag repeated for each record)
JSON	the top-level key	the key holding the list of records
EDIFACT	`ORDERS` (message type after `UNH`)	`LIN` (segment starting each product line)
Fixed-width	—	`DD` or `CC` (code starting each new record)

Concrete example: an EDIFACT order file with 3 products → root tag = `ORDERS` ("this is an order"), record tag = `LIN` ("each LIN segment = one product") → the module creates 3 order lines.

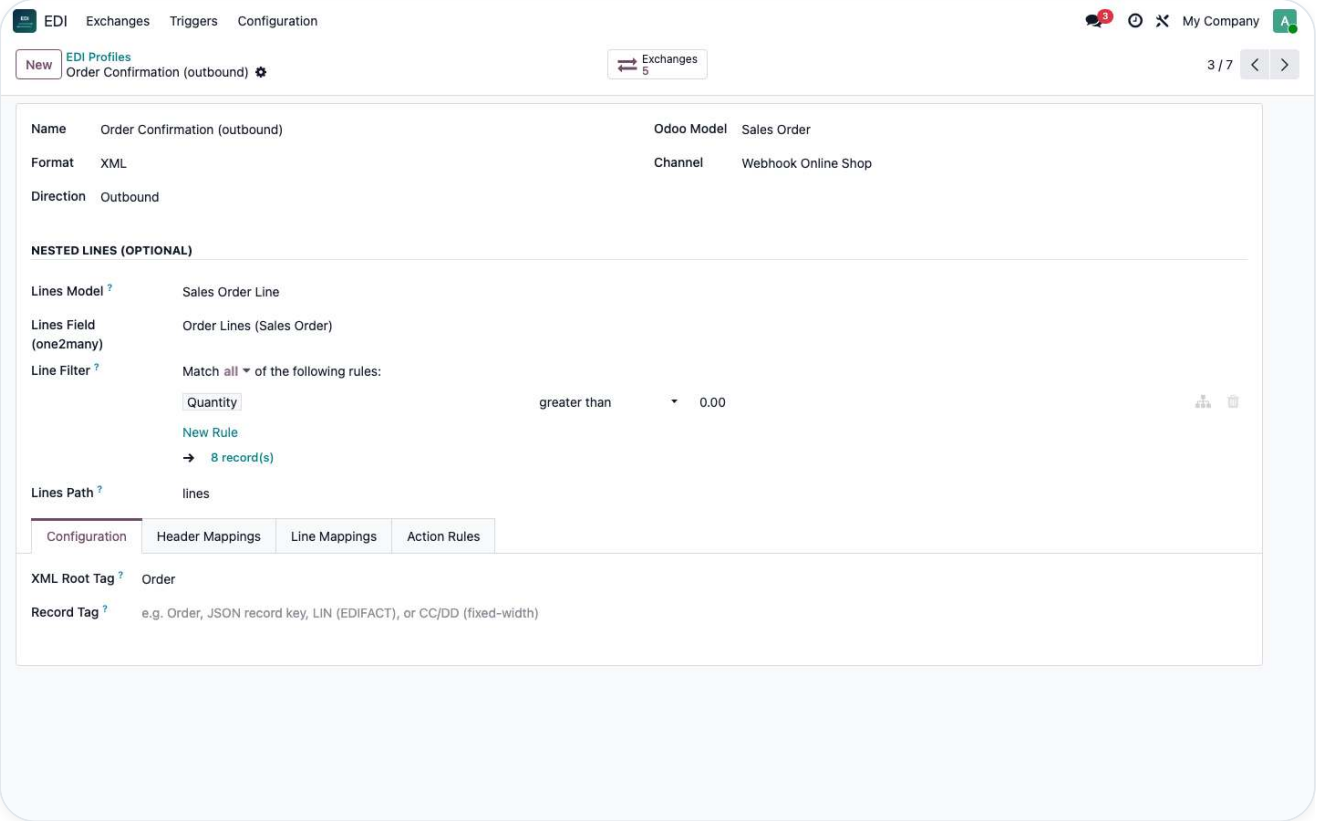
📌 Batch Size: grouping several records into one file

When several records are sent together at once (for example a bulk export over a batch of orders, rather than one trigger per order), **"Batch Size"** controls how many of them land in a single generated file: 0 means no limit — everything in one file. This setting only applies to profiles *without* "Nested Lines" — a nested profile always produces one file per parent record.

> 📌 **Order with lines: "Nested Lines"**
>
> An order isn't just a header (the customer, the reference) — it also has product lines. The **"Nested Lines"** block lets you specify that this profile should create *one single* order with *all its lines* attached, rather than a separate record per product line. Here you specify the lines model (e.g. Order Line) and the field connecting them to the header.

Line Filter: excluding certain lines

Once "Nested Lines" is set, two more fields appear right below it. The first, a **"Line Filter"**, is an optional condition that decides which lines actually make it into the outbound file. Lines that don't match are simply left out — no need to clean up the data beforehand.



The full "Nested Lines" block once configured: Line Filter (only lines with a quantity greater than 0) and Lines Path (see next) side by side

> 💡 **A common case: skip empty or non-physical lines**
>
> A delivery can contain lines that shouldn't reach the partner: a line at quantity 0, a service (handling fees), a pallet... A filter like `Quantity > 0` takes care of it automatically. And if *every* line ends up excluded, the module doesn't send an empty file at all — nothing is generated for that record.

Lines Path: nesting lines inside a wrapper (XML only)

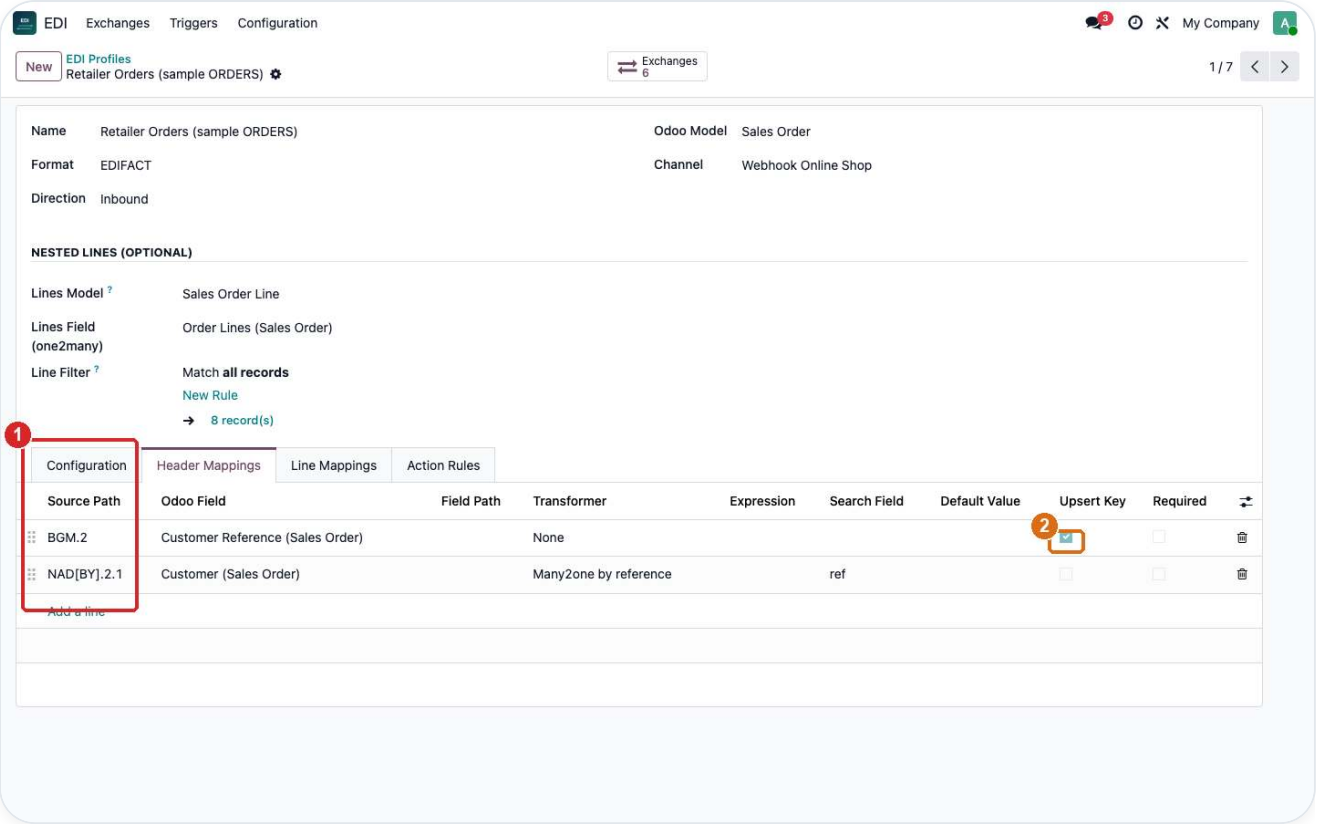
By default, the repeated line elements are attached directly under the root element. Some partners expect them wrapped in their own container instead — for example a single `<lines>` tag holding every line, or even a whole extra level between the root and the document itself. **"Lines Path"**, shown right below Line Filter in the figure above, tells the module where to attach them, as a slash-separated path.

> 💡 **Reusing the same wrapper as a header field**
>
> If a header mapping's Source Path already creates a wrapper (e.g. `goodsReceipt/idHeader`, nesting everything one level under a `goodsReceipts` root), set Lines Path to `goodsReceipt/lines` — the module reuses the very same `<goodsReceipt>` element instead of creating a second one, so the lines end up right where they belong: `<goodsReceipts><goodsReceipt>...<lines>...</lines></goodsReceipt></goodsReceipts>`. Leave the field empty to attach lines directly under the root, as before.

