LENIC EDI CONVERTER

User Manual

Setting up and operating · the mapping · the definition language

Version 1.15 · As of 2026-11-02

Ljuban Magdelinic, IT Consulting

Contents

1 The converter in operation

1.1 System requirements
1.2 Language
1.3 Installation
1.4 Requesting and installing a license
1.5 First start
1.6 Supplied samples
1.7 Updating to a new version
1.8 The editor
1.9 ZUGFeRD invoices
1.10 Data from SQL Server (SQL-LINK)
1.11 Calling the converter in production
1.12 Return values
1.13 Log, run log and history
1.14 Archive, restore and deletion protection
1.15 Concurrent runs
1.16 Backup
1.17 Different paths
1.18 E-mail via Microsoft 365
1.19 Troubleshooting and frequently asked questions
1.20 Uninstallation
1.21 Contact

2 The mapping

2.1 What a mapping is
2.2 The record description
2.3 Both directions from one description
2.4 A run to follow along
2.5 Despatch advice: shipment, pallets, line items
2.6 Invoice: parties, discounts, taxes
2.7 Continuing the draft by hand

3 The definition language

3.1 Structure of a definition
3.2 Values
3.3 GLOBAL-CONF
3.4 SOURCE-CONF
3.5 TARGET-CONF
3.6 START-CONF, RUN-CONF, END-CONF
3.7 Calling and return values
3.8 Complete example

PART 1

The converter in operation

Setting up, operating, running

LeNic EDI Converter converts files between your ERP system and your business partners – in both directions:

- **Outbound:** your inhouse data becomes an EANCOM/EDIFACT message, for example an invoice (INVOIC).
- **Inbound:** an EDIFACT message from your partner becomes inhouse data, for example a purchase order (ORDERS) as a CSV file for your ERP system.

It consists of two programs that share the database and the license:

- The **editor** is the user interface. This is where you maintain partners, configurations and converter definitions, and where you try out a conversion.
- The **converter** is the program that runs in production. It is called with exactly one parameter – the number of a saved configuration – and needs no user interface.

The product is called "LeNic EDI-Konverter" in German; program files, folders and the installer keep that name in both languages.

1.1 System requirements

- Windows 10 or Windows 11 (64-bit)
- **.NET 10 runtime**, namely the **Windows Desktop Runtime** (x64); it also contains the runtime for the converter. The setup program brings it along and installs it if needed.
- A valid license for the computer you work on (see "Requesting and installing a license").
- Administrator rights for the installation; in production a normal user account with write access to the folders that are read from and written to is sufficient.

You do not need to set up a database: the converter comes with SQLite and creates its file itself. Only if you read data from your ERP system with `SQL-LINK` do you need access to its Microsoft SQL Server (see "Data from SQL Server").

1.2 Language

Editor and converter speak **German** and **English**. Without a setting they follow the Windows display language – German on a German Windows, English on any other.

You change the language in the editor under **Settings** → **Language**: "Like Windows", "Deutsch" or "English". The setting is stored in the database and therefore applies to the editor and the converter alike:

- The **converter** uses it from the next run on – for messages, findings and the log file.
- The **editor** asks whether it should restart right away; the new language applies after the restart.

What is stored in the database stays as it was written: the run log keeps the messages in the language of the run. The definition language itself – `IF`, `WRITE`, `NAD+BY+VALUE` … – is the same in both languages.

1.3 Installation

The software is delivered in two forms, each in German and in English. The English files carry `-en` in their name; their installation wizard and license dialog are in English. Both languages install the same program and replace each other on an update.

- `LeNicEdiKonverter-<version>-Setup-en.exe` – the setup program. It brings the .NET Desktop Runtime 10 along and installs it before the program if it is missing. No internet connection is needed. For most installations this is the easiest way.
- `LeNicEdiKonverter-<version>-x64-en.msi` – the installation package on its own, for example for distribution via software management. It requires the .NET Desktop Runtime 10 and aborts with a message if it is missing.

Which file?

Both install the same program into the same folder – the setup program contains the MSI package and runs it itself. The difference is the .NET runtime:

	Setup program (`Setup.exe`)	Installation package (`x64.msi`)
Size	about 70 MB	about 12 MB
Contents	MSI package and .NET Desktop Runtime 10	only LeNic EDI Converter
.NET 10 missing	installed first, without internet	aborts with a message, install .NET 10 first
Intended for	manual installation on one computer	central distribution (Group Policy, Intune, SCCM) where .NET 10 is already present
Unattended	`/quiet InstallFolder=…`	`msiexec /i … /qn INSTALLFOLDER=…`
German version	`…-Setup.exe`	`…-x64.msi`

Both update an existing installation – for example 1.3 to 1.4 – in its previous folder; database, license and your files are kept.

Rule of thumb: if you install yourself, use the **setup program**. The **MSI package** is for administrators who distribute software centrally and already have the .NET Desktop Runtime 10 or distribute it separately.

Microsoft Defender SmartScreen warning

The packages are not digitally signed yet. Windows therefore warns when downloading and on first start – this is common for new, unsigned software and says nothing about the file itself.

- **Edge blocks the download:** open the download list (Ctrl+J), click **"..."** next to the file → **"Keep"** → **"Show more"** → **"Keep anyway"**.
- **Windows prevented the start: "More info"**, then **"Run anyway"**.

Whether the file arrived unchanged is shown by its checksum. It is listed in `SHA256SUMS.txt` next to the downloads and can be compared in PowerShell:

```
Get-FileHash .\LeNicEdiKonverter-1.9.1-Setup-en.exe -Algorithm SHA256
```

The wizard

Start one of them and follow the wizard:

- Read and accept the **license agreement**.

- **Destination folder**: `C:\Tools\LeNicEdiKonverter` is suggested; in the setup program you change it under **Options**. For an update it is the folder of the previous installation.
- **Install** – Windows asks for administrator rights.

Afterwards you will find the **editor** and the **user manual** in the Start menu under "LeNic EDI-Konverter".

For an unattended installation, for example via software distribution:

```
LeNicEdiKonverter-<version>-Setup-en.exe /quiet InstallFolder=D:\EDI\LeNicEdiKonverter
msiexec /i LeNicEdiKonverter-<version>-x64-en.msi /qn INSTALLFOLDER=D:\EDI\LeNicEdiKonverter\
```

The installation folder

The destination folder is called the **installation folder** below. It contains:

```
C:\Tools\LeNicEdiKonverter\          ← installation folder
    Konverter\LeNic.EdiKonverter.App.exe    ← the program for production
    Editor\LeNic.EdiKonverter.Wpf.exe       ← the user interface
    data\LeNic.EdiKonverter.db              ← database, with the samples
    Lizenz\license.dat                      ← your license file
    temp\                                   ← logs and locks
    Import\  INHOUSE\  Export\  Archiv\       ← input and output of the samples
    Beispiele\Beispiele.db                  ← the samples of this version
    User manual (PDF and Markdown, German and English)
    LICENSE.md, THIRD-PARTY-NOTICES.md (and their English versions)
```

The folder names are the same in both languages: `Konverter` is the converter, `Lizenz` the license, `Archiv` the archive and `Beispiele` the samples.

Both programs find their installation folder themselves: it is the folder **above** the program folder. You do not need to enter anything as long as you keep this structure. Different paths are possible, see "Different paths".

All users of the computer may write to `data\`, `Lizenz\`, `temp\`, `Import\`, `INHOUSE\`, `Export\` and `Archiv\` – editor and converter need this, even without administrator rights. The program folders `Konverter\` and `Editor\` remain read-only.

Your data belongs to you: the setup only creates the database and the test data if they do not exist yet. Neither a new version nor the uninstallation touches them – nor `Lizenz\` and your files in the input and output folders.

1.4 Requesting and installing a license

LeNic EDI Converter requires a license. A license applies **to one specific computer** and up to an expiry date.

- The editor tells you the computer name the license must carry: it is shown in the license window at start-up and under **Settings → Data & license → License**.
- Request the license from the licensor, stating the computer name and the product **"LeNic EDI-Konverter"**. A license for LeNicScheduler does not apply here.
- The file is called `license.dat` and belongs in the `Lizenz\` folder of the installation folder. It is more convenient in the editor: **Settings → License → "Install license file ..."**. If the editor does not start at all for lack of a license, it offers the installation right in the license window.
- An invalid file **never** replaces an existing valid license.

Without a valid license the editor does not start and the converter converts nothing; it ends with return value 13.

Modules

A license consists of the **base**; three **modules** are added as needed, each with its own expiry date:

Part	Includes
Base	CSV, text and fixed record length – in both directions
EDIFACT module	EDIFACT as input or output – recognised by `FORMAT = EDIFACT` and by the content of the file (UNA, UNB or UNH)
JSON/XML module	JSON and XML as source or target
ZUGFeRD module	output as ZUGFeRD PDF, including the XML invoice in it – this does not need JSON/XML

Which modules your license includes and until when is shown under **Settings** → **Data & license** → **License**. If a run needs a module that is missing or has expired, the converter converts nothing and ends with return value 13; the message names the module. All definitions can still be edited and tried out with the **test run** without a module. Messages and findings are then shown in full, but only the beginning of the output – at most 20 lines or EDIFACT segments; the same applies to the trace.

A license from before modules existed remains valid – with EDIFACT and JSON/XML, but without ZUGFeRD. Conversely, a license with modules only works from version 1.10; an older version rejects it. So update first, then install the new license.

Trying out ZUGFeRD: in the test run on the "Converters" page, **"View as ZUGFeRD PDF"** shows the output as an invoice. Without the ZUGFeRD module it is a **sample** – with a watermark and without the embedded XML, so not an e-invoice that could be sent.

1.5 First start

Start the editor. The delivery includes a database with working samples (see below); if it is missing, the editor creates an empty one under `data\` on first start. For a partner of your own, set up the following in this order:

- create a **partner** (company name and GLN),
- write a **converter** or have it generated from a record description,
- create a **configuration** for this partner that brings both together.

1.6 Supplied samples

So that you can see right away what a run looks like, the delivery includes a sample partner with eleven configurations and matching test data. The partner is "Magdelinic Consulting" with the test GLN `9999999999994`; the purchase orders, despatch advices and invoices belong together (purchase orders PO-2026-100234 and PO-2026-100235).

No.	Converter	Direction	Input	Output
3	ORDER-INBOUND	purchase order in	`{BASIS}\Import\ORDER*.EDI`	`{BASIS}\INHOUSE\ORDER_{ZAEHLER}.CSV`
4	ORDER-OUTBOUND	purchase order out	`{BASIS}\INHOUSE\ORDER_OUT.CSV`	`{BASIS}\Export\ORDER_OUT_{ZAEHLER}.EDI`
5	DESADV-OUTBOUND	despatch advice out	`{BASIS}\INHOUSE\DESADV_OUT.CSV`	`{BASIS}\Export\DESADV_OUT_{ZAEHLER}.EDI`
6	INVOIC-OUTBOUND	invoice out	`{BASIS}\INHOUSE\INVOIC_OUT.CSV`	`{BASIS}\Export\INVOIC_OUT_{ZAEHLER}.EDI`
7	ZUGFERD-CSV	invoice as ZUGFeRD PDF (EN 16931)	`{BASIS}\INHOUSE\ZUGFERD_OUT.CSV`	`{BASIS}\Export\ZUGFERD_OUT_{ZAEHLER}.PDF`
8	ZUGFERD-CSV	collective invoice as ZUGFeRD PDF (EN 16931)	`{BASIS}\INHOUSE\ZUGFERD_SAMMEL.CSV`	`{BASIS}\Export\ZUGFERD_SAMMEL_{ZAEHLER}.PDF`
9	ZUGFERD-INVOIC	EANCOM invoice as ZUGFeRD PDF (EXTENDED)	`{BASIS}\INHOUSE\ZUGFERD_INVOIC.EDI`	`{BASIS}\Export\ZUGFERD_INVOIC_{ZAEHLER}.PDF`
10	ORDER-INBOUND-XML	purchase order in, as XML	`{BASIS}\Import\XML_ORDER*.EDI`	`{BASIS}\INHOUSE\ORDER_{ZAEHLER}.XML`
11	ORDER-INBOUND-JSON	purchase order in, as JSON	`{BASIS}\Import\JSON_ORDER*.EDI`	`{BASIS}\INHOUSE\ORDER_{ZAEHLER}.JSON`
12	ORDER-OUTBOUND-XML	purchase order out, from XML	`{BASIS}\INHOUSE\ORDER_OUT.XML`	`{BASIS}\Export\ORDER_OUT_XML_{ZAEHLER}.EDI`
13	ORDER-OUTBOUND-JSON	purchase order out, from JSON	`{BASIS}\INHOUSE\ORDER_OUT.JSON`	`{BASIS}\Export\ORDER_OUT_JSON_{ZAEHLER}.EDI`

`{BASIS}` is the installation folder – so the samples run in any destination folder without you having to adjust paths.

The folders are located in the installation folder:

- `Import\` – what comes in from partners (EDIFACT),
- `INHOUSE\` – your inhouse files, in both directions,
- `Export\` – what goes out to partners (EDIFACT),
- `Archiv\` – processed input files.

How to try it out: on the "Partners" page select the sample partner and click **Run** next to one of its configurations. The output is created in the `Export\` or `INHOUSE\` folder, and the history shows the run. Note: the samples **archive** – after the run the input file is moved to `Archiv\`. To repeat the same run, bring it back with **Archive** → **Restore**. To try without output, use the **Test run** on the "Converters" page.

If you have deleted samples, or an update brings a new one, get them back under **Settings** → **Data & license** → **Samples** → **"Add samples ..."**. Only what is missing is added – partners by GLN, converters by name, configurations by converter and input file – with counter 0.

The four EDIFACT converters were generated from the supplied record descriptions (see "From record description") and are a good starting point for your own – copy, rename, adapt. How purchase order, despatch advice and invoice are built in detail is shown in the part "The mapping" with the inhouse lines and the finished message. The three ZUGFeRD samples turn an invoice into a ZUGFeRD PDF, see "ZUGFeRD invoices": **ZUGFERD-CSV** reads the in-house CSV – a simple invoice in configuration 7, a collective invoice covering two delivery notes in 8 –, **ZUGFERD-INVOIC** an EANCOM invoice. Each carries its **output layout** – the invoice as XML – itself, see "Output layouts".

Configurations 10 to 13 show the same purchase order as 3 and 4, but with an **XML or JSON file** instead of the in-house CSV. Both formats have the same structure: `bestellungen` with one `bestellung` each (number, order date and delivery date as YYYY-MM-DD, `partner` for buyer and delivery address, `position` for each line). The partners are told apart by `typ` – in JSON an entry of its own, in XML an attribute (`<partner typ="DP">`). **ORDER-INBOUND-XML** and **-JSON** write each message of the EDIFACT file as a purchase order of its own. **ORDER-OUTBOUND-XML** and **-JSON** read one purchase order per file and create the same EANCOM purchase order (D.01B) as ORDER-OUTBOUND from it; if the file holds a second purchase order, the run stops with a message. The definitions are also in `Konverter\Samples\` . The inputs are named `XML_ORDER.EDI` and `JSON_ORDER.EDI` so that configuration 3 (`ORDER*.EDI`) does not pick them up.