> 💡 **Several orders in a single file**
>
> Some partners group several orders into a single EDIFACT file sent at once (an "interchange" containing several documents). The module detects this automatically and creates **one separate order per document**, each with its own lines — never mixing information from two different orders.
>
> Same logic for a **CSV** file that would contain several orders one after another (one row per product, with the order reference repeated on every row): the module automatically groups rows that share the same header information.
>
> And in **fixed-width**, a "product line" can itself be written across several physical lines of the file (for example three technical lines together describing a single delivered item). The record tag marks the start of each new group — everything that follows, up to the next one, belongs to it.

Header Mappings tab

This is where the heart of the configuration lives: each row in the table says "go find this piece of information in the file, and put it in this Odoo field".



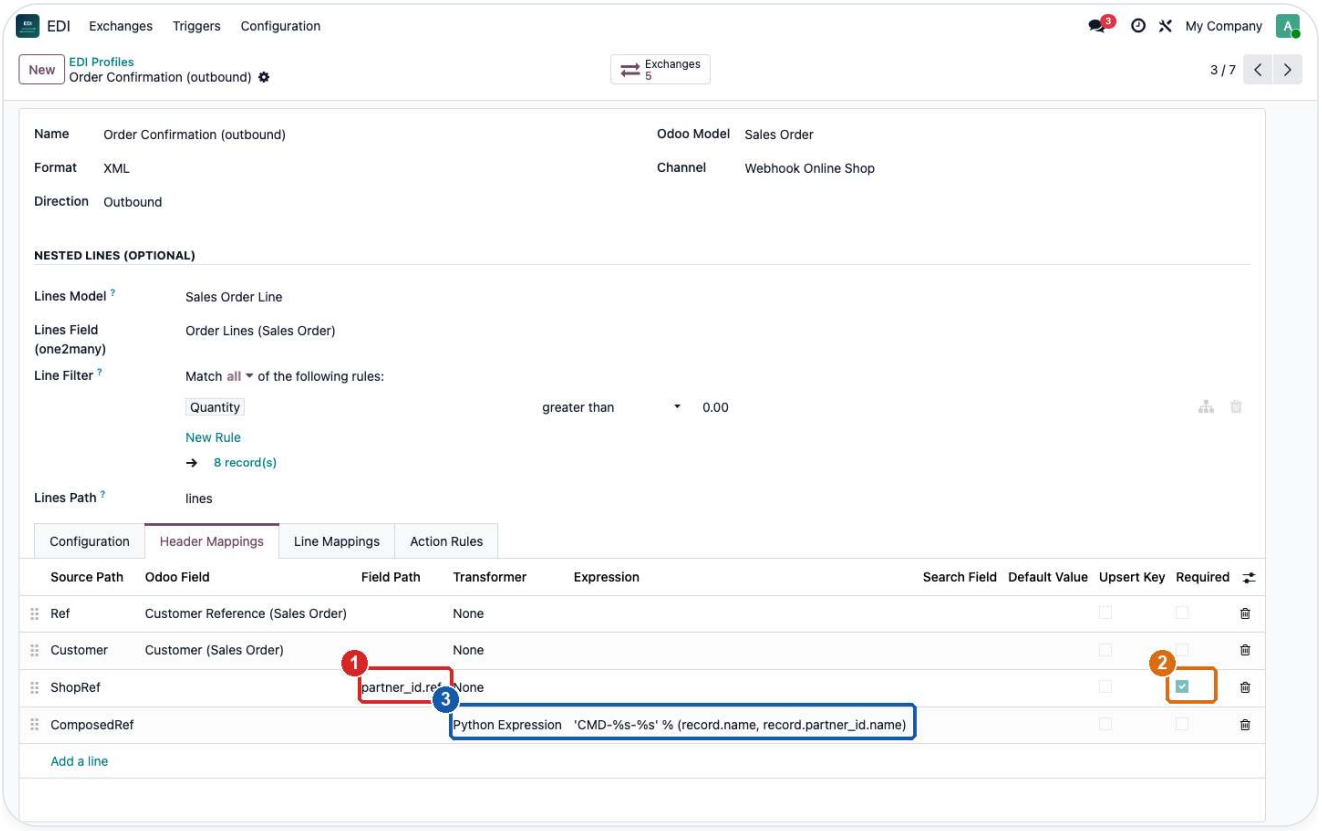
Header mappings — each row links a location in the file to an Odoo field

1 Source Path: the exact location of the data in the received file (the syntax depends on the format — see the box below).

2 Upsert Key: when this box is checked, this field is used to recognize whether the record already exists (to update it instead of creating a duplicate).

Column	What it means
Odoo Field	The destination field in Odoo (e.g. Customer Reference, Customer...)
Field Path	Outbound only — an alternative to "Odoo Field" when the value actually lives on a *related* record (e.g. `partner_id.ref` to read the customer's reference). Leave "Odoo Field" empty when using this.
Transformer	A conversion applied to the read value: turning it into a number, a date, or looking up an existing record (customer, product) by one of its references.
Expression	Only if the chosen Transformer is "Python Expression": a small custom transformation formula (reserved for special cases, usually set up with a consultant's help).
Search Field	For "Many2one" transformers: which field to use to find the existing customer/product (e.g. its barcode, its reference).
Default Value	Value used if the information is missing from the file.

Required	Outbound only — if this value is empty, the file isn't sent at all: the exchange is marked in error instead, with a clear message, so the missing information doesn't silently reach the partner.



Three outbound-only columns in action, on the same profile's Header Mappings

1. Field Path: reads partner_id.ref on the linked customer, since "Odoo Field" alone can't reach a related record.
2. Required: checked here — if this value ever comes back empty, the file isn't sent, the exchange goes to error instead.
3. Transformer set to "Python Expression": composes a brand new value from two fields at once (see the box below).

> 💡 **Composing a value from several fields (outbound)**
>
> Sometimes a partner expects a single value built out of two or more Odoo fields glued together — for example `CMD-S00001-Norda Retail Group` in the screenshot above, combining an order reference and a customer name with a fixed prefix. None of the transformers below can do that on their own, since each mapping normally reads from a single field.
>
> **Python Expression** can: for outbound mappings, both `value` (this mapping's own resolved value) and `record` (the whole record being sent) are available in the Expression field. `'CMD-%s-%s' % (record.name, record.partner_id.name)` produces exactly that composed reference — "Odoo Field" and "Field Path" can even be left empty in this case, since the expression builds the value entirely from `record` on its own.

💡 **Every "Transformer" option**

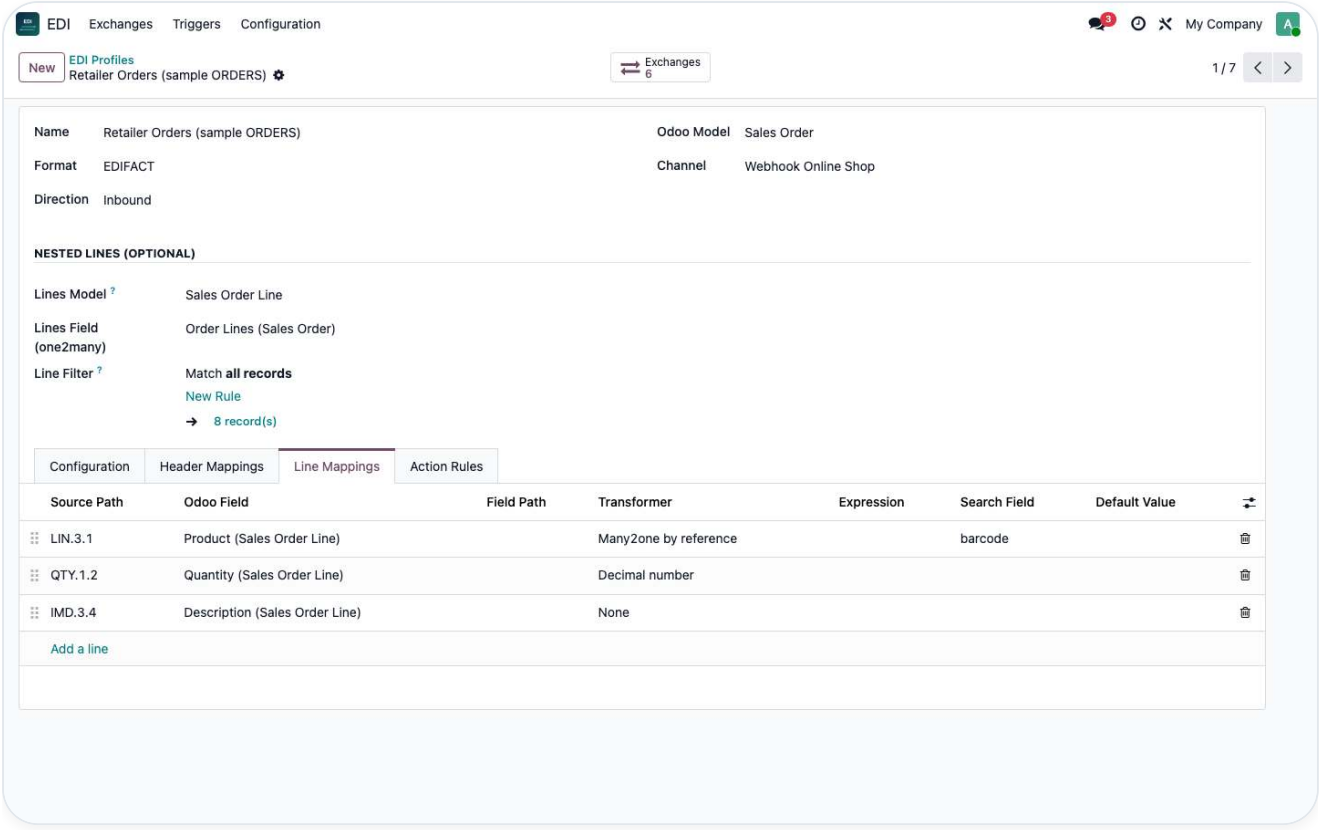
Transformer	Effect
None	The value is used as-is, without conversion (plain text).
Date (YYYY-MM-DD)	Parses the value as a date written year-month-day (the format Odoo stores dates in internally).
Date (DD/MM/YYYY)	Same, but for a date written day/month/year — the common European format used by many partners.
Decimal number	Converts the value to a number with decimals (e.g. a price or a quantity like 12.5).
Integer	Converts the value to a whole number (e.g. a quantity like 12).
Boolean (true/1/yes)	Converts the value to a checkbox (Yes/No), recognizing "true", "1" or "yes" (case-insensitive) as checked.
Many2one by name	Looks up an existing customer/product by its *name* (or the field chosen in "Search Field") instead of creating a new value.
Many2one by reference	Same idea, but searches by *reference* (barcode, internal code...) instead of name — the usual choice for EDI, since names rarely match exactly between systems.
Python Expression	A custom formula for special cases (see "Expression" above) — reserved for situations the other transformers can't handle.