1.7 Updating to a new version

Close the editor and the converter and start the new installation package. It replaces the previous version in the same folder – even if you chose a folder other than the suggested one during the first installation. Only the program folders `Konverter\` and `Editor\` , the manual and the license texts are replaced. **Your data stays:** database (`data\`), license (`Lizenz\`) and your input and output folders are not touched. The setup refuses to install an older version over a newer one.

New samples of a version therefore do not reach your database by themselves. Add them if needed in the editor under **Settings** → **Data & license** → **Samples** → **"Add samples ..."**: only what is missing is created; converters and configurations you already have or have changed remain untouched.

If the new version needs a change to the database, this happens automatically when it is first opened:

- Beforehand the program places **a complete backup** next to it, named like `LeNic.EdiKonverter.vor-Schema1-20260919-115102.db` .
- Afterwards the editor reports "Database updated" at the top and names the backup.
- If two programs open an old database at the same time, the second one waits; the change is made exactly once. If something fails, everything stays as it was.

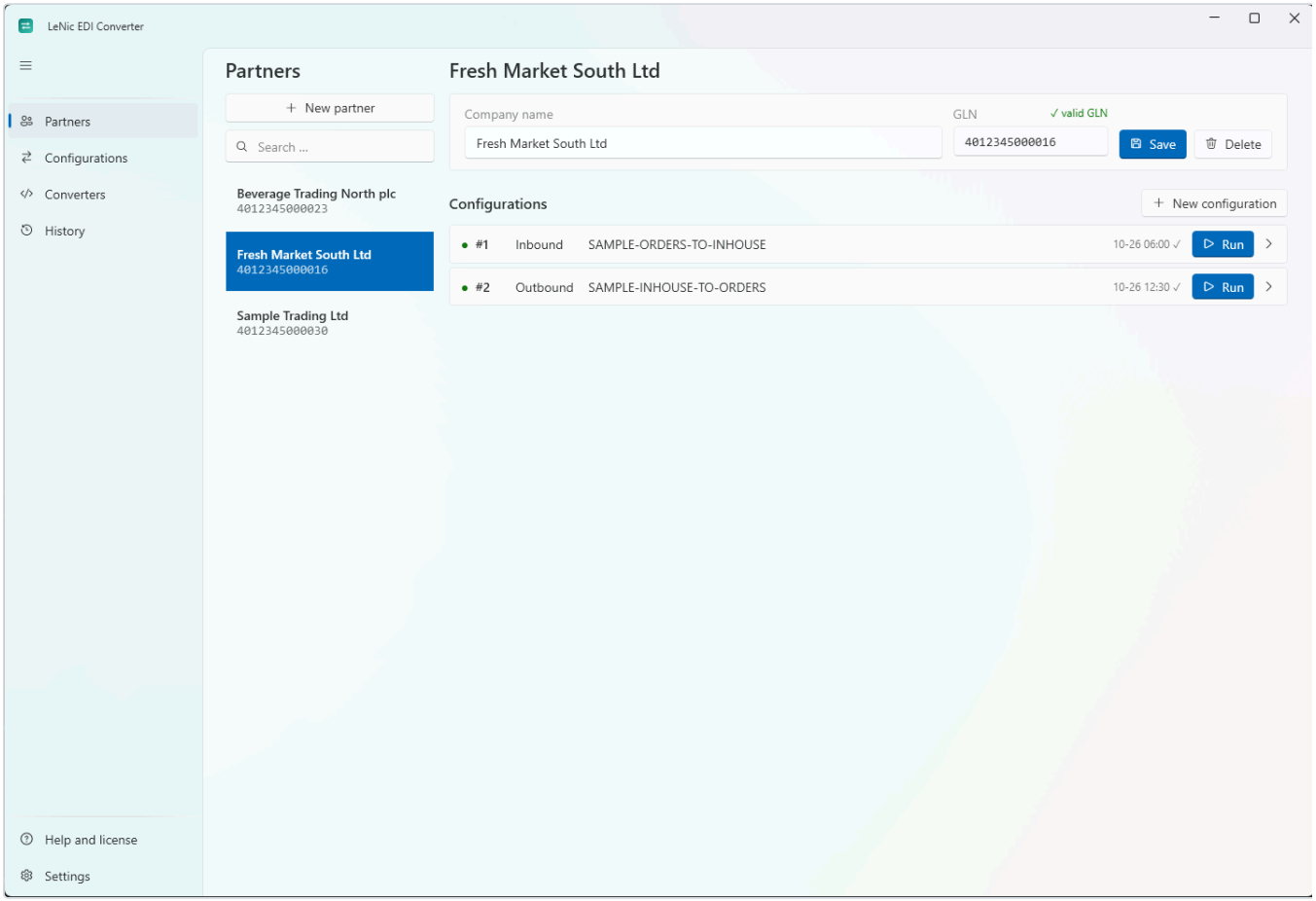
A database created by a **newer** program version is rejected – in that case the program needs updating, not the database.

If scheduled tasks call the converter, pause them for the duration of the update.

1.8 The editor

On the left you choose the page: **Partners**, **Converters**, **History**, and at the bottom **Help and license** and **Settings**.

"Partners" page



The partners page: on the left the partners, on the right the selected partner with one row per configuration. The dot shows the last run, "Run" executes a configuration immediately.

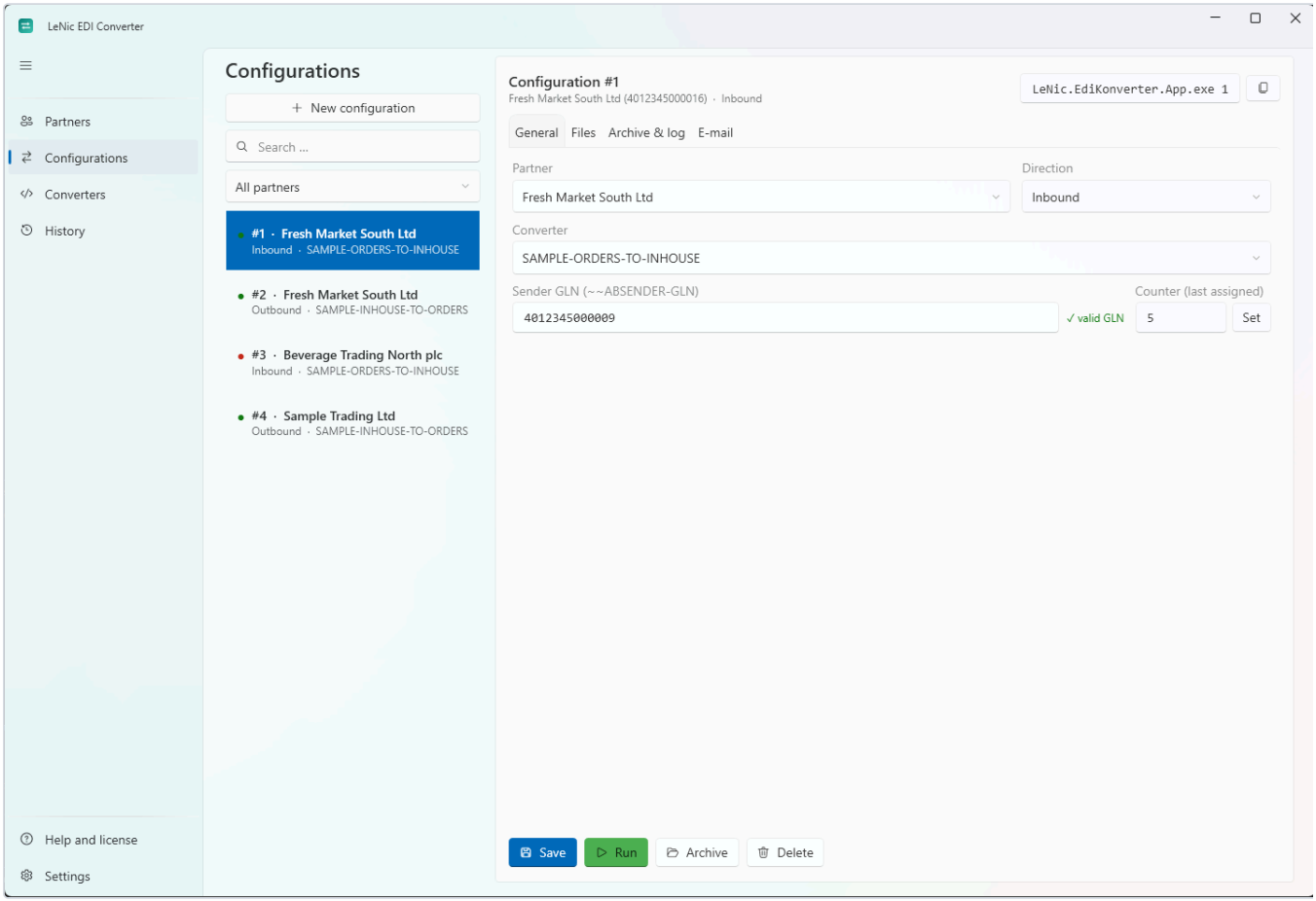
Work starts from the partner: you select a partner on the left and see its details and its configurations on the right.

- **Company name** – what the partner is called in your company.
- **GLN** – the partner's 13-digit Global Location Number. The editor checks it immediately and, after twelve digits, tells you the missing check digit. Each GLN may only be assigned once.

Configurations exist only for a **saved** partner. Each is shown in one row: a coloured dot for the last run (green successful, yellow with warnings, red failed, grey none yet), number, direction, converter and the time of the last run. The tooltip shows input and output file.

- **Run** executes the configuration immediately – the same run as in production.
- Clicking the row opens it on the "Configurations" page; "New configuration" creates one for this partner there.

"Configurations" page



> The configurations page: on the left all configurations, on the right the selected one with the command line for the Task Scheduler and its fields on four tabs.

On the left are **all** configurations with partner, direction, converter and the coloured dot of the last run. The search finds number, partner, converter and file names; below it you can filter by partner. On the right is the selected configuration: at the top number, partner and the **command line** for the Task Scheduler with a button to copy it, below that the fields on four tabs, at the bottom **Save**, **Run**, **Archive** and **Delete**. "Run" and "Archive" save first. If partner or converter is missing or a number of days is not a number, saving switches to the tab with that field.

"General" tab

- **Partner** – the partner the configuration belongs to. It can also be changed here.
- **Direction** – "Outbound" (inhouse → EDIFACT) or "Inbound" (EDIFACT → inhouse).
- **Converter** – which definition is used.
- **Sender GLN** – your own GLN, usable in the definition as `~~ABSENDER-GLN`. Only needed if the definition uses it; the check digit is checked on entry.
- **Counter (last assigned)** – the number used last. Saving does not change it; only "Set" explicitly writes a new value, for example after the turn of the year.

"Files" tab

- **Input file** – the full path of the file that is read. `{BASIS}` stands for the installation folder: `{BASIS}\Import\ORDER*.EDI` reads from the `Import` folder of the installation, wherever it is. The same applies to the output file and the archive and log folders. Instead of a fixed name a **pattern** may be used here if your partner delivers under changing names:
  - `*` stands for any number of characters: `C:\EDI\in\INVOIC_*.csv`
  - `?` stands for exactly one character: `C:\EDI\in\ORDERS_??????.edi`

Wildcards are only allowed in the file name, not in the folder – where to look must be fixed. **One run processes one file**, namely the **oldest** matching one; this preserves the order of delivery. If several are waiting, the message says how many – simply call the converter again until it reports return value 5, meaning nothing is left. Below the field the editor shows which file would be next.

A pattern needs archiving (see below): only if the processed file is moved away is the next one processed in the next run. If it stays, every run takes the same file again; the editor warns about this.

- **Input character set** – "Automatic" (default, detects it from the content), "Latin-1" or "UTF-8". UTF-16 files are rejected.
- **Output character set** – "Latin-1" (default), "UTF-8" (without byte order mark) or "Automatic", i.e. as the input was read. In Latin-1 every character becomes exactly one byte; a character that does not exist there – such as the euro sign – aborts the conversion instead of silently writing a question mark.
- **Output as** – "File, as the definition writes it" (default) or "ZUGFeRD PDF": the XML invoice of the definition then becomes a visible invoice as PDF/A-3 with the XML embedded, see "ZUGFeRD invoices".
- **Output file** – a **pattern** with placeholders:
  - `{ZAEHLER}` – the running number of this configuration (counter): 1, 2, 3 …
  - `{ZAEHLER:6}` – the same number padded with zeros to six digits: `000001`
  - `{DATUM}` – today's date `YYYYMMDD`
  - `{ZEIT}` – the time `HHMMSS`
  - `{RECHNUNGSNUMMER}`, equivalently `{RG-NO}` – the number of the invoice, such as `INVOICE_{RECHNUNGSNUMMER}.pdf`. Only for an invoice: EDIFACT INVOIC (number from the `BGM`) or ZUGFeRD (from the XML invoice). With any other output – or with several invoices with different numbers in one file – the run ends with return value 7 and writes nothing. Characters not allowed in a file name, such as `/`, become `_`.
  - `{BASIS}` – the installation folder

  The placeholders are written the same way in both languages. Example: `C:\EDI\out\INVOIC_{DATUM}_{ZAEHLER:6}.edi`. Below the field you see a preview of the next name. Folders from the pattern are created. If the pattern contains `{ZAEHLER}`, the output is always created; a number already in use is skipped. Without `{ZAEHLER}` an existing file is **not** overwritten – the run then ends with return value 6. This also applies to the same invoice a second time.