💡 **Understanding "Source Path" by format**

- **XML**: the tag path, e.g. `Header/RefCmd` or `@Ref` for an attribute.
- **JSON**: the key, e.g. `ref` or `address/city` for a value nested in a sub-object.
- **EDIFACT**: a compact notation `SEGMENT.element.component` — for example `BGM.2` means "in the BGM segment, the 2nd element". When a segment can appear several times with a different role (e.g. the buyer's or the supplier's address), a qualifier is added in brackets: `NAD[BY].2.1` means "the NAD segment qualified BY (buyer), 2nd element, 1st component". This is the most technical part of the module — it requires some knowledge of the partner's EDIFACT structure, usually provided in their integration manual.
- **Fixed-width**: `CODE.position.length` — for example `HH.2.10` means "on the line starting with HH, read 10 characters starting at position 2". Position is counted from 0 (the very first character of the line).
- **CSV**: simply the column name (e.g. `customer_reference`), or its index if the file has no header row.

Line Mappings tab

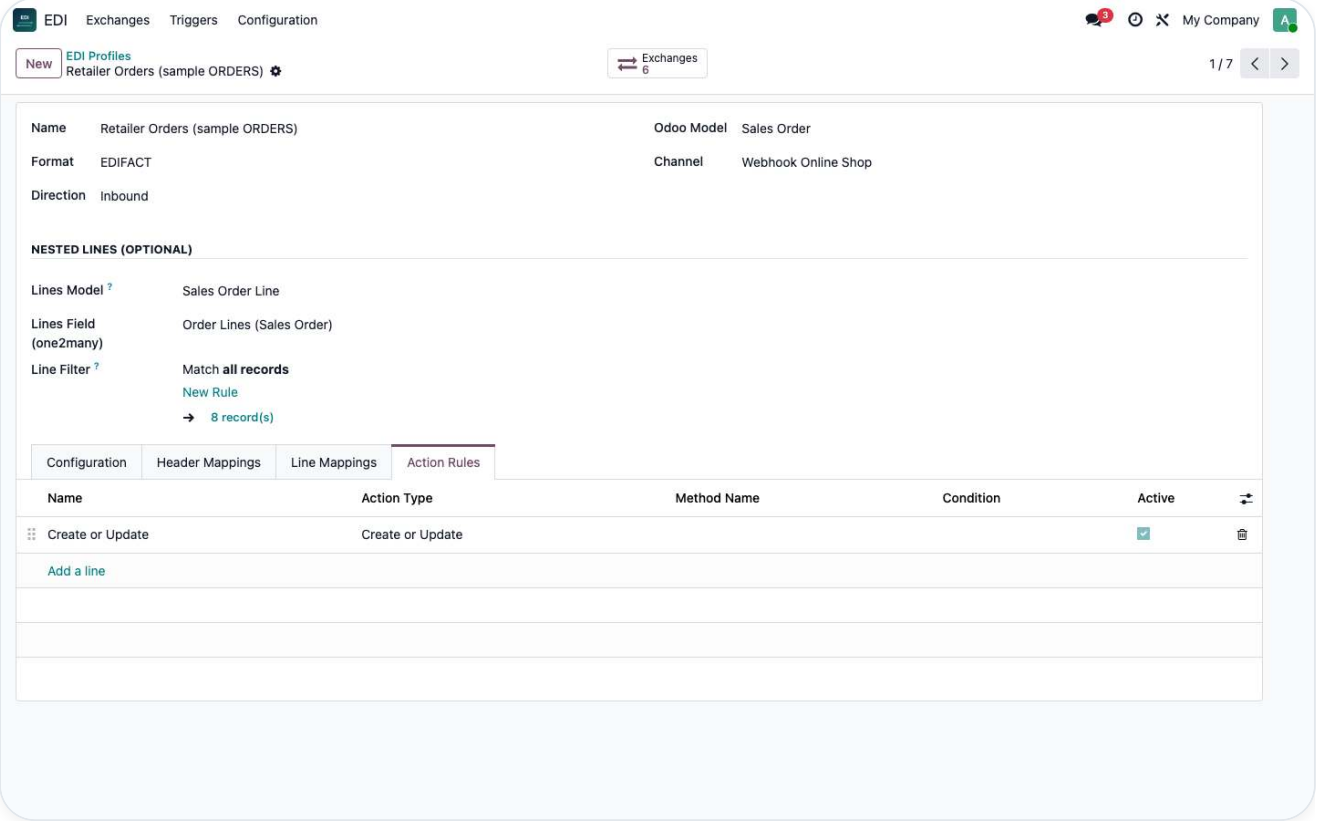
If the profile has "nested lines" enabled, a second, identical table appears, but for the product line fields (quantity, product, description...) instead of the header.



Line mappings — an EDIFACT line (LIN) becomes an Odoo order line

Action Rules tab

Once all the values have been retrieved, Odoo needs to be told what to do with them: create a new record, update an existing one, or either depending on the case ("upsert", the most common choice).



Action rules — what to do once the data has been extracted

Action	Effect
Create	Always creates a new record, even if a similar one already exists.
Update	Modifies an existing record (found via the "Upsert Key"). Does nothing if none is found.
Create or Update	The usual choice: updates if the record exists, creates it otherwise.
Call method	Triggers an existing Odoo action on the record (e.g. confirming an order).
Chatter notification	Simply adds a message on the record, without modifying it.

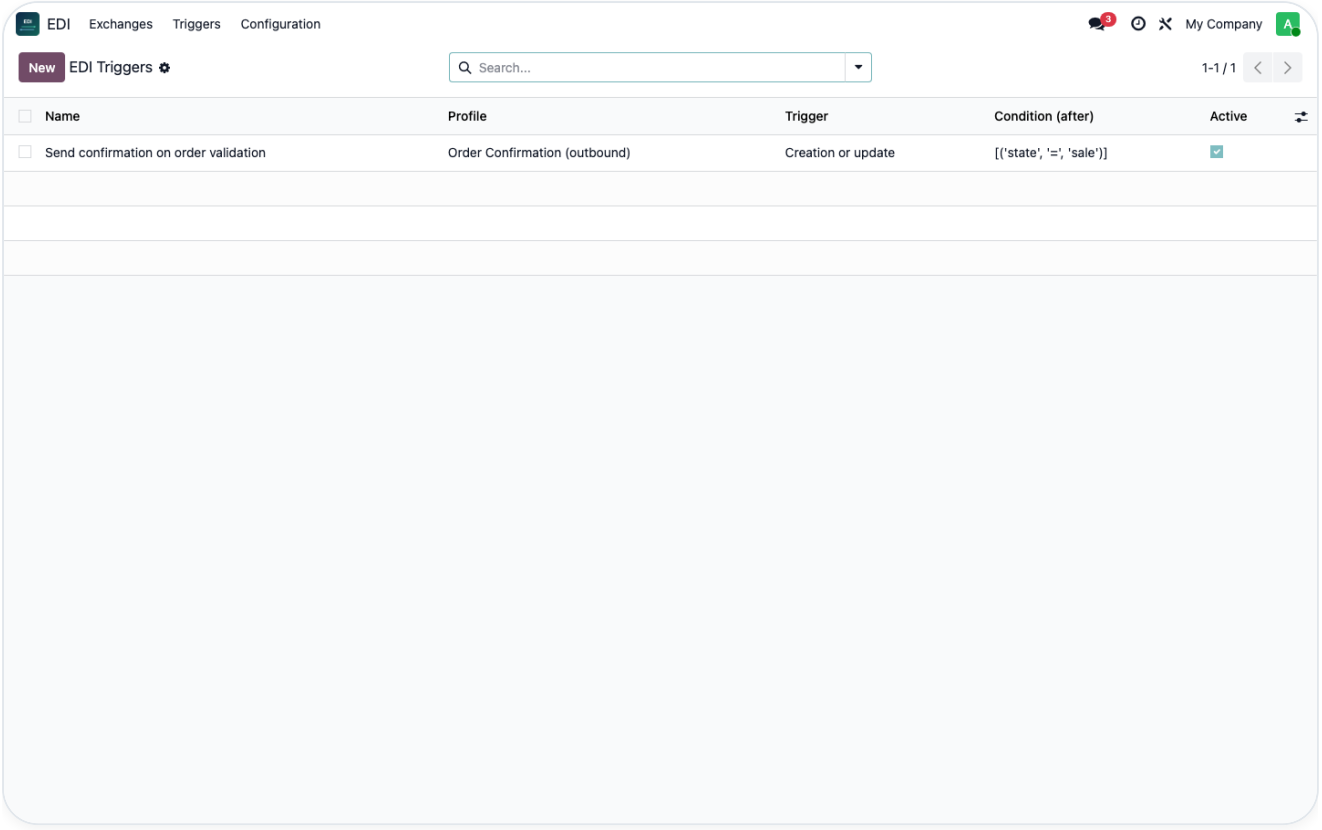
💡 **"Condition": running a rule only in certain cases**

Each action rule can carry an optional **Condition** — a small formula that must be true for that rule to run, checked against the values just extracted from the file (e.g. `values.get('state') == 'done'` only runs the rule if the order's status in the file is "done").

This is what lets several rules coexist on the same profile for different situations — for example, "Call method" (confirm the order) only when the status is "done", and "Chatter notification" as a fallback otherwise. Leave it empty for a rule that should always run.

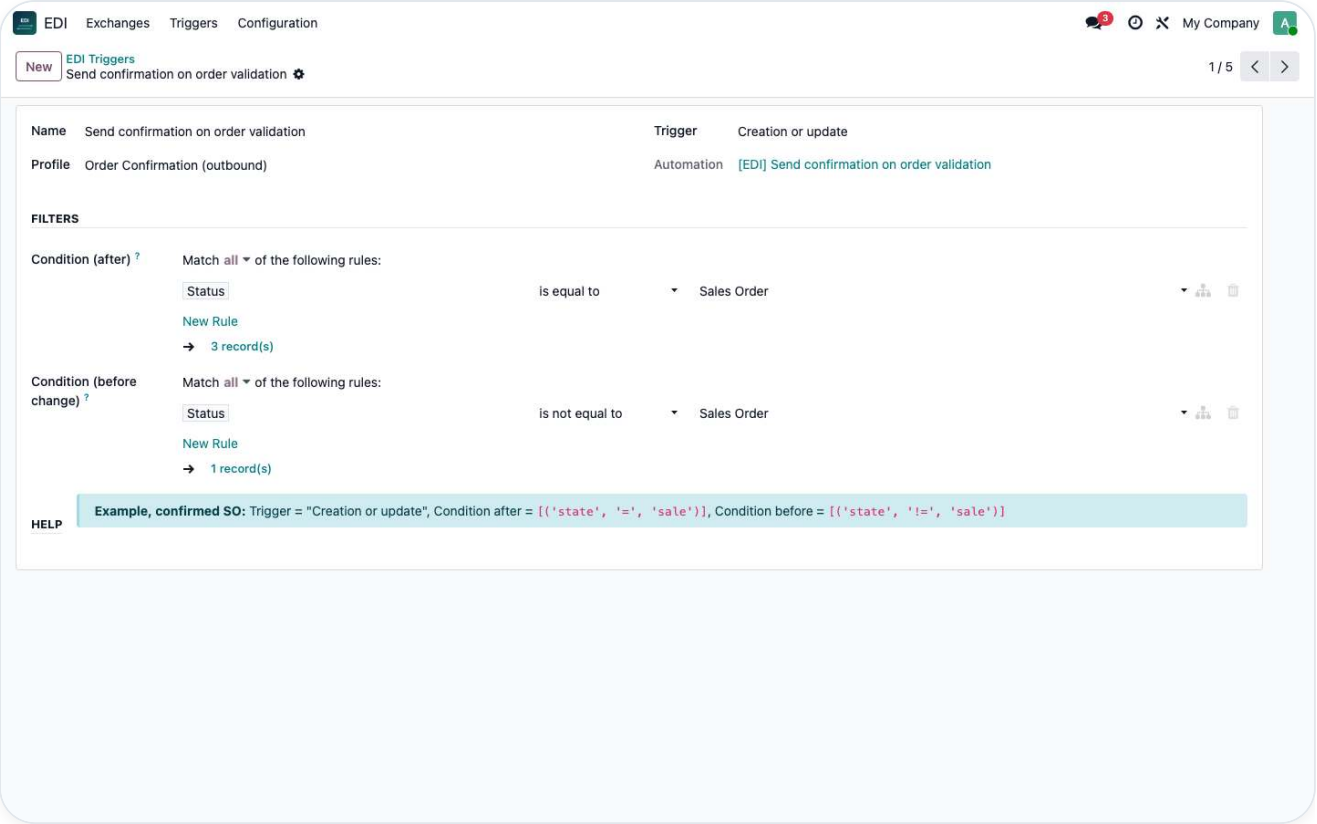
6. Triggers: automating outbound sending

So far, every example involved a **received** file. But the module can also automatically **send** a file the moment an Odoo record changes — no button to click, no manual action. That's the role of **triggers**.



List of configured triggers

A trigger connects three things: an **outbound profile** (the one that knows how to build the file), an **event** (creation, update, or deletion of a record), and optional **conditions** so it only fires in specific cases.



A trigger — send a confirmation as soon as an order moves to the "Sales Order" status

Field	What it means
Profile	The outbound profile used to build the file being sent (only outbound or both-direction profiles appear here).
Trigger	When to check the conditions: on creation, on creation or update, or on deletion of the record.
Condition (after)	The file is only sent if the record, once the action is done, matches this filter (e.g. the order is now confirmed).
Condition (before change)	Optional — lets you trigger only on a real *state transition* (e.g. the order was NOT yet confirmed before), rather than on every save of an already-confirmed order.

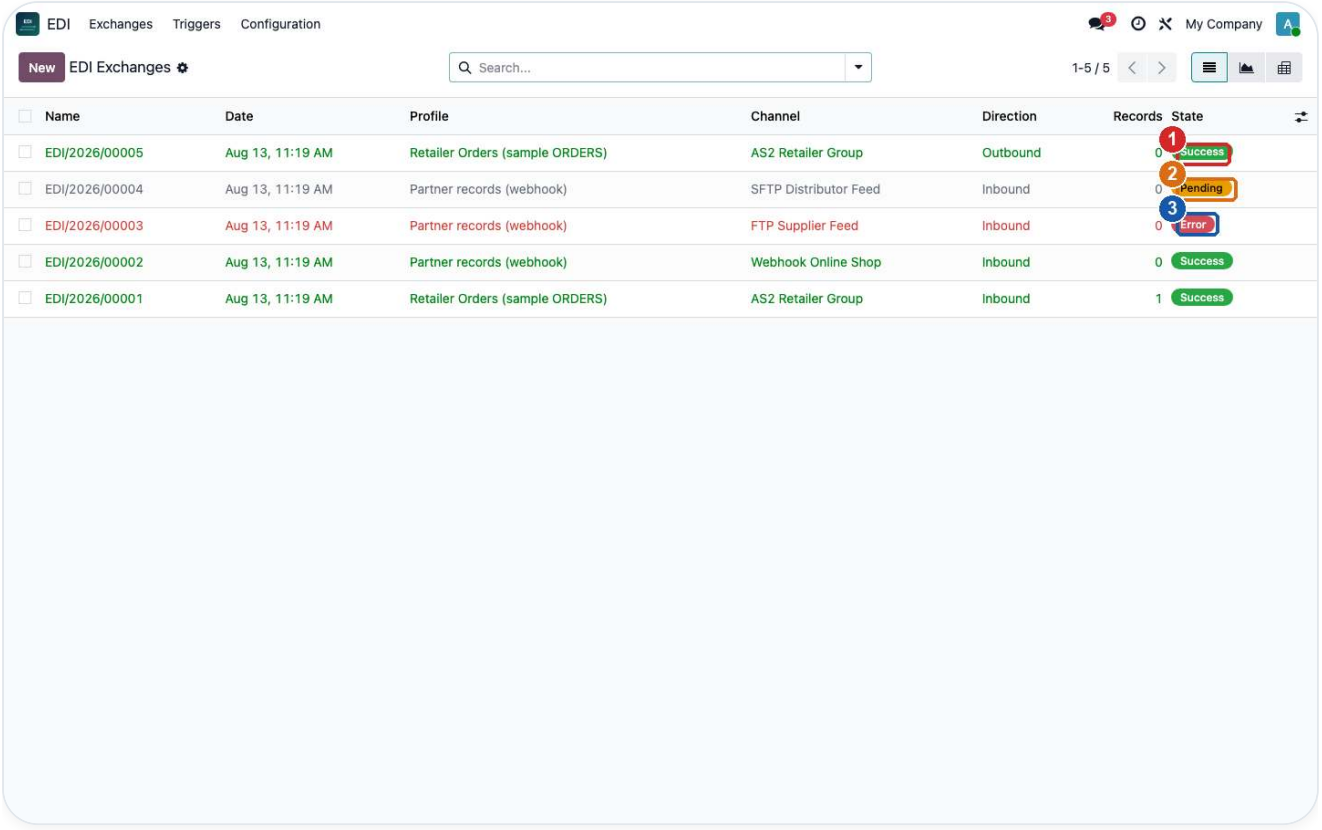
> 📌 **A visual filter, not a formula to type**
>
> Both conditions use the same visual filter builder as the "Line Filter" seen earlier for profiles — pick a field, an operator, a value, and the underlying domain is built for you. The examples below are shown in their raw form only to explain what's actually being checked.

> 💡 **The most common example: confirming an order**
>
> Automatically send a confirmation as soon as an order moves to "Sales Order" status: Trigger = "Creation or update", Condition (after) = `[('state', '=', 'sale')]`, Condition (before change) = `[('state', '!=', 'sale')]` — this last filter avoids resending the file on every later edit of an already-confirmed order.

> 📌 **Under the hood**
>
> Each trigger automatically creates a standard Odoo automation rule (visible in Settings → Technical) — no parallel mechanism to maintain, the module relies on Odoo's native automation engine.

7. Following exchanges day to day

This is the screen you'll open most often: the list of everything that has gone through the module, with a clear color code.