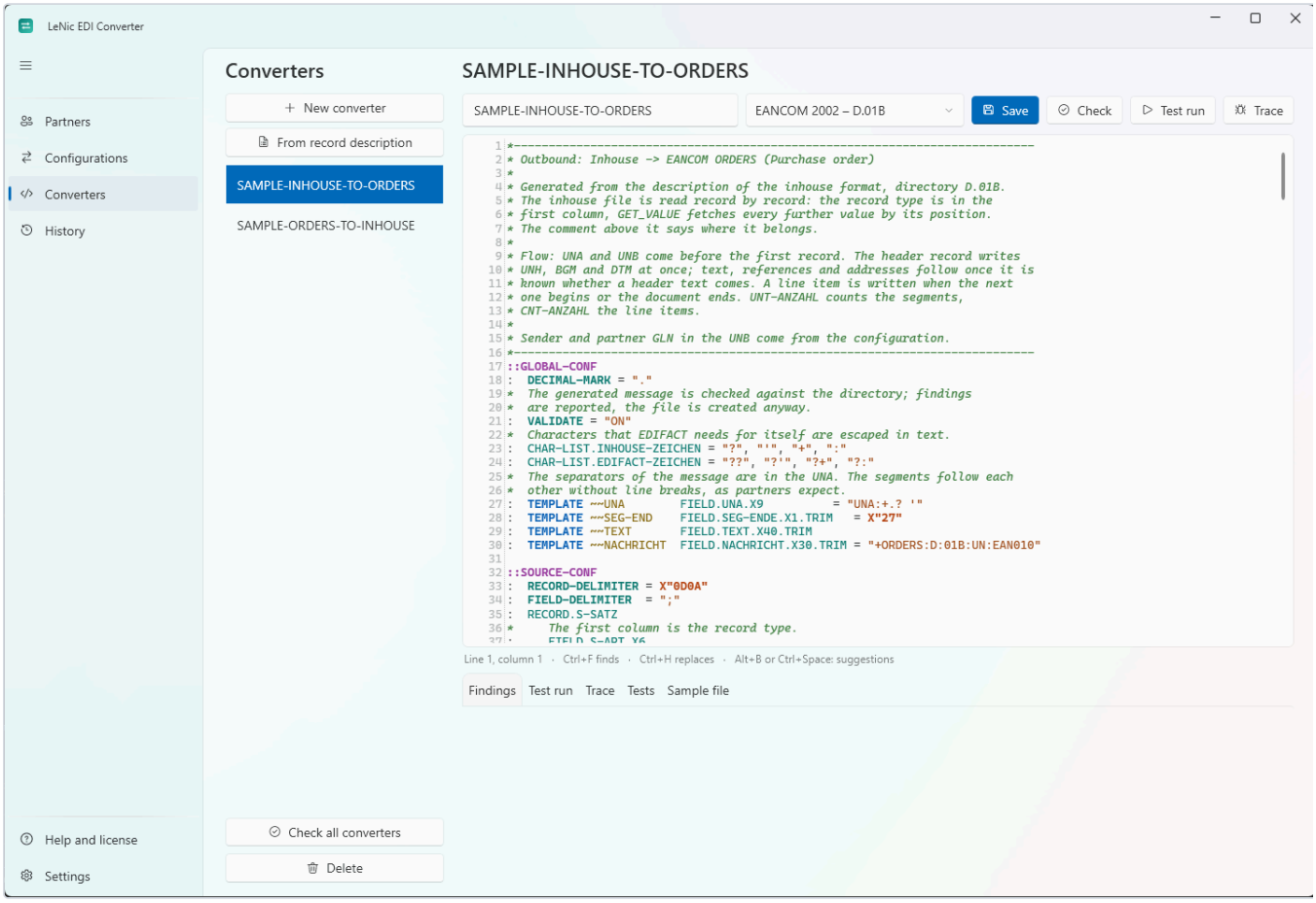
"Archive & log" tab

- **Archive**
  - "Archive the input file after successful processing" – moves the file that was read into the archive after the run, named like `ORDERS_20260919_115102.edi`.
  - "Archive folder" – empty means: a folder `Archiv` next to the input file. A relative path starts from there.
  - "Days" – how long archive files of this input are kept; `0` means forever.
  - "Also delete archive files outside the base directory" – see "Deletion protection". Without this tick, old archive files outside the installation folder simply stay where they are.
- **Log** – folder and retention in days for the `.LOG` file of each run. Without a value the converter writes it to `temp\` in the installation folder.

"E-mail" tab

- **E-mail notification** – "Send e-mail" *Never*, *Always* or *On error*, plus the recipients, separated by `;`. *Always* reports every run, *On error* only one that was not successful (return value neither 0 nor 11). A run that finds no input file (return value 5) never sends an e-mail – otherwise every Task Scheduler call would send one. The e-mail states configuration, partner, computer, return value, input and output file and the message of the run. It is sent as set under Settings → E-mail delivery; whether it went out is shown in the message and in the history. A failed delivery does not change the return value.

"Converters" page



The converters page with a definition. The colours show sections, commands, fields and patterns.

This is where you write the definition, i.e. the rules for how a conversion works. The language is described in the "Language reference" (`docs/LanguageReference.md`).

- **New converter** starts with a small, working template.
- **From record description** generates a **draft** without you having to write a single line of the definition language. You only describe your inhouse format – one line per record type with the record ID and the column names:

  ```
  K;OrderNumber;OrderDate;DeliveryDate;Buyer;ShipTo;Name;Street;Postcode;City;Country;
  P;LineNumber;GTIN;ArticleNumber;ArticleDescription;Quantity;Unit;
  E;NumberOfLines;
  ```

  For **purchase order, despatch advice and invoice** such a template is in the field when it opens; if you switch the message type, it switches too, as long as you have not entered anything of your own. With **Choose file …** you read a description from a file.

  The editor knows by itself where the values come from: the segments and qualifiers of an EANCOM message are standardized, and your column name says what is meant. "Buyer", "Customer" and "Buyer GLN" mean the same thing, and so do German names such as "Käufer" or "Besteller". From this it writes the complete definition – header, line items, text records, trailer record, including the comparison of the number of line items with the `CNT` segment. The comments in the generated definition are written in the language that is set.

  At the top you choose **Direction**, **Directory** (the UN/EDIFACT release, for example EANCOM 2002 – D.01B) and **Message type** (the message types of the selected directory) in three drop-down lists.

  The same description serves both directions:

  - **Inbound** – reads the EDIFACT message and writes your inhouse file.

- **Outbound** – reads your inhouse file and writes the message: envelope (`UNA` , `UNB` , `UNZ`), message (`UNH` … `UNT`), header, line items and summary section. The definition counts the number of segments in the `UNT` and of line items in the `CNT` while writing; sender and partner GLN in the `UNB` come from the configuration. Free texts are escaped so that an apostrophe in an article description does not end the segment.

What the name does not reveal is not guessed: the field is created, its assignment is commented out, and the message names the column. The draft arrives in the editor **unsaved** – give it a name, complete it, check it, save it.

A proposal is available for these message types, in every supplied directory that contains them:

Message type	Document	Quantity	Special features
ORDERS – purchase order	`BGM+220`	`QTY+21`	The order number is the document number.
INVOIC – invoice	`BGM+380`	`QTY+47`	Reference to the order (`RFF+ON` , `DTM+171`) and delivery note (`RFF+DQ`), as a collective invoice also per line item (`RFF+DQ` , `DTM+171`); buyer, delivery address, supplier and invoicee each with address and VAT ID (`NAD` , `RFF+VA`); currency, price (`PRI`), line amount (`MOA+203`), tax rate (`TAX`), a tax rate in the header as default; discounts on document and line item (`ALC` , `PCD` , `MOA+204`); in the summary section invoice amount, goods value, taxable amount and tax amount (`MOA+77` , `79` , `125` , `124`) and one tax line per tax rate.
DESADV – despatch advice	`BGM+351`	`QTY+12`	Dispatch and delivery date (`DTM+11` , `DTM+17`); the shipment (`CPS+1`) with number of pallets and total weight, each pallet its own level with weight and SSCC (`GIN+BJ`), below it the line items with batch and best-before date (`GIN+BX` , `DTM+361`). Without prices and without `UNS` .
ORDRSP – order response	`BGM+231` , function 29	`QTY+113`	Each line item is accepted with `LIN+…+5` .

If a column has no place in the selected message – for example a price in the despatch advice – the field is created and the draft notes "not provided for in DESADV". The order of the segments comes from the selected directory. In D.96A and D.01B the `UNH` carries the EANCOM identifier (purchase order `EAN008` / `EAN010` , invoice `EAN008` / `EAN011` , despatch advice `EAN005` / `EAN007`); the other releases and the order response carry none. In D.96A the GTIN is marked as `EN` and the article description is written according to the structure of that release (`IMD+F++:::…`).

The first column of every inhouse line is the **record type** (`K` , `P` , `E` in the example). The draft creates the template `~~SATZART` for it: every record starts with it, and it may be up to six characters long. A record type may appear several times in the description to show the nesting (pallet, line items, pallet, line items).

Some things are decided by **context**: `Name` , `Street` … and a VAT column belong in the header to the party whose GLN precedes them; a tax rate in the header applies to line items without their own; a discount rate after a line item concerns that line item. Details are in the part "The mapping".

Dates: order, delivery and best-before dates are written to your inhouse file as `DD.MM.YYYY` when a message is read; when a message is written, the draft reads them as `DDMMYYYY` – with or without dots – and converts them to the EDIFACT format with `FORMAT-DATE` . You set other formats in the definition yourself, for example `FORMAT-DATE (GET_VALUE (3), "CCYY-MM-DD", "102")` .

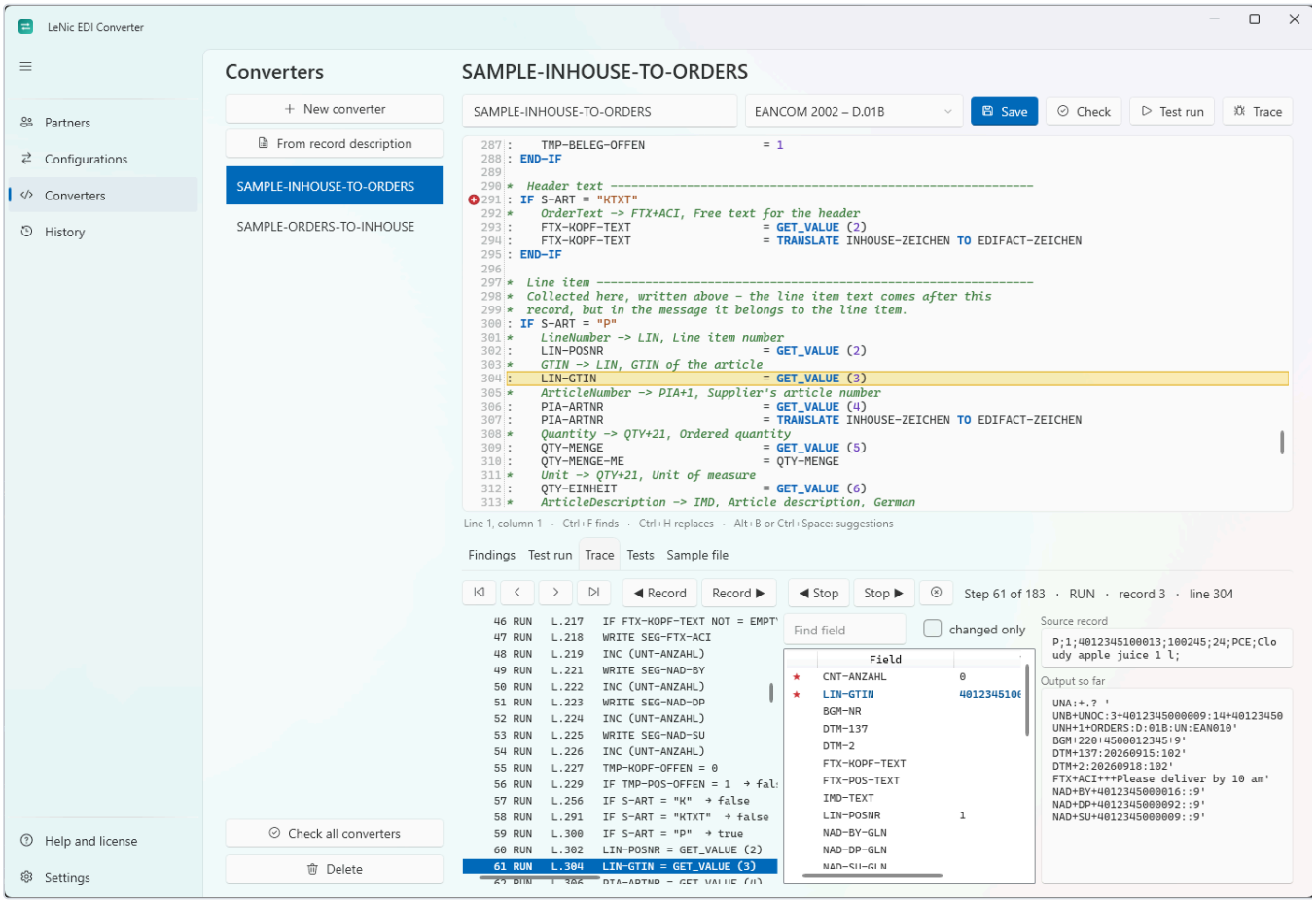
- **Check** reads the definition and shows all findings at once, with line number. Double-clicking a finding jumps to the line – or applies a suggestion if there is one.
- **Test run** converts a sample file and shows the result, optionally with a line break after every segment. **No file is written** and nothing is archived. For `~~PARTNER-GLN` and `~~PARTNER-NAME` you choose a partner; you enter the sender GLN next to it.

- **Trace** follows the test run step by step like a debugger, with breakpoints and watched fields – see "The trace" below.
- **Tests** records inputs together with the result the converter should deliver for them, and checks whether it still does after a change – see "Regression tests" below.
- **Sample file** shows the test file record by record. Place the cursor on the value you want to read: the editor suggests the pattern (for example `NAD+BY+VALUE`) and inserts it into the definition at the click of a button or with a double-click.
- **Explanation on hover:** move the mouse over a pattern or a segment tag and the editor explains the segment, highlights which position is read and gives the meaning of the qualifiers from the selected UN/EDIFACT directory.
- **Directory** (top right next to the name) – which directory release applies to explanations and patterns: automatically from the UNH of the test file or fixed, for example "EANCOM 1997 – D.96A". Releases D.96A to D.06B are included.

Keyboard shortcuts in the definition field

- `Ctrl+F` find, `F3` next, `Shift+F3` previous
- `Ctrl+Alt+↓` / `Ctrl+Alt+↑` – to the next or previous occurrence of the word at the cursor (field, variable, command) or of the selected text; a whole word is only found as a whole word. At the end or start it stops, and a hint says that nothing more follows
- `Ctrl+H` replace; "All" replaces in one step that can be undone with `Ctrl+Z`
- `Alt+B` or `Ctrl+Space` – suggestions for commands and names of the definition; `Esc` closes them
- `Ctrl+Z` / `Ctrl+Y` – undo and redo

The trace – debugging step by step



The trace of a test run: on the left the steps, in the middle the fields – watched ones with ★ on top –, on the right source record and output. In yellow the line of the step, the red dot with a plus is a conditional breakpoint.

The **Trace** button follows the test run step by step – like a debugger. It is recorded with the same input file, partner and sender GLN as the test run; afterwards the "Trace" tab shows:

- on the left every step with section (START, RUN, END), line, statement and result – for a condition "true" or "false"; a new source record is shown in bold,
- in the middle all fields with their value **after** this step; whatever the step has set is highlighted ("changed only" shows only those),
- on the right the source record and the output so far.

The line of the step is highlighted in yellow in the definition. The arrows or `F10` go one step forward, `Shift+F10` back; `F11` and `Shift+F11` jump to the next or previous source record. If the conversion fails, the trace stops right at the step that caused the error. Nothing is written and nothing is archived. At most 200,000 steps are recorded; for very large test files the editor says that it has truncated the trace.

Breakpoints: a click left of the line number sets a red dot, a second one removes it; `F9` does the same for the line with the cursor (on the "Trace" tab for the line of the current step). The dots move with their line while you edit; if the line is deleted, the dot disappears. `F5` or "Stop ▶" jumps to the next step on a line with a breakpoint, `Shift+F5` or "◀ Stop" to the previous one. If there are breakpoints, a new trace starts right at the first hit. The button with the cross removes all breakpoints. Because the trace is recorded completely, you may also set breakpoints afterwards and jump forwards as well as backwards.