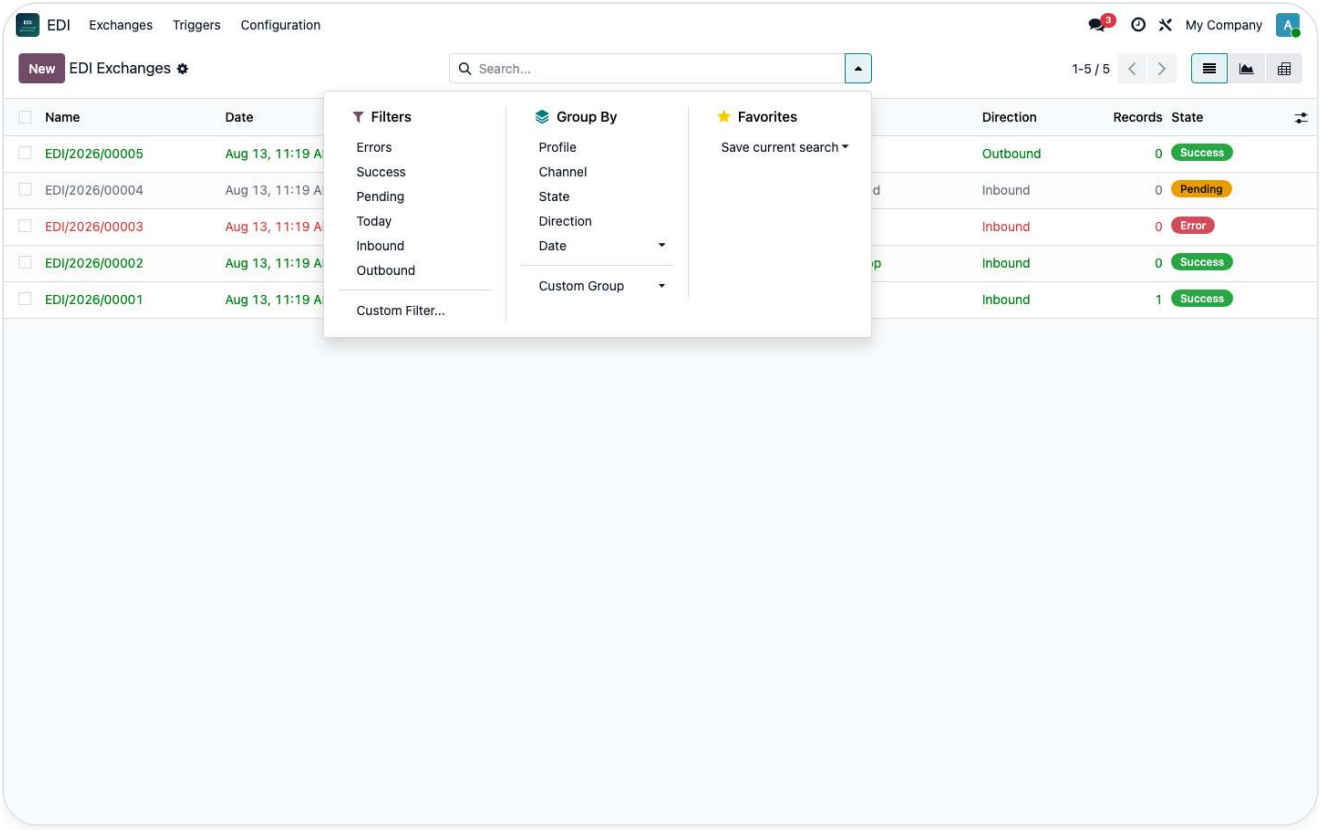


Exchange list with their status

1. Success (green): everything went well, the Odoo record was created or updated.
2. Pending (orange): the file has arrived but hasn't been processed yet.
3. Error (red): something prevented processing — see below how to read the details.

Filtering and grouping

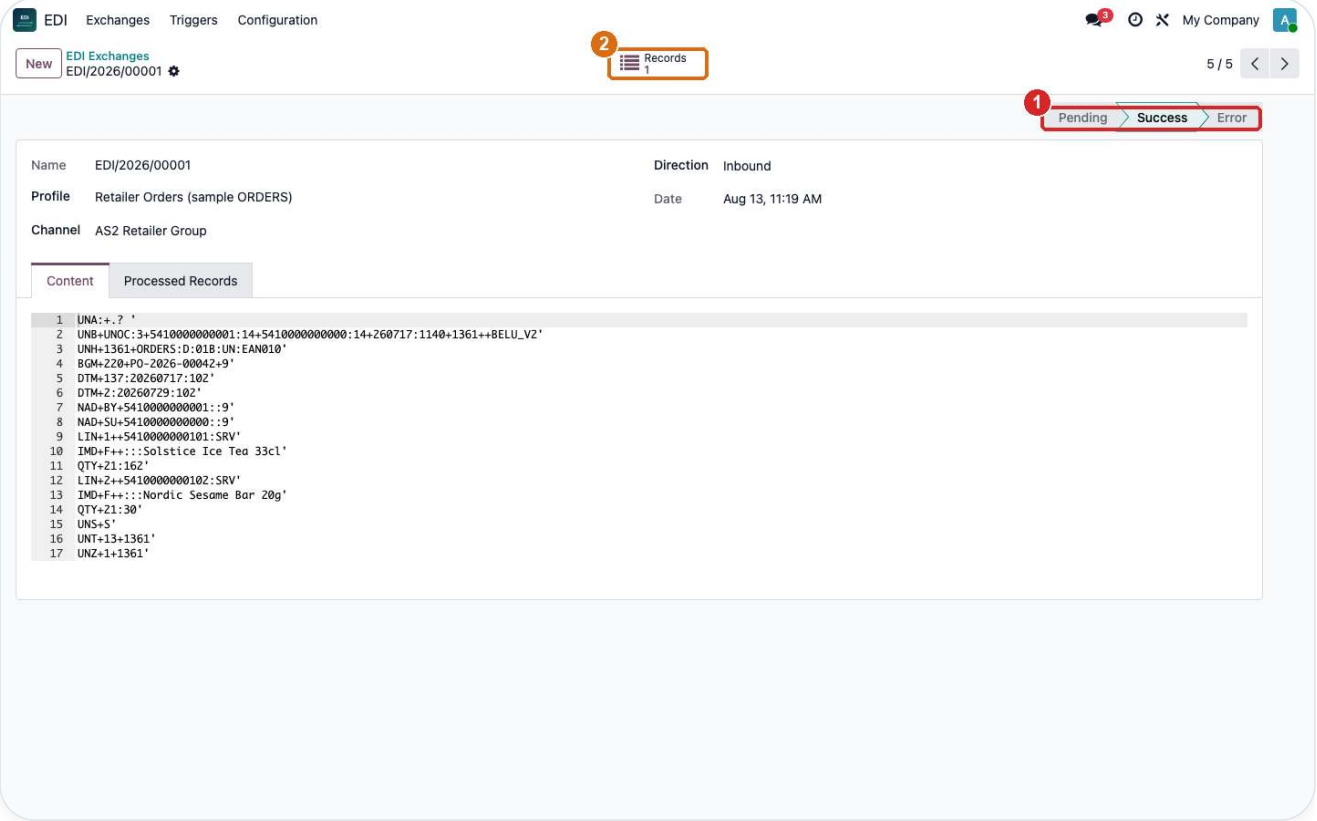
The search menu offers ready-to-use quick filters.



Quick filters: errors, today, inbound/outbound, grouping by profile or channel...

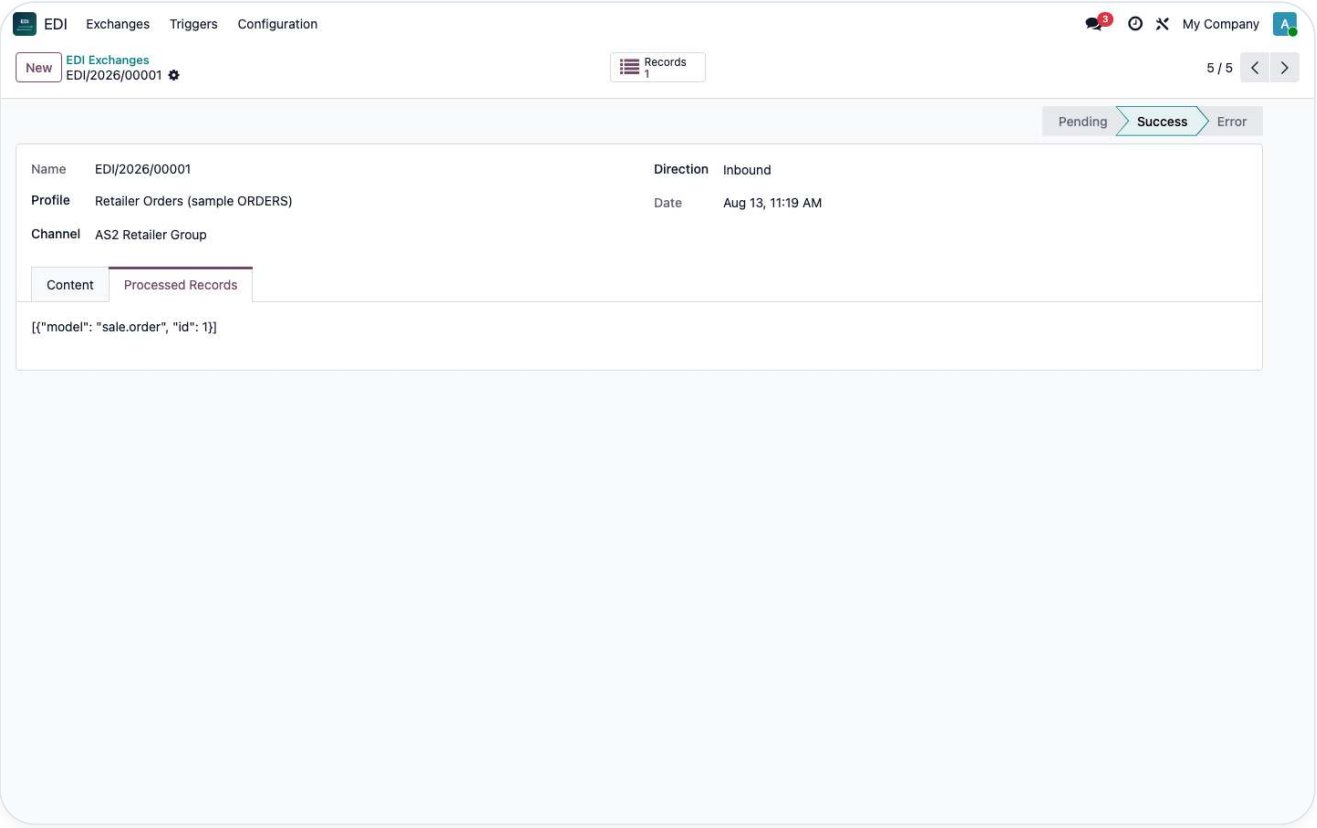
Exchange details

Opening an exchange shows you its raw content, its progress, and — once successfully processed — direct access to what was created.



Details of a successful exchange

1. The status bar shows where processing stands.
2. The "Records" button opens directly the Odoo record created or modified by this exchange.



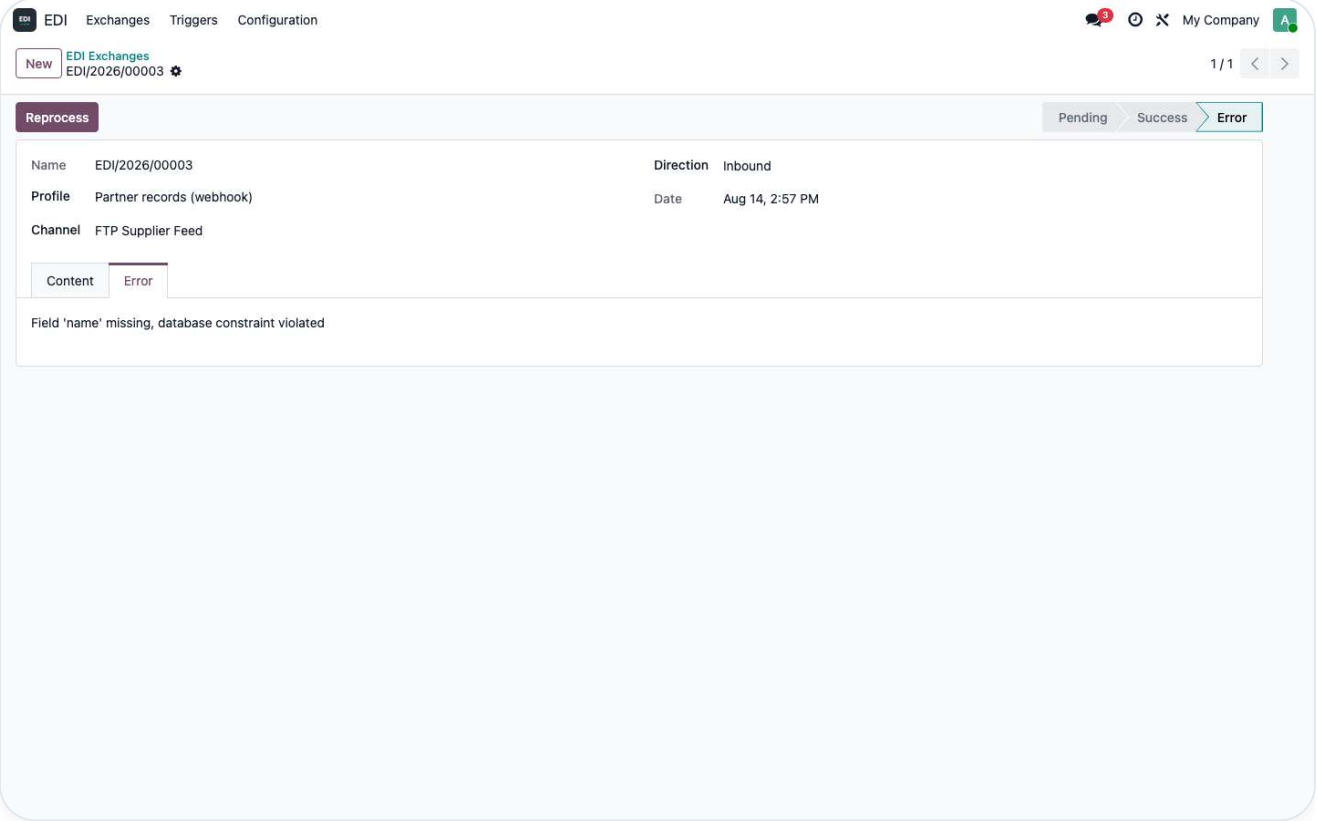
The "Processed Records" tab technically lists what was affected

> 💡 **"Reprocess" vs "Retry" — what's the difference?**
>
> **Reprocess** is inbound-only — it simply re-runs the parsing of the file already received, as long as it isn't already successful. **Retry** is outbound-only — it only appears once an exchange is in error, and fully regenerates the content from the source record before resending it, the button to use once the missing information has been fixed.

> 📌 **"Resend" — sending the exact same file again**
>
> Shown on a **successful** outbound exchange, "Resend" covers a different case than Retry: nothing went wrong on your side, but the partner asks for the file again (lost on their end, reprocessing issue...). It resends the exact same content, byte for byte, without regenerating it — so the partner receives precisely what was acknowledged the first time. A confirmation popup appears before sending, and a new exchange is created for it, keeping the original's success intact in the history.

Understanding an error

When the status is "Error", an extra tab appears with the explanatory message.



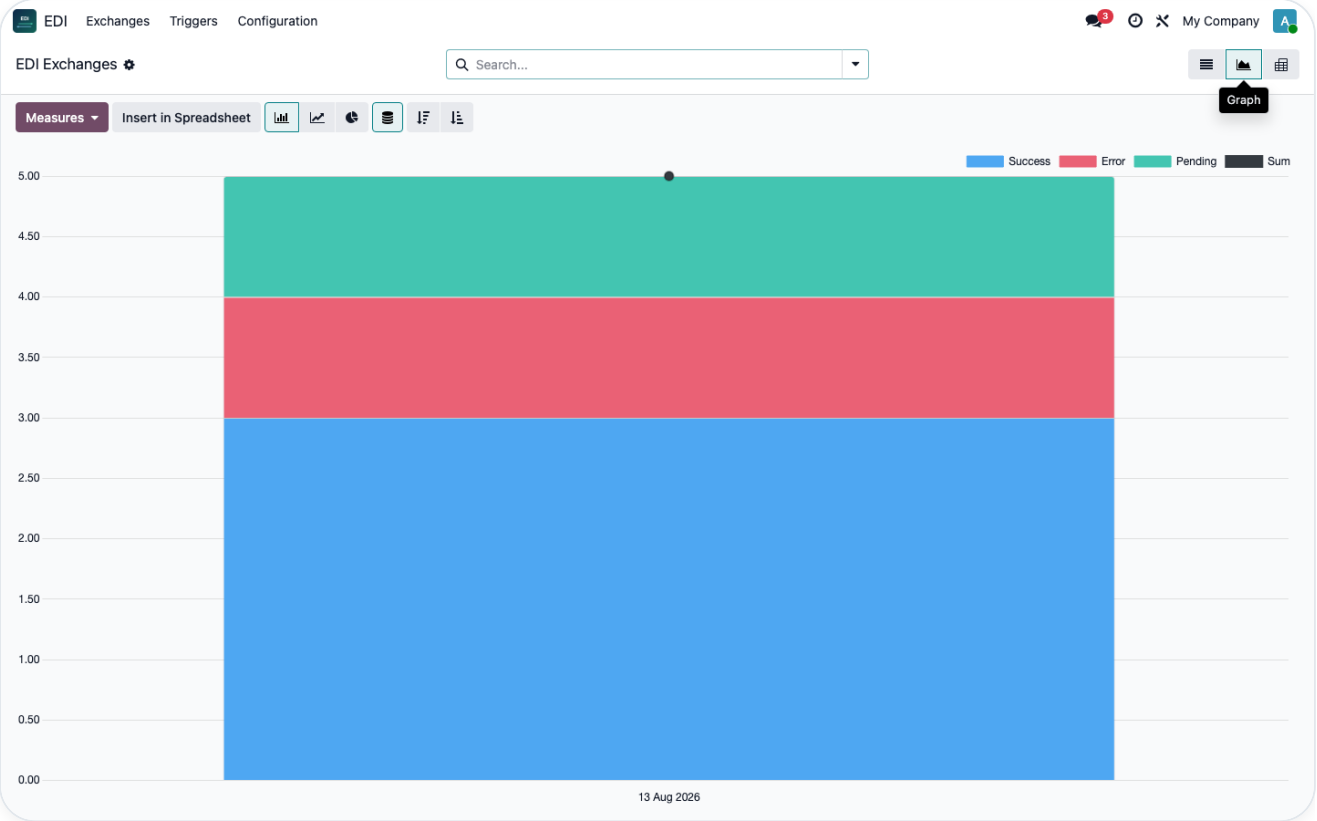
Details of an exchange in error, with the explanatory message

> ⚠️ **What to do in case of error?**
>
> The error message almost always gives the exact cause (e.g. "required field missing", "no partner found"). Once the cause is fixed (in Odoo, or by asking the partner for a correct file), relaunch it: **"Reprocess"** for an