Conditional breakpoints: a right-click on the margin opens a field for a condition – in the same form as after `IF`, also with `AND` and `OR`:

```
S-ART = "P"
LIN-GTIN = "4012345100013"
MENGE > 10 AND NAD-DP-GLN <> EMPTY
```

`Enter` accepts, `Esc` cancels, an empty field always stops again. The condition is checked **before** the line, with the state of the fields before it – as in Visual Studio. A dot with a condition carries a white plus, and the tooltip above the dot shows the condition. If a field name does not occur in the trace (for example a typo), the editor warns when jumping.

Watching fields: a right-click on a field in the field list offers

- **Watch** – the field always stays on top with ★, even if the filter or "changed only" would otherwise hide it;
- **Stop on change** – `F5` stops at every step that gives the field a different value (●); clearing after a `WRITE` counts too. The status line then shows "changes …". Setting the same value again does not stop.

The button with the cross also removes the breakpoints on fields.

Breakpoints are kept: breakpoints including conditions, watched fields and breakpoints on fields belong to the converter. They are saved with every change and are back the next time the converter is opened – even after a restart of the editor. A breakpoint remembers the text of its line: if the line has moved because of changes to the definition, it finds it again; if it no longer exists, the breakpoint is dropped.

Replaying a production run: on the "History" page, **Replay in trace** (button or right-click) replays a run in the debugger – with its input file, the archived one if available, otherwise the original one, and with the partner, sender GLN and character set of its configuration. If the conversion fails, the trace stops right at the step that caused the error. The replay uses the definition as it is **today**; if the converter has been changed since the run, the result may differ. If the run failed because of something other than the conversion – for example missing permission for the output folder – the hint says so. If the input file no longer exists, the button is greyed out.

Regression tests

A regression test records an input and the result the converter should deliver for it. After every change – to the converter or through a new version of the program – a comparison shows whether it still delivers the same. The tests are on the **Tests** tab of the converter page:

- **Save as test** creates a test: with the input file, the partner and the sender GLN from the "Test run" tab. Whatever the saved converter makes of it now counts as expected from then on – a conversion error or a user error can be an expected result, too. The input is stored in the test itself; the file may disappear afterwards.
- **Check** converts every input with the definition in the editor and compares character by character. This also works before saving: you see what a change does before it reaches production. ✓ means passed, ✗ differing; on the right is the first differing line – for EDIFACT the segment – with expected and current content, below it both results in full.
- **Accept as expected** makes the current result of a differing test the expected one – for an intended change. This only works with a saved converter.
- The bin deletes the selected test.

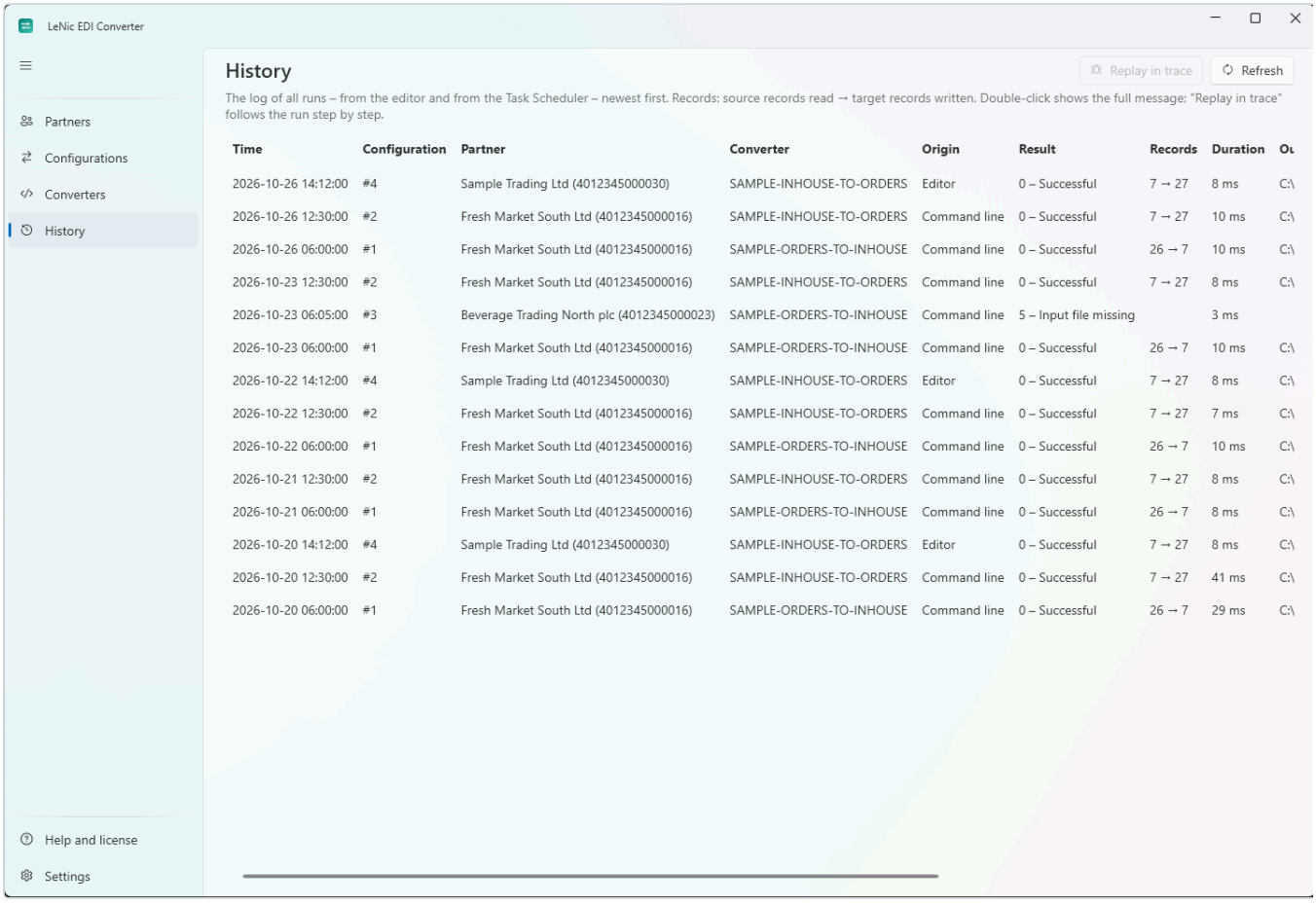
Date and time: when a test is created it records date and time ("reference date") and always converts with them. `DATE` and `TIME` – for example in the UNB – thus give the same result at every check instead of turning every test red.

Check all converters (on the left below the converter list) checks the tests of all converters with their **saved** definition, differences first – for example before an update at a customer site. A double-click opens the converter on the "Tests" tab.

Tests belong to the converter: if it is renamed, they move with it; if it is deleted, they go with it. They are stored in the database and backed up with it.

Messages in a test result follow the language that was set when the test was created. If you switch the language, a test that expects a conversion error may therefore differ – "Accept as expected" takes over the message in the new language.

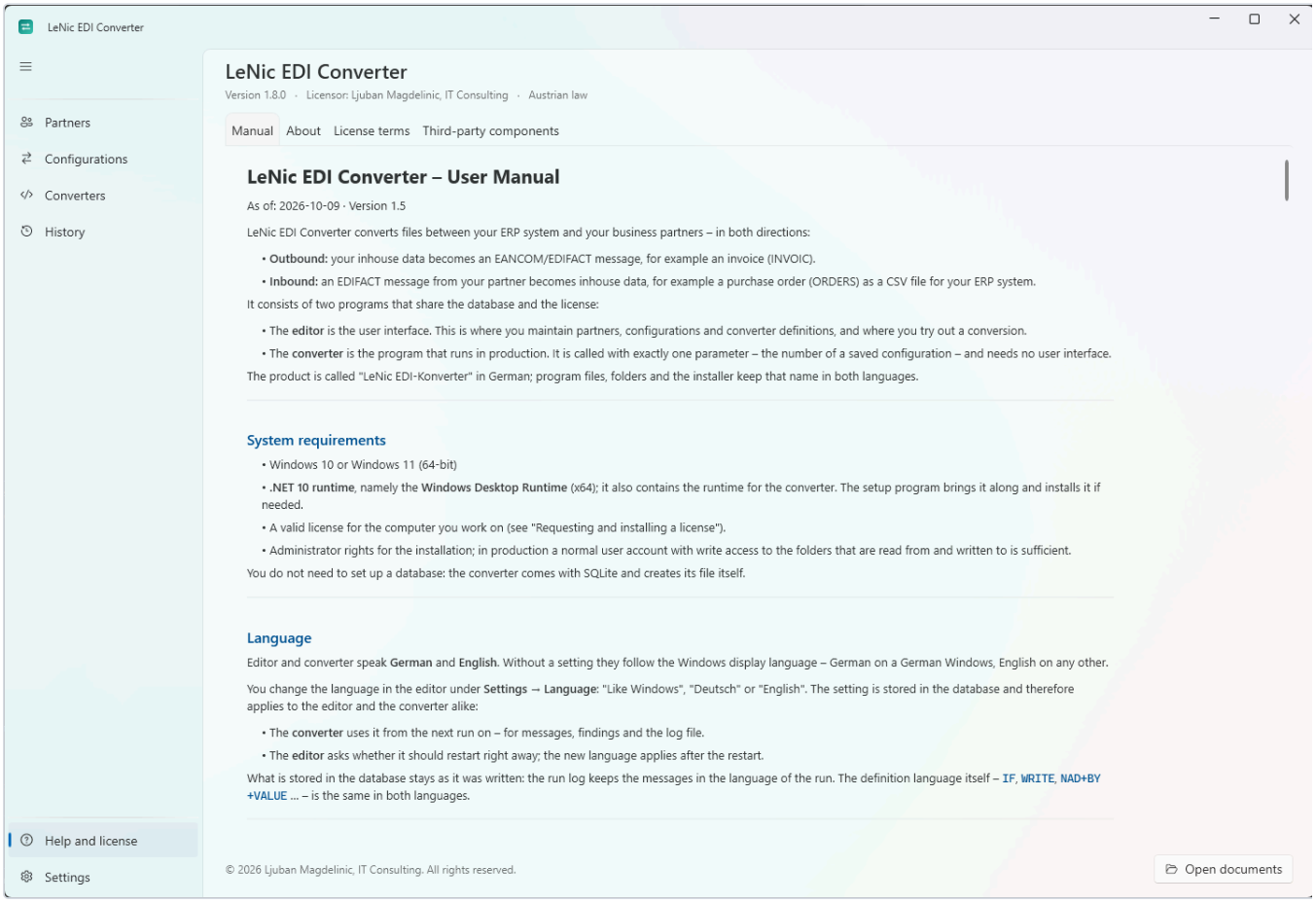
"History" page



The history: every run with time, configuration, partner, converter, origin and return value.

The log of all runs – from the editor and from the Task Scheduler – newest first, with time, configuration, partner, converter, origin, result, number of records read and written, duration, output and archive file. A double-click shows the full message. **Replay in trace** follows the selected run in the debugger – see "The trace".

"Settings" page



> This manual is also built into the editor: under "Help and license", tab "Manual".

The settings are on three tabs.

"General" tab

- **Appearance** – Like Windows, Light or Dark.
- **Language** – Like Windows, Deutsch or English; see "Language".
- **Global configuration** – "Delete run log after (days)". Default 90, `0` means never, at most 3650. Applies to all configurations; saving cleans up immediately.
- **About** – the version of the editor.

"Data & license" tab

- **Database** – where it is, with the button "Show in Explorer".
- **Samples** – "Add samples …" adds the supplied samples of this version as far as they are missing from your database.
- **License** – status, computer name and installation of a new license file.

"E-mail delivery" tab

- **E-mail delivery** – how notifications are sent, the same for all configurations, plus the **sender address**:

- *SMTP server* – server, port (default 587), encryption with STARTTLS, user and password (empty user = no login). Port 465 (encrypted from the start) is not supported – use 587.
- *Microsoft 365 (Graph)* – tenant ID, application ID and client secret of an app registration; see "E-mail via Microsoft 365". Microsoft 365 no longer accepts sign-in with a password over SMTP.

Password and client secret are stored encrypted and bound to this computer; on another computer – for example after moving the database – enter them again. An empty field keeps the saved value. "Send test e-mail" sends an e-mail with the values entered to the sender address, even before saving.

1.9 ZUGFeRD invoices

The converter creates a **ZUGFeRD invoice** (also Factur-X) from an inhouse file of your ERP system: a PDF that people can read and that contains the same invoice in machine-readable form as XML. The PDF complies with **PDF/A-3**, the XML follows the **EN 16931** profile according to UN/CEFACT CII.

To try it out there is configuration 7 with the converter **ZUGFERD-CSV** (see "Supplied samples"): press **Run** at the sample partner, and `Export\` contains `ZUGFERD_OUT_1.PDF`. If it is missing in an older installation, get it via **Settings → Data & license → Samples → "Add samples ..."**.

A ZUGFeRD converter contains everything it needs:

- the **output layout** in LAYOUT-CONF – the invoice itself, fixed for one profile, with its fields in TARGET-CONF (see "Output layouts"),
- **source and mapping** in SOURCE-CONF and RUN-CONF – they read your file and fill the fields. This is the only part you adapt.

For your own invoices:

1. **Definition:** A sample is supplied next to the converter, `Konverter\Samples\ZUGFERD-CSV.DEF`, with the sample invoice `ZUGFERD-EN16931.csv`; next to it `ZUGFERD-EN16931-RABATTE.csv` with a price discount, several discounts and a charge on one line item, a loyalty discount and freight, and `ZUGFERD-EN16931-KORREKTUR.csv` with a cancellation. Use it on the "Converters" page to create a new converter and adapt SOURCE-CONF and RUN-CONF to the structure of your file. The comments at the top describe which data the file needs.
2. **Configuration:** set **Output as** to "ZUGFeRD PDF" and enter an output file ending in `.pdf`, for example `{BASIS}\Export\INVOICE_{ZAEHLER}.pdf`.

What happens:

- The definition fills the layout, which yields the XML invoice. The visible page – header, address, line items, tax per rate, totals and payment – is created from **this XML**. PDF and XML therefore always show the same invoice.
- The XML is embedded as `factur-x.xml`, **always in UTF-8**, whatever output character set is configured.
- **One invoice per file.** If an input file contains two, the sample reports an error and writes nothing.
- The **totals** come from your ERP system. The converter only adds up the allowances and charges from the A records and checks that the tax basis matches line items, allowances and charges; otherwise it reports an error and writes nothing.
- If the output of the definition is not a CII invoice, the run ends with return value 7 (conversion error) and a message saying what is missing.

The sample knows:

- **Invoice, credit note and corrective invoice:** the invoice type in the K record is 380, 381 or 384. A correction or cancellation gives number and date of the corrected invoice in the **R record**; a cancellation carries negative quantities and amounts. The PDF is then titled "Credit note" or "Corrective invoice" and names the corrected invoice.

- **Discounts and charges on a line item:** a **price discount** per unit in the P record (gross price and discount, the net price is the price after it) and **any number of allowances and charges** as **N records** directly after the P record – amount and optionally percent, with a reason. The net amount of the line item is the amount after all N records.
- **Allowances and charges on the whole invoice** – loyalty discount, freight, packing – as **A records**, each with tax rate and reason.

Discounts and charges also appear on the visible invoice: below the line item or between "Total net" and the tax.

The sample covers tax category "S" (standard rate, also several rates) and payment by bank transfer. Tax-exempt invoices and the **XRechnung** for public-sector customers (routing ID, contact person, electronic addresses) need additions to layout and definition.

A generated invoice can be checked with freely available tools such as the Mustang validator; it checks PDF/A and the rules of EN 16931.

Output layouts

If more than one ERP system supplies invoices – or they arrive as an EANCOM message – the ZUGFeRD invoice does not have to be written anew for every source. An **output layout** is the invoice itself, as XML with placeholders such as `{KO-NUMMER}`. It is part of the definition itself, in LAYOUT-CONF, together with `FORMAT = LAYOUT` and its fields in TARGET-CONF; RUN-CONF only fills these fields. The order in which your file delivers header, line items and totals does not matter – the layout puts things in order. What a layout can do and how to write one is described in the language reference under "Output layout".

Two layouts with the same fields are supplied as **copy templates** in `Konverter\Samples` – they are not converters but are taken over into a definition:

Copy template	Profile	Delivery notes
`ZUGFERD-VORLAGE-EN16931.LAYOUT`	EN 16931 – also the basis of XRechnung	each delivery note once as a supporting document in the header
`ZUGFERD-VORLAGE-EXTENDED.LAYOUT`	EXTENDED	on **every line item**, with order, line within it and date – the collective invoice covering several delivery notes

EN 16931 has no delivery note on the line item; a collective invoice that assigns each line item to its delivery note needs EXTENDED. Not all recipients accept it, though – public-sector customers usually require XRechnung. Since both layouts have the same fields, you switch the profile by replacing LAYOUT-CONF with that of the other copy template. The visible invoice shows order and delivery note below the line item, or the delivery notes in the header.

Two definitions come along as samples, both also as sample configurations:

- `Konverter\Samples\ZUGFERD-CSV.DEF` (EN 16931) reads the in-house CSV – with three optional columns on the P record for delivery note, line and date. An N record belongs to the P record before it; the other records may come in any order, the layout puts them in order.
- `Konverter\Samples\ZUGFERD-INVOIC.DEF` (EXTENDED) reads an EANCOM invoice (D.01B) with order (`RFF+ON`) and delivery note (`RFF+DQ`) per line item.

1.10 Data from SQL Server (SQL-LINK)

If the input file lacks something that is in your ERP system – the description of an item, the delivery address for a customer number – the converter fetches it with `SQL-LINK` directly from a **Microsoft SQL Server**. The result is then available as a temporary table and is read with `WHILE NEXT`. The exact syntax is in the language reference (`docs/LanguageReference.md` , section 4).

1. Enter the connection. Every converter carries its own connection, in a section of its own, `::LINK-CONF` :

```
::LINK-CONF
: SQL-CONNECTION = "Server=sql01;Database=ERP;User Id=edi_reader;TrustServerCertificate=True"
: SQL-PASSWORD   = "your password"
```

You may type the password in plain text: on **saving** – and before a test run – the editor replaces it with an encrypted form (`"dpapi:…"`). It is never stored readable in the database. The encryption is bound to the computer; if you copy the converter to another computer, enter the password there again and save. Without `SQL-PASSWORD` the converter signs in with the Windows account it runs under – then `Integrated Security=True` belongs in the connection.

2. Declare the table. In SOURCE-CONF, one field per column of the result, in the same order:

```
: TEMP-TABLE.ITEMS
:    FIELD.I-EAN.X13
:    FIELD.I-TEXT.X35
```

3. Query. In RUN-CONF (or START/END-CONF); values from the definition are written in braces:

```
: SQL-LINK ITEMS = "SELECT EAN, Description FROM dbo.Items WHERE CustomerNo = {CUSTNO}"
: WHILE NEXT ITEMS
:    …
: END-WHILE
```

Good to know:

- **SQL-LINK only reads.** Exactly one `SELECT` is allowed – no `INSERT` , `UPDATE` , `DELETE` , no `EXEC` , no semicolon, no comments. The editor reports anything else as a finding when you save. In production every query also runs in a transaction that is always rolled back.
- **Nevertheless, set up a dedicated database user that may only read** (role `db_datareader` , or only `SELECT` on the tables and views needed). That is the most effective protection.
- A value in `{…}` is sent to the server as a parameter; characters such as `'` or `;` in your data cannot change the query.
- If a column is named like an SQL command, e.g. `Update` , write it in square brackets: `[Update]` .
- Every `SQL-LINK` in RUN-CONF queries once per source record. Fetch what is the same for all records once in START-CONF.
- If the server cannot be reached or the query fails, the run stops with return code **7**; the message in the log names the table and the server's text.

1.11 Calling the converter in production

The converter is called with **exactly one parameter**: the number of the configuration. Everything else is in the configuration.

```
C:\Tools\LeNicEdiKonverter\Konverter\LeNic.EdiKonverter.App.exe 1
```

The number is shown in the editor in front of each configuration (`#1`), the complete line at the top right of the "Configurations" page, with a button to copy it.

Automatic calls

The product **LeNicScheduler** is recommended: there you create a task with the program and the number as parameter and get scheduling, monitoring and notification as well.

The **Windows Task Scheduler** works just as well:

- Program/script: `C:\Tools\LeNicEdiKonverter\Konverter\LeNic.EdiKonverter.App.exe`

- Arguments: `1`
- Run with an account that can reach the input and output folders.

The program's return value is also its exit code – both tools recognize from it whether the run succeeded.

1.12 Return values

- **0 – successful.** The output file has been written.
- **1 – wrong call.** The number of the configuration is missing, or it is not a number.
- **2 – configuration not found.** Check the number; in the editor it is shown in front of each configuration.
- **3 – converter not found.** The configuration refers to a definition that no longer exists.
- **4 – error in definition.** Click "Check" in the editor; the findings name the line.
- **5 – input file missing.** Check the path; did the partner's file arrive? With a pattern the value means: there is currently nothing matching left in the folder – for a scheduled call this is the normal case and no reason to worry.
- **6 – output file exists.** Only with a fixed name without `{ZAEHLER}`. Remove the old file or add `{ZAEHLER}` to the pattern.
- **7 – conversion error.** For example a value that is longer than its field, or a character the output character set does not know. The message says where. Reproduce it with "Test run" in the editor.
- **8 – file error.** The file cannot be read or written. Check the permissions, or whether the file is currently held by another program.
- **9 – settings error.** The settings or the database cannot be opened. Is `data\` in the expected place? Does the database come from a newer program version?
- **10 – user error.** The definition reported `ERROR`; nothing was written. The messages are in the log.
- **11 – successful with warnings.** The output was written; the definition reported `WARNING` or the check against the EDIFACT directory found something.
- **12 – archive error.** The output has been written, but the input file could not be moved to the archive; it is still in its original location.
- **13 – license error.** No valid license for this computer, or it does not include a module the run needs (see "Modules"); nothing was converted.
- **14 – already running.** The same configuration or the same input file was busy for more than ten minutes.

1.13 Log, run log and history

One `.LOG` file per run in the log folder of the configuration (without a value `temp\` in the installation folder). It contains the details of the run, the return value, the message, all `WARNING` and `ERROR` messages and a statistic of how often each `IF` condition was true and false – helpful when a field stays empty and you want to know which branch ran. If the output file contains `{ZAEHLER}`, the log is named like the output and is overwritten for the same number; otherwise it carries a timestamp. An aborted run leaves a log, too. The log is written in the language that is set.

The run log in the database records every run, including failed ones. You read it in the editor under "History"; the last run is also shown for each configuration, as a coloured dot and with its time. How long it is kept is set under Settings → Global configuration.

1.14 Archive, restore and deletion protection

If archiving is switched on, the input file is moved to the archive **after a successful run** and gets date and time in its name. After a failure it stays where it is – so the run can be repeated once the cause has been fixed.

In the editor, the **Archive** button on the configuration opens the list:

- For each file you see the run that archived it and a preview.
- **Restore** copies the file back to the location of the input file so that it is processed again in the next run. A file already there is only overwritten after confirmation. **The archive itself stays unchanged** – it is the record of what was delivered. If the input file is a pattern, the file comes back under its original name: `INVOIC_7_20260920_101010.csv` becomes `INVOIC_7.csv` again.
- **Use as test file** opens the converter with this file.

Deletion protection: the program only deletes on its own within its installation folder. If your archive is somewhere else – and that is the normal case – old archive files are only deleted if you explicitly permit it in the configuration. Every permission, every revocation and every deletion outside is recorded in the database with time and Windows user; the last permission is shown in the form. Log files outside the installation folder are never deleted.

1.15 Concurrent runs

Every call is a process of its own; different configurations run side by side. If two runs of the **same** configuration or the same input file meet, the second one waits – via lock files in `temp\Sperren`, at most ten minutes. After that it ends with return value 14 without converting anything. This also applies between the Task Scheduler and the editor.

1.16 Backup

Back up regularly:

- `data\` – the database with partners, configurations, definitions and the run log. This is the most important folder.
- `Lizenz\` – your license file.
- Your own input, output and archive folders, unless they are backed up anyway.

You do not need to back up the program folders; they come from the delivery.

Going back to a backup: close the editor and the converter, rename the desired file `LeNic.EdiKonverter.vor-Schema…db` to `LeNic.EdiKonverter.db` (put the current one aside first) and start the editor. If the backup comes from an older version, it is brought up to date again when opened.

1.17 Different paths

If the database or the license should be located elsewhere – for example on a network drive – create the file `LeNic.EdiKonverter.json` in the installation folder:

```
{ "Datenbank": "data\\LeNic.EdiKonverter.db", "Lizenzordner": "Lizenz" }
```

`Datenbank` is the database, `Lizenzordner` the license folder. A relative path starts from the installation folder; a full path replaces it. In the paths of a configuration the installation folder is written as `{BASIS}`; this way they also survive a move of the installation. You can set the installation folder itself with the environment variable `LENIC_EDIKONVERTER_HOME`; it takes precedence over the folder convention.

1.18 E-mail via Microsoft 365

Microsoft 365 (Exchange Online) no longer accepts sign-in with user and password over SMTP. There the converter sends via **Microsoft Graph**: it signs in without a user, as an app, and sends on behalf of a mailbox. It connects over HTTPS (port 443) to `login.microsoftonline.com` and `graph.microsoft.com`. A Microsoft 365 administrator has to set this up.

1. Register an app – in the Microsoft Entra admin center under *Identity → Applications → App registrations → New registration*:

- A name such as "LeNic EDI Converter"; *Accounts in this organizational directory only*; no redirect URI.
- On the overview page, note the **Application (client) ID** and the **Directory (tenant) ID**.

2. Create a client secret – *Certificates & secrets → New client secret*. Copy the **value** at once, not the secret ID; it is not shown again later. Note the expiry date: after it, delivery fails until a new secret is entered.

3. Grant the permission – one of two ways:

- *Simple: API permissions → Add a permission → Microsoft Graph → Application permissions →* `Mail.Send`, then **Grant admin consent**. The app may then send on behalf of **any** mailbox.
- *Restricted to one mailbox (recommended):* do **not** grant `Mail.Send` in Entra, but in Exchange Online PowerShell for the sender mailbox only. The object ID is shown in Entra under *Enterprise applications* for the app:

  ```
  Connect-ExchangeOnline
  New-ServicePrincipal -AppId <application ID> -ObjectId <object ID> -DisplayName "LeNic EDI Converter"
  New-ManagementScope -Name "LeNic sender" -RecipientRestrictionFilter "PrimarySmtpAddress -eq 'edi-converter@company.com'"
  New-ManagementRoleAssignment -App <application ID> -Role "Application Mail.Send" -CustomResourceScope "LeNic sender"
  ```

  If `Mail.Send` is also granted in Entra, any mailbox applies again.

4. Enter it in the editor – *Settings → E-mail delivery → Microsoft 365 (Graph)*: tenant ID (or `company.onmicrosoft.com`), application ID, client secret and, as the **sender address**, the mailbox the e-mail is sent from – a user or shared mailbox in Exchange Online. Save, then "Send test e-mail". A new permission can take a few minutes to apply.

Common messages:

Message	Cause
`AADSTS7000215: Invalid client secret`	The secret ID was entered instead of the value, or a typo
`AADSTS7000222`	The client secret has expired – create and enter a new one
`AADSTS700016`	The application ID does not exist in this tenant
`AADSTS90002`	The tenant does not exist – check the tenant ID
`403 ErrorAccessDenied`	`Mail.Send` is missing, consent is missing, or the sender is outside the scope
`404 ErrorInvalidUser`, `MailboxNotEnabledForRESTAPI`	The sender address is not a mailbox in Exchange Online

---

1.19 Troubleshooting and frequently asked questions

The editor does not start and reports a missing component. The .NET 10 runtime ("Windows Desktop Runtime") is missing. Install it and restart the editor.

The editor shows the license window at start-up. The license is missing, has expired or was issued for another computer. The window names the reason and the computer name for the request.

The run ends with 5 although the file is there. Check the path in the configuration for typos and whether the Task Scheduler account can reach the drive – a mapped network drive (`X:\`) is often unknown to a service account; use the full network path (`\\Server\Share\…`).

The run ends with 6. The output file already exists. Add `{ZAEHLER}` or `{DATUM}` to the pattern; then a new name is created every time.

A field in the output stays empty. Look at the `IF` statistic in the log: if the condition was never true, the pattern does not match. In the editor, check the pattern against the file with "Sample file" – if it does not match, the field stays unchanged.

The run ends with 7: "Segment 1 is not an EDIFACT segment". The file is separated differently than the definition assumes – for example a wrong segment terminator in the `UNA`, or it is not an EDIFACT file at all. The message shows the start of the segment; control characters appear in it as `<CR>` and `<LF>`. Line breaks alone are not a problem: after every segment, before the segment terminator or in the middle of a segment (a file wrapped after 80 characters) they are skipped.

The run ends with 7: "FORMAT-DATE: … does not match the format …". A date in the input does not have the expected format, for example `7.8.2026` instead of `07.08.2026`, or it is not a valid date such as `31.02.2026`. Dots or no dots do not matter – the digits are counted. The message names the value; correct it in the ERP system or the format in the definition.

The despatch advice has one pallet too many or too few. Every record with an SSCC starts a new pallet; the line items after it belong to it. Check the order of the records in your file.

A tax line is missing in the invoice. Tax lines are collected per invoice and written after the totals – but only if the record type of the tax line comes before the summary record of this invoice. A tax line after the summary already belongs to the next one.

Accented characters arrive incorrectly. Set the input and output character sets explicitly in the configuration instead of leaving it to detection.

The run ends with 14. Another run of the same configuration is still busy or was terminated abruptly. If nothing is running any more, the files in `temp\Sperren` can be removed.

Can the same definition apply to several partners? Yes. The partner data comes from the configuration and is available in the definition as `~~PARTNER-GLN`, `~~PARTNER-NAME` and `~~ABSENDER-GLN`. Create one configuration per partner that uses the same converter.

Where does the number in the output file come from? The counter is maintained per configuration in the database and only assigned after a successful conversion. In the form it can be set explicitly with "Set".

The editor or the converter speaks the wrong language. Check **Settings** → **Language**. "Like Windows" uses German only on a German Windows display language. The editor applies a change after a restart; the converter at its next run.