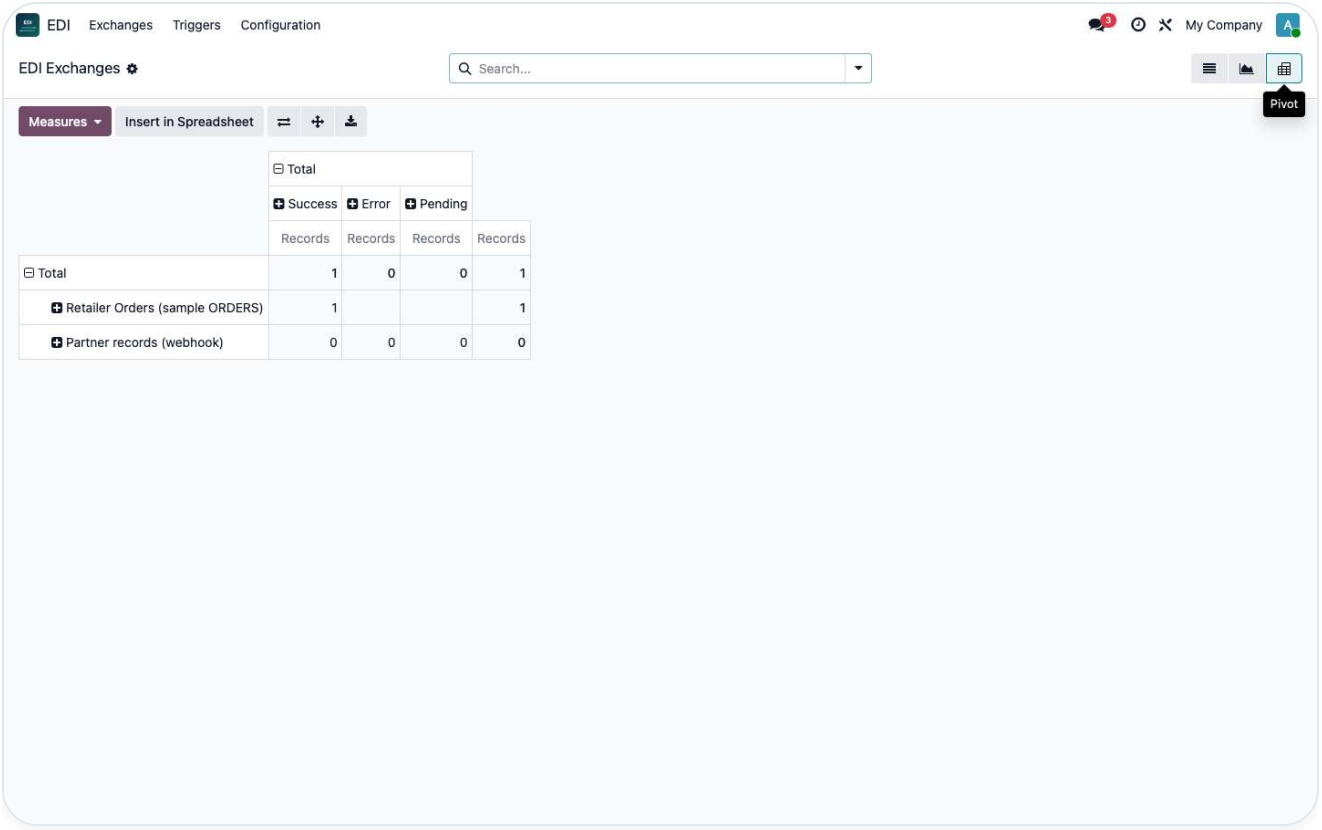
inbound file (shown here) — or **"Retry"** for an outbound one, which regenerates the file before resending it.

Graph and pivot views

To monitor volumes over time (for example, spotting a partner generating a lot of errors), two analysis views are available from the same icons at the top right of the list.



Graph view: exchange volumes by day and by status



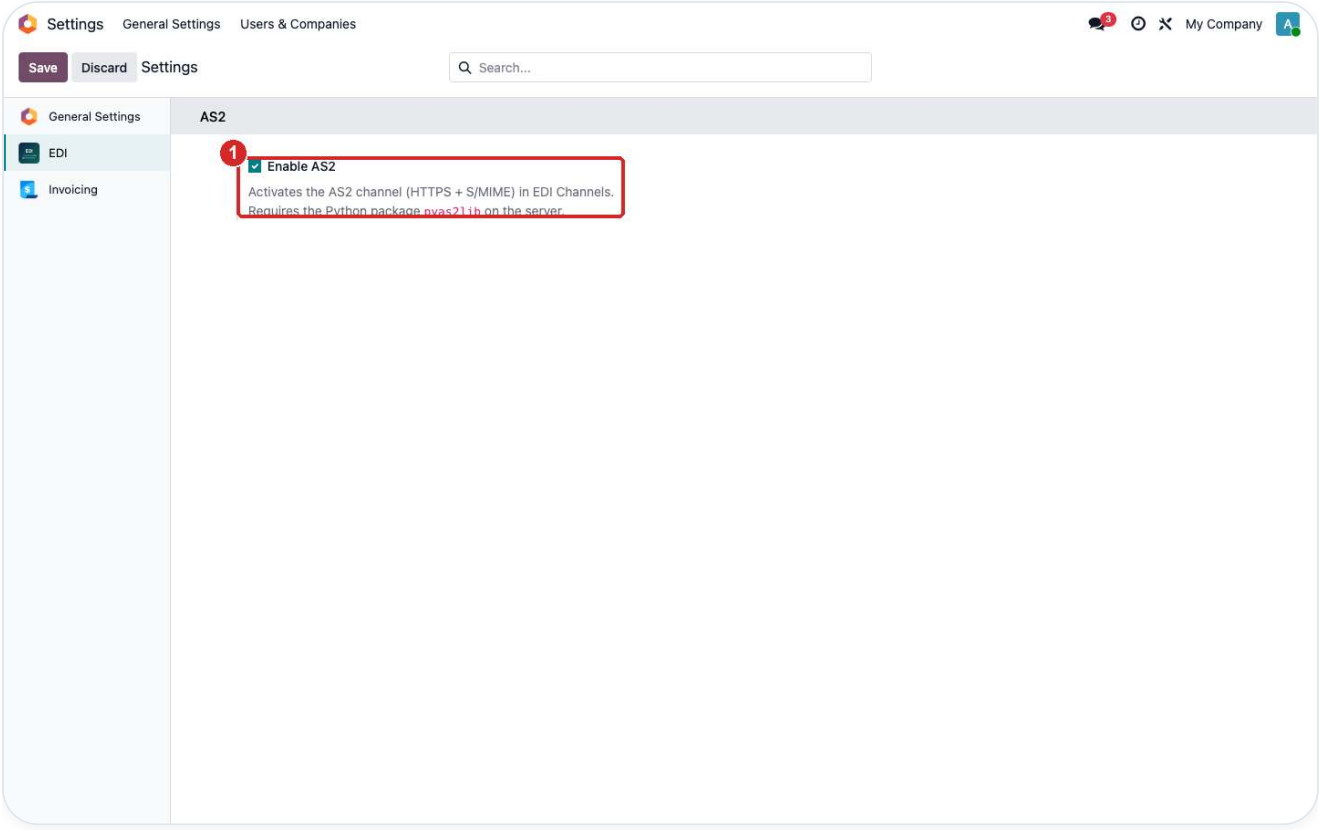
Pivot view: breakdown by profile and by status

8. The AS2 protocol in detail

AS2 is the protocol that most large retailers and manufacturers require from their suppliers. It adds three guarantees that other channels don't have natively:

- **Signature**: the file is digitally signed, which proves it really comes from you (and not a third party).
- **Encryption**: the content is unreadable to anyone intercepting it in transit.
- **Delivery receipt (MDN)**: the partner sends back cryptographic proof that they received and decrypted the file — legal proof in case of a dispute ("I never received your order").

> ⚠️ **Mandatory master switch**
>
> Before an AS2 channel can be used, the feature must be enabled in Odoo's general settings. Without this, all AS2 channels remain blocked, even if correctly configured.



Settings → EDI: the switch that enables AS2 across the whole module

1. This box must be checked and saved for AS2 channels to work.

Certificates

AS2 relies on **electronic certificates** — a bit like a digital ID card. Three are needed per channel:

Certificate	Role
Our certificate + our private key	Used to sign our messages and decrypt what the partner sends us.
Partner's certificate	Used to verify their signature and encrypt what we send them.
CA Certificate *(optional)*	If the partner uses a certificate signed by a recognized certificate authority rather than self-signed, this certificate lets you validate the whole trust chain.

These files are usually provided by the partner's IT department, or by yours if it already manages this type of certificate for other exchanges.

Synchronous or asynchronous mode

The delivery receipt (MDN) can arrive immediately in the same connection (**synchronous**, the simplest and most common), or be sent back separately a bit later by the partner (**asynchronous**) — in that case, the module automatically knows how to match this delayed receipt to the right exchange as soon as it arrives.

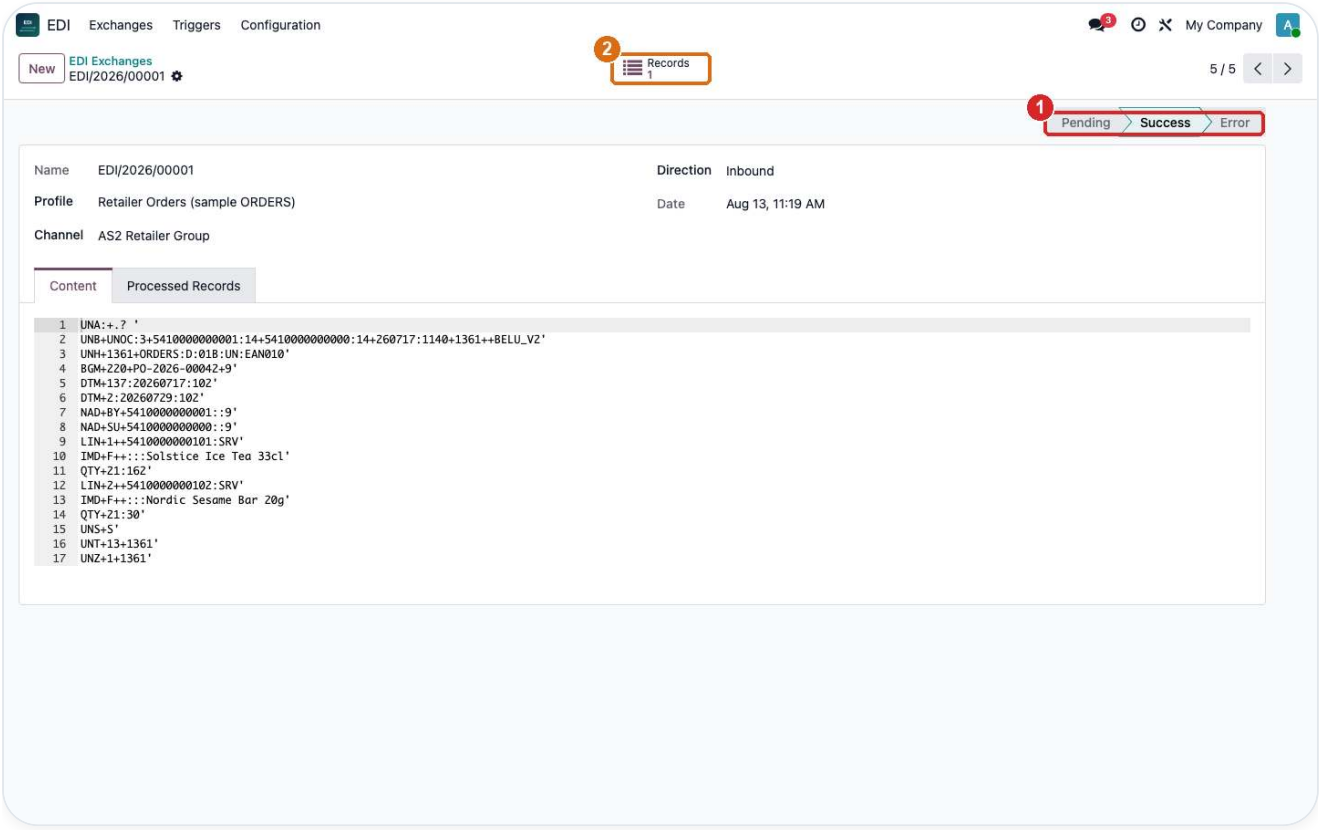
> 💡 **What is this matching based on?**
>
> Not on business data like the order reference — on a **technical AS2 protocol identifier** (the "Message-ID"), generated automatically when sending and independent of the file's content. When the partner sends their receipt back later, they return this same identifier; the module finds the corresponding exchange thanks to it, even if the file sent is encrypted and unreadable outside Odoo.