1.20 Uninstallation

- Close the editor and the converter as well as scheduled tasks that call the converter.
- Remove "LeNic EDI-Konverter" under **Settings** → **Apps** → **Installed apps** (Windows 10: "Apps & features").
- **Your data stays:** `data\`, `Lizenz\`, `temp\` and the input and output folders including their contents. If they are no longer needed, delete the installation folder by hand afterwards – back up `data\` and `Lizenz\` first if you want to keep the data.
- Nothing remains in the registry; personal settings of the editor (only the colour scheme) are stored under `%APPDATA%\LeNic.EdiKonverter`.

1.21 Contact

Ljuban Magdelinic, IT Consulting — licensor and manufacturer Franz Lehar-Gasse 43, A-2384 Breitenfurt bei Wien, Austria **ljuban@magdelinic.consulting**

The license terms are in `LICENSE.md` (German, legally binding) and `LICENSE.en.md` (English translation), the third-party components used in `THIRD-PARTY-NOTICES.md` and `THIRD-PARTY-NOTICES.en.md`; all of them are next to the program and in the editor under "Help and license".

PART 2

The mapping

How column names become a converter

This chapter explains what happens between your file and your partner's message – and how the editor writes a converter for it without you having to type a single line of the definition language.

It is the middle part of the manual: the previous chapter shows the user interface, the following one describes the language in detail. What connects the two is described here.

2.1 What a mapping is

A mapping is the answer to a single question: **which value from the input belongs at which position of the output?**

The two sides are built differently. Your ERP system thinks in records and columns – a header record, line items below it, every column in its place. An EANCOM message thinks in segments and qualifiers – `BGM` carries the document number, `NAD+BY` the buyer, `QTY+21` the ordered quantity. Both are fixed, just differently.

The converter reads the input record by record and writes the output with `WRITE`. What happens in between is in the definition:

```
: IF S-SEG = "BGM"
:    K-ORDERNUMBER                    = BGM+*+VALUE
: END-IF
```

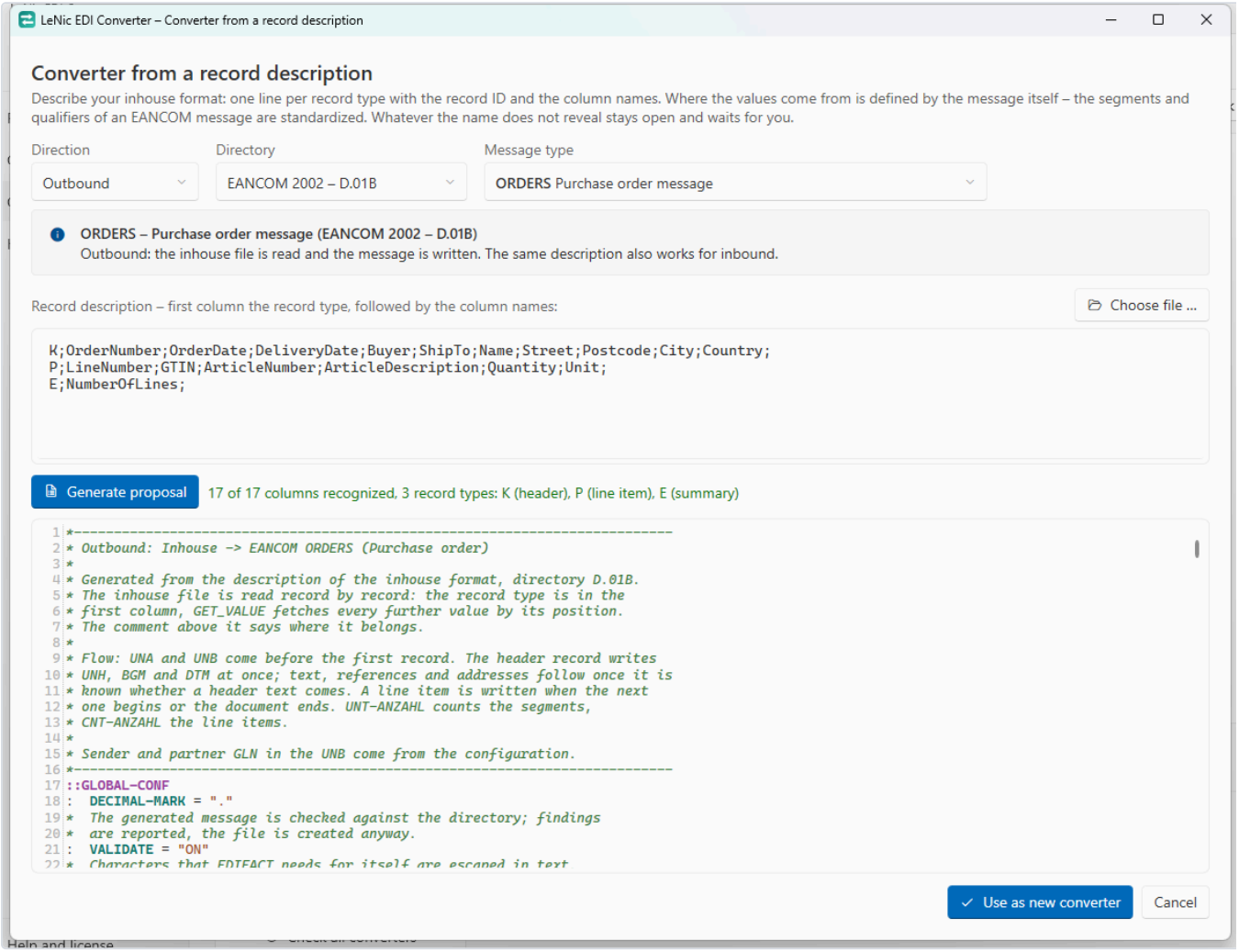
Read as: *if the record just read is a `BGM` segment, the order number is in its second group.* The same line in the opposite direction is:

```
:    BGM-NR                           = GET_VALUE (2)
```

The order number is in the second column of the inhouse record.

That is all a mapping is. It consists of many such lines, in the right order, with the cases in between that occur in everyday life: a date with a time, a text in several languages, a line item without a unit of measure.

2.2 The record description



> The draft is generated from the record description – here outbound, EANCOM 2002 – D.01B, ORDERS. The line next to the button says how many columns were recognized.

For the editor to write these lines, it needs a single piece of information: **what your file looks like.** That is the record description – one line per record type, first the record ID, followed by the names of the columns:

```
K;OrderNumber;OrderDate;DeliveryDate;Buyer;ShipTo;Supplier;
KTXT;OrderText;
P;LineNumber;GTIN;ArticleNumber;Quantity;Unit;ArticleDescription;
PTXT;LineText;
E;NumberOfLines;
```

This is **not a sample file with values** but a description. Nobody has to demonstrate what is in the columns: the meaning is in the name, and where this value goes in the message is standardized. German column names such as `Bestellnummer` or `Käufer` are recognized just as well.

The editor detects the separators itself (semicolon, tab, vertical bar or comma); an empty column after the last separator does not count. Lines starting with `*` are comments.

A record type may appear several times to show how the records interlock – in the despatch advice for example pallet, line items, pallet, line items. It is counted once. If it appears a second time with **different** columns, that is a contradiction, and the editor says so.

For purchase order, despatch advice and invoice the editor brings a template along. If you choose the message type in the window, its template is in the field – as long as you have not written anything of your own there. The templates are also supplied as files (`ORDER_VORLAGE.csv` , `DESADV.txt` , `INVOIC.txt`); their column names are German.

Which role a record has

The description first determines **what each record is**:

Role	How it is recognized
Line item	The record carries what makes up a line item – line number, GTIN, article number, quantity or article description. Price, amount and tax rate alone do not make a line item; they also appear in the header and in the totals
Header	A record without such columns that has none of the other roles
Header text, **line item text**	The record ID builds on that of another record: `KTXT` belongs to `K`, `PTXT` to `P`
Document discount, **line item discount**	The record carries a discount (`Discount %`, `Discount amount`). It belongs to the line item if its parent is one – `PR` to `P` – or, without a matching ID, to the record before it; otherwise to the whole document (`KR` to `K`)
Document surcharge, **line item surcharge**	Like a discount record, but with a surcharge (`Surcharge %`, `SurchargeAmount`): `KZ` to `K`, `PZ` to `P`
Tax line	Tax rate and taxable amount, but no invoice amount – one line per tax rate
Shipment	In the despatch advice: number of packages or total weight of the shipment
Package	In the despatch advice: the record carries an SSCC – for example a pallet. The line items after it belong to it until the next one starts
Summary	The record carries the number of line items or a total such as the invoice amount, but no line item – or is simply called `E` and comes last

In a text record the role takes precedence over the name: a column `ArticleDescription` in `PTXT` is the line item text, even though its name means an article description.

Which meaning a column has

Next every column gets its **meaning**. The comparison is lenient: the name is converted to lower case, German umlauts are resolved (`ä` → `ae`), and everything that is not a letter or a digit is dropped. "Buyer GLN" thus becomes `buyergln`. First equality is checked, then containment, longer keywords first – so that `LineText` does not get stuck on `Line`.

The editor knows these meanings, with their position in each message type. A dash means: not provided for there – the column is created but stays open, like an unrecognized one.

Meaning	Recognized by (selection)	ORDERS	INVOIC	DESADV	ORDRSP
Document number	`documentnumber`, `documentno`, `invoicenumber`	`BGM+220`	`BGM+380`	`BGM+351`	`BGM+231`
Order number	`ordernumber`, `orderno`, `purchaseordernumber`	`BGM+220`	`RFF+ON`	`RFF+ON`	`RFF+ON`
Document date	`documentdate`, `invoicedate`, `despatchdate`	`DTM+137`	`DTM+137`	`DTM+137`	`DTM+137`
Order date	`orderdate`, `purchaseorderdate`, `podate`	`DTM+137`	`DTM+171`	`DTM+171`	`DTM+171`
Delivery date	`deliverydate`, `requesteddeliverydate`, `arrivaldate`	`DTM+2`	`DTM+35`	`DTM+17`	`DTM+2`
Dispatch date	`shipdate`, `shippingdate`, `dispatchdate`	–	–	`DTM+11`	–
Buyer GLN	`buyer`, `buyergln`, `customer`	`NAD+BY`	`NAD+BY`	`NAD+BY`	`NAD+BY`
Delivery address GLN	`shipto`, `shiptogln`, `deliveryparty`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Supplier GLN	`supplier`, `suppliergln`, `seller`	`NAD+SU`	`NAD+SU`	`NAD+SU`	`NAD+SU`
Invoicee GLN	`invoicee`, `invoiceegln`, `billto`	`NAD+IV`	`NAD+IV`	`NAD+IV`	`NAD+IV`
Delivery address name	`companyname`, `company`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Delivery address street	`street`, `streetaddress`, `address1`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Delivery address city	`city`, `town`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Delivery address postcode	`zip`, `zipcode`, `postcode`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Delivery address country	`country`, `countrycode`	`NAD+DP`	`NAD+DP`	`NAD+DP`	`NAD+DP`
Currency	`waehrung`, `waehrungscode`, `currency`	`CUX+2`	`CUX+2`	–	`CUX+2`
Header text	`headertext`, `ordertext`, `documenttext`	`FTX+ACI`	`FTX+ACI`	`FTX+ACI`	`FTX+ACI`
Line number	`linenumber`, `lineno`, `lineitemnumber`	`LIN`	`LIN`	`LIN`	`LIN`
GTIN	`gtinnumber`, `eancode`, `barcode`	`LIN`	`LIN`	`LIN`	`LIN`
Article number	`articlenumber`, `articleno`, `itemnumber`	`PIA+1`	`PIA+1`	`PIA+1`	`PIA+1`
Quantity	`quantity`, `qty`, `orderedquantity`	`QTY+21`	`QTY+47`	`QTY+12`	`QTY+113`
Unit	`unit`, `uom`, `unitofmeasure`	`QTY+21`	`QTY+47`	`QTY+12`	`QTY+113`
Article description	`articledescription`, `itemdescription`, `description`	`IMD`	`IMD`	`IMD`	`IMD`
Line text	`linetext`, `itemtext`, `remark`	`FTX+LIN`	`FTX+LIN`	`FTX+LIN`	`FTX+LIN`
Price	`price`, `unitprice`, `netprice`	`PRI+AAA`	`PRI+AAA`	–	`PRI+AAA`
Line amount	`lineamount`, `linetotal`, `linevalue`	`MOA+203`	`MOA+203`	`MOA+203`	`MOA+203`
Tax rate	`taxrate`, `vatrate`, `vatpercent`	`TAX+7`	`TAX+7`	–	`TAX+7`
Number of lines	`numberoflines`, `linecount`, `totallines`	`CNT+2`	`CNT+2`	`CNT+2`	`CNT+2`
Goods value	`goodsvalue`, `netamount`, `nettotal`	`MOA+79`	`MOA+79`	–	`MOA+79`
Tax amount	`taxamount`, `vatamount`, `taxtotal`	–	`MOA+124`	–	–
Invoice amount	`invoiceamount`, `totalamount`, `grossamount`	–	`MOA+77`	–	–
Number of packages	`numberofpallets`, `palletcount`, `numberofpackages`	–	–	`PAC`	–
Shipment gross weight	`shipmentgrossweight`, `totalgrossweight`	–	–	`MEA+AAB`	–
Shipment net weight	`shipmentnetweight`, `totalnetweight`	–	–	`MEA+AAA`	–
Package number	`palletnumber`, `palletno`, `packagenumber`	–	–	–	–

SSCC	`serialshippingcontainercode`	–	–	`GIN+BJ`	–
Gross weight	`grossweight`, `gross`	–	–	`MEA+AAB`	–
Net weight	`netweight`	–	–	`MEA+AAA`	–
Lot number	`lotnumber`, `lotno`, `lot`	–	–	`GIN+BX`	–
Best-before date	`bestbeforedate`, `bestbefore`, `expirydate`	–	–	`DTM+361`	–
Delivery note number	`deliverynotenumber`, `deliverynoteno`, `packingslipnumber`	`RFF+DQ`	`RFF+DQ`	`BGM+351`	`RFF+DQ`
Supplier VAT ID	`vatnumber`, `vatno`, `taxid`	`RFF+VA`	`RFF+VA`	`RFF+VA`	`RFF+VA`
Taxable amount	`taxbase`, `taxableamount`, `taxbasis`	–	`MOA+125`	–	–
Discount percent	`discount`, `discountpercent`, `discountrate`	`PCD+1`	`PCD+1`	–	`PCD+1`
Discount amount	`discountamount`, `discountvalue`, `allowanceamount`	`MOA+204`	`MOA+204`	–	`MOA+204`
Surcharge percent	`surcharge`, `surchargepercent`, `surchargerate`	`PCD+2`	`PCD+2`	–	`PCD+2`
Surcharge amount	`surchargeamount`, `surchargevalue`, `chargeamount`	`MOA+23`	`MOA+23`	–	`MOA+23`
Terms of delivery	`deliveryterms`, `termsofdelivery`, `incoterm`	`TOD+6`	`TOD+6`	`TOD+6`	`TOD+6`

When the context decides

Some names do not say enough on their own. Then **where** the column is decides:

- **Address and VAT ID belong to the party before them.** In the header, `Name`, `Street`, `Postcode`, `City`, `Country` and a VAT ID belong to the party whose GLN comes last before them. After `Supplier` it is therefore the supplier's address (`NAD+SU`), after `ShipTo` the delivery address (`NAD+DP`). The VAT ID appears in the message as `RFF+VA` right after its `NAD`.
- **A tax rate in the header** applies to every line item that does not carry its own.
- **`Discount %` and `Discount amount`** are a discount on the whole document in the document discount, and one on the preceding line item in the line item discount. In a surcharge record, a column named `Wert` (value) is the surcharge amount.
- **In a tax line** rate, taxable amount and tax amount are those of this tax rate; in the summary the same names are the totals of the document.
- **`Net value`** is the line item's amount in the line item (`MOA+203`) and the total of the line items in the summary (`MOA+79`).
- **The delivery note number** is the document itself in the despatch advice (`BGM`) and a reference in the invoice (`RFF+DQ`). If it is in the line item record, it belongs to the line item: in the invoice every line item then carries its delivery note (`RFF+DQ` after the price), and a delivery date in the line item record is the date of that delivery note (`DTM+171`). This makes a collective invoice.

What is not a known name is not guessed. The column is created anyway, its assignment stays open, and the message below the button names it. In the definition there is a note in its place:

```
*    Column 8 "ClerkName": not recognized – please add the assignment by hand
```

That is intentional. A guessed value in the wrong place is only noticed by the partner; an open line is noticed at once.

2.3 Both directions from one description

The same description serves both directions. What "DeliveryDate" means and where `PTXT` belongs does not change depending on whether the file is read or written – only what is created from it.

You choose the direction at the top of the window in the **Direction** field (Inbound or Outbound), next to it the **Directory** and the **Message type**: purchase order (ORDERS), invoice (INVOIC), despatch advice (DESADV) or order response (ORDRSP). The types differ less than one might think – header with `BGM`, `DTM` and addresses, line items with `LIN`, `PIA`, `IMD` and `QTY`, at the end `CNT`. What one carries beyond that, such as prices and totals in the invoice or the shipping level `CPS` in the despatch advice, the editor knows. It takes the **order of the segments** from the selected directory, because it differs between the types and the releases.

The examples in this chapter show the purchase order.

Inbound: message → inhouse

The message is read, your file is written. The order is given by the message, and it is the reason for the only subtlety of this direction:

- The header segments – `BGM`, `DTM`, `NAD`, `FTX` – fill the fields of the header record.
- The header record is only written at the **first** `LIN`, however. Before that it is not certain that all header segments have been read.
- A line item is collected and written at the **next** `LIN` **or at** `UNS` – `QTY` and `IMD` only follow after `LIN`.
- At `UNT` come the last line item, the trailer record and the comparison with `CNT`: if the message reports a different number of line items than were read, there is a warning in the log.

The processing state is kept in a record `TEMP` that is never written: the qualifier just read, the language, the counters and two flags.

Two details of this direction:

- **The article description** is read from the `IMD` in German – or from an `IMD` without any language (`IMD+F++:::Text`), as many partners send it.
- **Order, delivery and best-before dates** are written to your file by the draft as `DD.MM.YYYY`, i.e. `04.08.2026` instead of `20260804`.

Outbound: inhouse → message

Your file is read, the message is written. The record type is in the first column, `GET_VALUE` fetches every further value by its position. In the opposite direction the draft writes the record type at the start of each record via the template `~~SATZART`.

- `UNA` and `UNB` come before the first record and are created in START-CONF. **Sender and partner GLN come from the configuration**, not from the definition – this way one converter serves several partners.
- The header record writes `UNH`, `BGM` and the `DTM` segments at once. `FTX` and the `NAD` group only follow once it is known whether a header text comes: in the message the free text comes **before** the `NAD` group, in your file after the header record.
- A line item is collected and written when the next one begins or the document ends – for the same reason: the line item text comes after the line item record, in the message after the quantity.
- The trailer record closes with `UNS` (where the message has one), `CNT`, the totals and `UNT`; `UNZ` is written in END-CONF.
- **Tax lines** come before the total in your file but after it in the message. They are therefore collected in a temporary table – each with the number of its message, so that with two invoices in one file each only gets its own – and written in the summary.
- **Date:** the draft reads order, delivery and best-before dates as `DDMMYYYY` and writes them in format 102 (`CCYYMMDD`) with `FORMAT-DATE`. A date with dots (`06.10.2026`) is read as well. If your file has no document date, `DTM+137` is written with the day of conversion – EANCOM requires it.
- **The segments follow each other without line breaks**, as partners usually expect them.

Counting happens while writing. `UNT-ANZAHL` starts at 1 – the `UNT` counts itself – and is incremented after every segment written. After the `WRITE` it is back at 1, because a record returns to its initial state after it has been written. Likewise `CNT-ANZAHL` counts the line items and `UNH-NR` the messages.

This direction takes care of three things that are easily forgotten by hand:

- **Free texts are escaped.** An apostrophe in an article description would otherwise end the segment; `TRANSLATE INHOUSE-ZEICHEN TO EDIFACT-ZEICHEN` turns it into `?'`.
- **The generated message is checked.** `VALIDATE = "ON"` holds it against the UN/EDIFACT directory – segment structure, formats, code lists, plus the envelope and the counters in `UNT` and `UNZ` – and reports what does not fit before the file reaches the partner.
- **A file without a trailer record is an error.** A message without `UNT` should not go out; the run reports it and writes no output file.

A number field that is only written when needed – a discount, a weight, a tax rate – starts **empty** instead of with 0. This way a segment is only created if your file really has a value there, and not a discount of 0 in every line item.

2.4 A run to follow along

Both converters are created from the description above. The partner's purchase order …

```
UNA:+.? '
UNB+UNOC:3+4333685000000:14+5990038744005:14+260804:1050+37626138+++++EANCOM'
UNH+1+ORDERS:D:01B:UN:EAN010'
BGM+220::9+780307082681+9'
DTM+137:20260804:102'
DTM+2:20260807:102'
NAD+BY+4335815000003::9'
NAD+DP+4335815000089::9'
NAD+SU+5990038744005::9'
RFF+YC1:0007803'
LIN+1++5996227007944:SRV'
PIA+1+50091040010:IN::92'
… (29 more lines)
```

… becomes this inhouse file inbound:

```
K;780307082681;04.08.2026;07.08.2026;4335815000003;4335815000089;5990038744005;
P;1;5996227007944;50091040010;24;;Wiesbauer Grillmix;
P;2;5996227003656;50091090010;108;;Wiesbauer Tura SK1;
P;3;4052917470663;50091460010;60;;Wiener Wuerstchen;
P;4;4052917470687;50091500014;12;;Stuttgarter/ Tiroler SK1;
P;5;4052917470724;50091540014;12;;Griller SK1;
E;5;
```

And the same file becomes a purchase order again outbound:

```
UNA:+.?
'UNB+UNOC:3+4333685000000:14+4335815000003:14+261102:2221+2611022221'UNH+1+ORDERS:D:01B:UN:EAN010'BGM+220+78
0307082681+9'DTM+137:20260804:102'DTM+2:20260807:102'NAD+BY+4335815000003::9'NAD+DP+4335815000089::9'NAD+SU+
5990038744005::9'LIN+1++5996227007944:SRV'PIA+1+50091040010:SA::91'IMD+F++:::Wiesbauer
Grillmix'QTY+21:24'LIN+2++5996227003656:SRV'PIA+1+50091090010:SA::91'IMD+F++:::Wiesbauer Tura
SK1'QTY+21:108'LIN+3++4052917470663:SRV'PIA+1+50091460010:SA::91'IMD+F++:::Wiener
Wuerstchen'QTY+21:60'LIN+4++4052917470687:SRV'PIA+1+50091500014:SA::91'IMD+F++:::Stuttgarter/ Tiroler
SK1'QTY+21:12'LIN+5++4052917470724:SRV'PIA+1+50091540014:SA::91'IMD+F++:::Griller
SK1'QTY+21:12'UNS+S'CNT+2:5'UNT+30+1'UNZ+1+2611022221'
```

What stands out is intended: the second purchase order is not byte for byte the first. The envelope carries date, time and reference of **this** run, the GLNs come from the configuration, and segments that the inhouse format does not carry – in the example `RFF`, `MEA` and the Hungarian article description – are not back. A converter only passes on what is in the description.

2.5 Despatch advice: shipment, pallets, line items

The despatch advice (DESADV) tells the customer what comes on which pallet. The supplied template describes it like this: header `K`, shipment `S`, one package `SSCC` per pallet and below it its line items `P`:

```
K;OrderNumber;DeliveryNoteNumber;OrderDate;DeliveryDate;Buyer;ShipTo;Name;Street;Postcode;City;Country;
S;NumberOfPallets;ShipmentGrossWeight;ShipmentNetWeight;
SSCC;PalletNumber;SSCC;GrossWeight;NetWeight;
P;LineNumber;GTIN;ArticleNumber;ArticleDescription;DeliveredQuantity;Unit;LotNumber;BestBeforeDate;
P;LineNumber;GTIN;ArticleNumber;ArticleDescription;DeliveredQuantity;Unit;LotNumber;BestBeforeDate;
SSCC;PalletNumber;SSCC;GrossWeight;NetWeight;
P;LineNumber;GTIN;ArticleNumber;ArticleDescription;DeliveredQuantity;Unit;LotNumber;BestBeforeDate;
```

A delivery from the supplied test data (`INHOUSE\DESADV_OUT.CSV`) – two pallets with five line items, some with batch and best-before date:

```
K;PO-2026-100234;LS-2026-500123;06.10.2026;14.10.2026;4012345000009;4012345111118;Muster Handels GmbH Lager
Nord;Lagerstrasse 7;4020;Linz;AT;
S;2;710.5;662.5;
SSCC;1;340123450000000017;412.5;387.5;
P;1;4001234567890;10001;Schrauben M8x40 verzinkt;500;PCE;;;
P;2;4001234567891;10002;Kabel NYM-J 3x1,5 mm2;250;MTR;L2026-0815;;
P;3;4001234567892;10003;Wandfarbe weiss matt 10 Liter;24;PCE;L2026-0921;30.09.2028;
SSCC;2;340123450000000024;298.0;275.0;
P;4;4001234567893;10004;Industriereiniger Konzentrat;60;LTR;L2026-1002;31.03.2029;
P;5;4001234567894;10005;Handschuhe Nitril Gr. L (Karton a 100 Stk);12;CT;;;
```

This becomes the following message (shown without envelope, one segment per line):

```
UNH+1+DESADV:D:01B:UN:EAN007'
BGM+351+LS-2026-500123+9'
DTM+137:20261102:102'
DTM+17:20261014:102'
RFF+ON:PO-2026-100234'
DTM+171:20261006:102'
NAD+BY+4012345000009::9'
NAD+DP+4012345111118::9++Muster Handels GmbH Lager Nord+Lagerstrasse 7+Linz++4020+AT'
CPS+1'
PAC+2++201'
MEA+PD+AAB+KGM:710.5'
MEA+PD+AAA+KGM:662.5'
CPS+2+1'
PAC+1++201'
MEA+PD+AAB+KGM:412.5'
MEA+PD+AAA+KGM:387.5'
PCI+33E'
GIN+BJ+340123450000000017'
LIN+1++4001234567890:SRV'
PIA+1+10001:SA::91'
IMD+F++:::Schrauben M8x40 verzinkt'
QTY+12:500:PCE'
LIN+2++4001234567891:SRV'
PIA+1+10002:SA::91'
IMD+F++:::Kabel NYM-J 3x1,5 mm2'
QTY+12:250:MTR'
PCI+36E'
GIN+BX+L2026-0815'
LIN+3++4001234567892:SRV'
PIA+1+10003:SA::91'
IMD+F++:::Wandfarbe weiss matt 10 Liter'
QTY+12:24:PCE'
PCI+36E'
DTM+361:20280930:102'
GIN+BX+L2026-0921'
CPS+3+1'
PAC+1++201'
MEA+PD+AAB+KGM:298'
MEA+PD+AAA+KGM:275'
PCI+33E'
GIN+BJ+340123450000000024'
LIN+4++4001234567893:SRV'
PIA+1+10004:SA::91'
IMD+F++:::Industriereiniger Konzentrat'
QTY+12:60:LTR'
PCI+36E'
DTM+361:20290331:102'
GIN+BX+L2026-1002'
LIN+5++4001234567894:SRV'
PIA+1+10005:SA::91'
IMD+F++:::Handschuhe Nitril Gr. L (Karton a 100 Stk)'
QTY+12:12:CT'
CNT+2:5'
UNT+54+1'
```

What the converter does by itself:

- **It counts the shipping levels:** `CPS+1` is the shipment, every pallet gets the next level below it (`CPS+2+1` , `CPS+3+1` …). The column "LfdNr der Palette" (running pallet number) is not needed for that.
- **Pallets are EUR pallets** (`PAC+…++201`), weights in kilograms (`MEA+PD+AAB` gross, `MEA+PD+AAA` net).
- **The SSCC** appears as `GIN+BJ` after `PCI+33E` , **batch and best-before date** as `GIN+BX` and `DTM+361` after `PCI+36E` – only for line items that carry one of them. `33E` and `36E` are EANCOM codes; the check against the UN directory does not report them as a deviation.