9. A complete end-to-end example

To make all this concrete, here is a representative example: a retailer sends an order in EDIFACT format, containing two products.

The file received

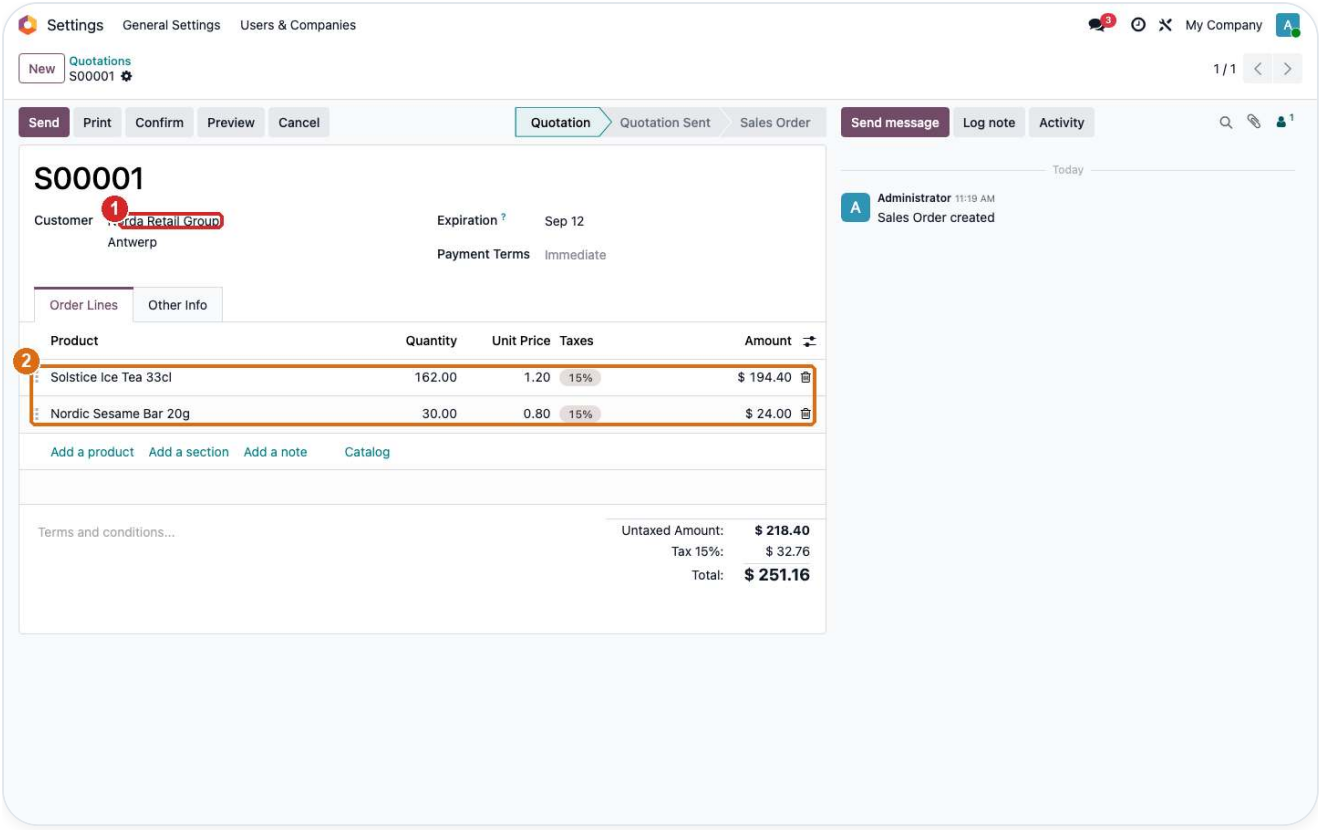
Here is, in an exchange's detail, the raw file as it arrives — unreadable to the naked eye, but perfectly structured for the module:



The raw EDIFACT file received from the partner

The result in Odoo

Thanks to the profile configured in section 5, this file automatically becomes **a sales order with its two lines**, the customer already identified:



The sales order automatically created, with its lines matching exactly the received file

1. The customer was found automatically thanks to their GLN identifier present in the file.
2. The two product lines, with the correct quantities (162 and 30), come directly from the LIN and QTY segments of the EDIFACT file.

> 💡 **No re-typing**
>
> Nobody typed the customer's name, nor "162", nor the product names. Everything comes from the file — the profile's whole purpose is to say, once and for all, where to find each piece of information.

10. Who can do what

The module defines two access levels, so operational teams can monitor exchanges without having rights to change the technical configuration.

Group	Can do	Cannot do
EDI User	View channels and profiles, view and retry exchanges	Modify a channel, a profile, or delete an exchange
EDI Manager	Everything, including creating/modifying channels and profiles	—

These groups are managed like any other Odoo access right, from a user's record.

11. FAQ and troubleshooting

An exchange stayed "Pending" — what should I do?

For an FTP/SFTP channel, this usually means the periodic automatic processing hasn't run yet — it's normal to wait a few minutes. You can also trigger it manually with the "Retry" button on the exchange, or "Trigger polling" on the channel.

I enabled AS2 but nothing happens

Check in order: (1) the switch in Settings → EDI is checked *and saved*, (2) the AS2 identifiers on both sides are filled in, (3) the certificates are properly uploaded, (4) the partner's remote URL is correct.

How can I know if my channel works before the first real send?

Use the "Test connection" button on the channel's form (available for Webhook, FTP and SFTP). For AS2, the first test is usually done with a real file sent by the partner in a test environment.

The same partner sometimes sends orders, sometimes invoices — how does the module tell the difference?

That's the role of the **root tag** / **message type** configured on each profile (section 5). If several profiles are attached to the same channel, the module looks at the file's content to automatically pick the right profile.

The partner changed their file format — do I need to redo the whole configuration?

No — you just need to adjust the relevant mapping rows (the "Source Path") in the existing profile. No code to change.

Can I see exactly what was sent to a partner?

Yes, every outbound exchange keeps the exact content of the file sent, viewable at any time in its detail (section 7).

How do I know which format to choose for a new partner?

It's not a choice — it's the format already used by the partner that dictates the profile's format. Open a real sample file: tags `<...>` → XML, braces `{...}` → JSON, blocks `BGM+...'` → EDIFACT, columns separated by a comma or semicolon → CSV, and if none of that but every line always has exactly the same length → fixed-width.

12. Full glossary

AS2

Secure EDI transport protocol (signature, encryption, delivery receipt), widely required in retail.

Channel (edi.channel)

The configuration of the means of transport for an EDI file (Webhook, FTP, SFTP, AS2, Manual).

CA Certificate

Certificate from a recognized certificate authority, used to validate that a partner certificate is genuine and not merely self-signed.

Upsert Key

Field used to recognize whether a record already exists, in order to update it rather than create a duplicate.

CSV

Tabular file format where the values on a row are separated by a character (comma, semicolon...), usually with a first row of column headers.

EDI

Electronic Data Interchange — the automated sending of business documents between computer systems, without manual re-entry.

EDIFACT

Historic international EDI exchange standard, organized into segments (e.g. BGM, NAD, LIN), heavily used in commerce and industry.

Exchange (edi.exchange)

The record of a file received or sent, with its content, status and history.

GLN

Global Location Number — a unique international identifier assigned to each company or site, used to automatically identify partners in EDI files.

JSON

Text-based data format structured as keys/values, heavily used by modern web applications.

Fixed-width

Text file format with no separator, where each piece of information always occupies the same columns; each line usually starts with a short code stating its type. Format inherited from legacy "mainframe" systems, still used by some retailers.

Mapping (edi.field.mapping)

A row of correspondence between a location in the source file and an Odoo field.

MDN

Message Disposition Notification — the cryptographic delivery receipt sent back by the partner in an AS2 exchange.

Profile (edi.profile)

The configuration describing how to read or write a type of document in a given format, and to which Odoo model.

Qualifier (EDIFACT)

A code specifying the role of a segment repeated several times in the same file (e.g. distinguishing the buyer's address from the supplier's, both in NAD segments).

Segment (EDIFACT)

A block of information in an EDIFACT file, identified by a 3-letter code (BGM, NAD, LIN, QTY...).

Transformer

The conversion applied to a read value before saving it in Odoo (number, date, lookup of an existing record...).

Webhook

A web address to which an external system can automatically send data, or that Odoo can call to send data outward.

XML

Text-based data format structured as nested tags, widely used in business-to-business exchanges.

📩 A question? A problem?

Contact the team that built this module — we will get back to you within 8 business hours.

Email: cma@nalios.be

Nalios — Odoo Gold Partner, Belgium — nalios.be