2.6 Invoice: parties, discounts, taxes

The invoice (INVOIC) carries more than the other messages: three parties with VAT ID, discounts on document and line items and one tax line per tax rate. The supplied template:

```
K;InvoiceNumber;OrderNumber;DeliveryNoteNumber;OrderDate;DeliveryDate;Buyer;VATID;ShipTo;VATID;Name;Street;P
ostcode;City;Country;Supplier;VATID;Name;Street;Postcode;City;Country;Currency;TaxRate;
KR;Discount %;DiscountAmount;
P;LineNumber;GTIN;ArticleNumber;ArticleDescription;Quantity;Unit;Price;NetValue;TaxRate;DeliveryNoteNumber;D
eliveryDate;
PR;Discount %;DiscountAmount;
P;LineNumber;GTIN;ArticleNumber;ArticleDescription;Quantity;Unit;Price;NetValue;TaxRate;DeliveryNoteNumber;D
eliveryDate;
PR;Discount %;DiscountAmount;
TAX;TaxBase;TaxAmount;TaxRate;
TAX;TaxBase;TaxAmount;TaxRate;
SUM;GrossAmount;NetValue;TaxBase;TaxAmount;
```

An invoice from the test data (`INHOUSE\INVOIC_OUT.CSV`), a collective invoice covering two delivery notes: line items 1 to 3 came with `LS-2026-500123` , line items 4 and 5 with `LS-2026-500125` ; the header therefore carries no delivery note. In addition 2 % discount on the document, 5 % on line item 3, which carries no tax rate of its own and therefore gets the 20 % from the header, and line item 5 with 10 %:

```
K;RE-2026-900001;PO-2026-100234;;06.10.2026;;4012345000009;ATU12345678;4012345111118;ATU12345678;Muster
Handels GmbH Lager Nord;Lagerstrasse 7;4020;Linz;AT;4012345999990;ATU87654321;LeNic Industriebedarf
GmbH;Industriestrasse 12;1100;Wien;AT;EUR;20;
KR;2;40.42;
P;1;4001234567890;10001;Schrauben M8x40 verzinkt;500;PCE;0.12;60.00;20;LS-2026-500123;14.10.2026;
P;2;4001234567891;10002;Kabel NYM-J 3x1,5 mm2;250;MTR;1.45;362.50;20;LS-2026-500123;14.10.2026;
P;3;4001234567892;10003;Wandfarbe weiss matt 10 Liter;24;PCE;39.90;909.72;;LS-2026-500123;14.10.2026;
PR;5;47.88;
P;4;4001234567893;10004;Industriereiniger Konzentrat;60;LTR;8.50;510.00;20;LS-2026-500125;16.10.2026;
P;5;4001234567894;10005;Handschuhe Nitril Gr. L (Karton a 100 Stk);12;CT;14.90;178.80;10;LS-2026-
500125;16.10.2026;
TAX;1805.38;361.08;20;
TAX;175.22;17.52;10;
SUM;2359.20;2021.02;1980.60;378.60;
```

The corresponding message:

```
UNH+1+INVOIC:D:01B:UN:EAN011'
BGM+380+RE-2026-900001+9'
DTM+137:20261102:102'
RFF+ON:PO-2026-100234'
DTM+171:20261006:102'
NAD+BY+4012345000009::9'
RFF+VA:ATU12345678'
NAD+DP+4012345111118::9++Muster Handels GmbH Lager Nord+Lagerstrasse 7+Linz++4020+AT'
RFF+VA:ATU12345678'
NAD+SU+4012345999990::9++LeNic Industriebedarf GmbH+Industriestrasse 12+Wien++1100+AT'
RFF+VA:ATU87654321'
CUX+2:EUR:4'
ALC+A+++1+DI'
PCD+1:2'
MOA+204:40.42'
LIN+1++4001234567890:SRV'
PIA+1+10001:SA::91'
IMD+F++:::Schrauben M8x40 verzinkt'
QTY+47:500:PCE'
MOA+203:60'
PRI+AAA:0.12'
RFF+DQ:LS-2026-500123'
DTM+171:20261014:102'
TAX+7+VAT+++:::20'
LIN+2++4001234567891:SRV'
PIA+1+10002:SA::91'
IMD+F++:::Kabel NYM-J 3x1,5 mm2'
QTY+47:250:MTR'
MOA+203:362.5'
PRI+AAA:1.45'
RFF+DQ:LS-2026-500123'
DTM+171:20261014:102'
TAX+7+VAT+++:::20'
LIN+3++4001234567892:SRV'
PIA+1+10003:SA::91'
IMD+F++:::Wandfarbe weiss matt 10 Liter'
QTY+47:24:PCE'
MOA+203:909.72'
PRI+AAA:39.9'
RFF+DQ:LS-2026-500123'
DTM+171:20261014:102'
TAX+7+VAT+++:::20'
ALC+A+++1+DI'
PCD+1:5'
MOA+204:47.88'
LIN+4++4001234567893:SRV'
PIA+1+10004:SA::91'
IMD+F++:::Industriereiniger Konzentrat'
QTY+47:60:LTR'
MOA+203:510'
PRI+AAA:8.5'
RFF+DQ:LS-2026-500125'
DTM+171:20261016:102'
TAX+7+VAT+++:::20'
LIN+5++4001234567894:SRV'
PIA+1+10005:SA::91'
IMD+F++:::Handschuhe Nitril Gr. L (Karton a 100 Stk)'
QTY+47:12:CT'
MOA+203:178.8'
PRI+AAA:14.9'
RFF+DQ:LS-2026-500125'
DTM+171:20261016:102'
TAX+7+VAT+++:::10'
UNS+S'
CNT+2:5'
MOA+77:2359.2'
MOA+79:2021.02'
MOA+124:378.6'
MOA+125:1980.6'
TAX+7+VAT+++:::20'
```

```
MOA+125:1805.38'
MOA+124:361.08'
TAX+7+VAT+++:::10'
MOA+125:175.22'
MOA+124:17.52'
UNT+76+1'
```

Please note:

- **The converter does not recalculate the amounts.** Line amounts, taxes and totals come from your ERP system and go out as they are there.
- **Delivery note per line item.** `DeliveryNoteNumber` and `DeliveryDate` at the end of the line item record are optional. A file without these columns – like the second invoice in the test data – remains valid; a line item with an empty delivery note number gets no `RFF+DQ` and no date. A delivery date in the line item record without a delivery note number stays open, because its `DTM+171` belongs to the group of the delivery note.
- **One discount per line item.** If two discount records follow a line item, the last one applies. The same goes for surcharges.
- **Surcharges** appear in the message like discounts, but as `ALC+C` with `PCD+2` (percent) and `MOA+23` (amount). The kind of surcharge is `ABK` (miscellaneous); if you know what it is, enter the more precise code – for example `FC` for freight or `PC` for packing.
- **Terms of delivery** – a column `DeliveryTerms` or `Incoterm` – go into every message as text: `TOD+6++:::free domicile`. If your partner requires the Incoterm as a code (`TOD+6++EXW`), change the target record `SEG-TOD` accordingly.
- **If your file has no invoice date**, the `DTM+137` carries the day of conversion. If it should be the date from the ERP system, add a column "InvoiceDate" to the header.

2.7 Continuing the draft by hand

The draft arrives in the editor **unsaved**. What remains to be done:

1. **Fill the open lines.** Every unrecognized column is there as a comment. For the outbound direction the assignment is one line: `MY-FIELD = GET_VALUE (8)`. For the inbound direction the **Sample file** tab helps: place the cursor on the value and the editor suggests the pattern that reads it – for example `RFF+YC1:VALUE`.
2. **Check the qualifiers.** The draft assumes the usual ones: `220` for the purchase order (`380` invoice, `351` despatch advice, `231` order response), `9` for the original, `SRV` for the GTIN (`EN` in D.96A), `SA::91` for the supplier's article number, `IMD+F++:::` for the article description without language, `DI` for a discount, `ABK` for a surcharge. Your partner may require others; their message implementation guide decides. If you move the mouse over a qualifier in the editor, its meaning from the directory is shown.
3. **Give the converter a name and save it.** Only then is it in the list and can it be selected in a configuration.
4. **Do a test run** before a configuration points to it. The test run writes no file and archives nothing.

A word on expectations: the draft is **a start, not a result**. It takes the framework off your hands – segment order, counters, release characters, record changes – i.e. exactly the work where one miscounts. The last few percent belong to your partner and your company.

PART 3

The definition language

All sections, commands and formats

This reference describes what the converter **actually understands today**. It is the basis for the editor and for every change to the language: what is not described here does not exist. Whenever the language changes, this file is updated at the same time. The German version is `docs/Sprachbeschreibung.md`.

The language itself is the same in German and English: commands, sections and placeholders are written identically. Only the findings and messages follow the language set in the editor (Settings → Language).

3.1 Structure of a definition

A definition is a text made of lines. What counts is the first character of a line; leading spaces are ignored:

Line starts with	Meaning
`::` … `-CONF`	start of a section, for example `::RUN-CONF`
`:`	statement; the rest of the line without the colon, trimmed on both sides
`*`	comment, also `*:` for a commented-out statement
anything else	is skipped (separator lines, headings, empty lines)

Statements before the first section belong to GLOBAL-CONF.

Sections

Section	Required	Contents
`::GLOBAL-CONF`	yes	settings, constants, templates, character lists
`::TARGET-CONF`	yes	structure of the target: records and fields
`::SOURCE-CONF`	yes	structure of the source: separators, record type
`::START-CONF`	no	runs once before the first source record
`::RUN-CONF`	yes	runs once per source record
`::END-CONF`	no	runs once after the last source record
`::LAYOUT-CONF`	with `FORMAT = LAYOUT`	the output layout as XML; its lines count as they stand – see 5
`::LINK-CONF`	with `SQL-LINK`	the connection to Microsoft SQL Server (`SQL-CONNECTION`, `SQL-PASSWORD`) – see 4

If a required section is missing, the editor reports a finding.

Findings

While reading, **all** errors are collected, not just the first. Every finding names its line in the source text, for example `Line 12: TARGET: Unknown type specification '35' …`. A finding that concerns the whole definition, such as a missing section, has no line. A text without a closing quotation mark, such as `WARNING ("Attention)`, is a finding in every section. A definition with findings is not executed.

3.2 Values

Notation	Meaning	Example
`"Text"`	text; spaces inside are kept	`"NAD+BY+"`
`X"hh…"`	characters by their code, two hex digits per character	`X"0D0A"` (CR LF), `X"27"` (')
`~~NAME`	constant from `CONST` in GLOBAL-CONF	`~~INVOICE`
integer	number without quotation marks	`380` , `0`
`EMPTY`	empty value; see assignments and conditions	

Predefined constants

Three constants do not appear in any definition. Their value comes from the configuration the converter runs with:

Constant	Value	Example
`~~PARTNER-GLN`	GLN of the configuration's partner	`4012345000009`
`~~PARTNER-NAME`	company name of the partner	`Smith Ltd`
`~~ABSENDER-GLN`	sender GLN, a field of its own in the configuration; may stay empty if the definition does not need it	`4000001000005`

This way no GLN in the UNB has to be fixed in the definition, and one converter serves several partners and senders. The constants apply wherever a constant may be used: in assignments, in `CONCAT` , in conditions, as the value of a template in TARGET-CONF (`~~FILL-10 = ~~PARTNER-GLN`) and as the initial value of a field (`FIELD.NAD-ILN.N13 = ~~PARTNER-GLN`).

```
: RECORD.SEG-UNB
:    ~~FILL-10                        = "UNB+UNOC:3+"
:    ~~FILL-10                        = ~~ABSENDER-GLN
:    ~~FILL-10                        = ":14+"
:    ~~FILL-10                        = ~~PARTNER-GLN
:    ~~FILL-10                        = ":14"
```

- A `CONST` or `TEMPLATE` with one of these names is a finding.
- In the editor's **test run**, partner and sender GLN are entered next to the input file. If the value is missing, a definition that needs the constant aborts with a message. The same applies in production with an empty sender GLN (return value 7). The name is never output as text.
- A `CONST` whose value is a predefined constant (`CONST ~~RECIPIENT = ~~PARTNER-GLN`) is also a finding, because the value only arrives with the run. Use the constant directly.

A character code from `X"nn"` stands for exactly this byte value. The **output** is written byte by byte in Latin-1, so that an `X"84"` arrives in the file as byte `0x84` .

The **input file** may also arrive in UTF-8. Which character set applies is set in the configuration; the default "Automatic" detects it from the content: a byte order mark decides; otherwise UTF-8 applies if the whole file can be read as UTF-8 and at least one character above 127 occurs, else Latin-1.

This changes nothing for the definition: an "ä" becomes the same character with the number `E4` in both cases, and `CHAR-LIST` matches it as before. Only a character that does not exist in Latin-1 at all – for example the euro sign from a UTF-8 source – cannot be written to the output; that is a conversion error (return value 7) and not a silent question mark. Where such characters occur, translate them with `CHAR-LIST` .

3.3 GLOBAL-CONF

Entry	Meaning
`TARGET-DECIMAL-MARK = "."`	decimal mark of the output for the formats `Z` and `-`
`DECIMAL-MARK = ","`	decimal mark for source and output where `SOURCE-DECIMAL-MARK` or `TARGET-DECIMAL-MARK` is missing
`SOURCE-DECIMAL-MARK = …`	decimal mark of the source; for EDIFACT with UNA the character from UNA applies
`CONST ~~NAME = value`	constant, for example `CONST ~~INVOICE = 380`
`TEMPLATE ~~NAME FIELD.…`	field definition template, see TARGET-CONF
`CHAR-LIST.NAME = …`	character list for `TRANSLATE` and `REMOVE`, see below
`VALIDATE = "OFF" / "ON" / "STRICT"`	check of the EDIFACT side against the converter's directory, see below. Without a value `OFF`

Without `TARGET-DECIMAL-MARK` and `DECIMAL-MARK` the dot applies. **Every other entry is a finding** – a typo in GLOBAL-CONF does not go unnoticed.

Character lists

```
CHAR-LIST.INHOUSE-CHARS     = X"E4", X"C4", X"F6", "'", ":", "+", "?"
CHAR-LIST.EANCOM-CHARS      = X"84", X"8E", X"94", "?'", "?:", "?+", "??"
CHAR-LIST.NON-NUMERIC-CHARS = X"00" - X"2F", X"3A" - X"FF"
CHAR-LIST.LOWER             = "a" - "z", "ä", "ü"
```

A character list is an ordered, comma-separated sequence of entries:

- `X"nn"` or `"Text"`: one entry, which may also hold several characters (`"?'"`)
- `X"nn" - X"nn"` or `"a" - "z"`: a range of characters

A comma inside quotation marks does not separate.

Check against the EDIFACT directory

`VALIDATE` switches on the check of the EDIFACT side against the UN/EDIFACT directory selected for the converter – segment structure, formats and code lists of the messages, the envelope and the counters.

Value	Meaning
`OFF` (default)	do not check. Without a value everything stays as before
`ON`	check and **only report**: every finding becomes a warning, the output is created anyway (return value 11)
`STRICT`	check and **insist on errors**: a finding of severity "error" aborts, nothing is written (return value 10). Hints and warnings are reported as with `ON`

The side that is EDIFACT is checked – for **outbound** the generated output before it is written, for **inbound** the input that was read. For outbound a finding therefore still prevents a faulty file from reaching the partner.

Checked are:

- **every message, from its `UNH` to its `UNT`**, against the structure of its message type in the directory;
- **the envelope** – `UNB` and `UNZ`, with groups `UNG` and `UNE` – against the directory of service segments;
- **the counters:** the `UNT` must state the segments of its message, `UNH` and `UNT` included, and refer to the reference of its `UNH`; the `UNZ` the number of messages (or groups) and the reference from the `UNB`. A wrong counter makes the partner reject the file and counts as an error with `STRICT`.

Codes from a foreign list are not checked: if the element names a responsible agency other than UN/ECE, for example EAN in `IMD+F+ANM::9`, the code is not in the UN/EDIFACT directory; format and structure are still checked.

Which directory applies is set on the converter: fixed or from the UNH of the message. If the stated release is not included, D.01B is used instead – as in the editor. If the EDIFACT side cannot be read at all, that is a finding too and is not concealed.

The findings appear in the log of the run and in the message, at the same place as `WARNING` and `ERROR` from the definition.

3.4 SOURCE-CONF

Entry	Meaning
`RECORD-DELIMITER = X"0D0A"`	record separator. With `X"0D0A"` files that end with LF only are read as well
`FIELD-DELIMITER = ";"`	field separator; `GET_VALUE (n)` counts by it
`GROUP-DELIMITER = "+"`	group separator; only with `FORMAT = EDIFACT`, if it differs from the standard
`NO-RECORD-ID`	the source has no record type identifier of its own
`FIXED-LENGTH`	**fixed record length**: the source has no field separators, every field is at a fixed position, see below
`FORMAT = CSV` / `EDIFACT` / `JSON` / `XML`	structure of the source file; without a value `CSV`, i.e. a file with separators or fixed record length. `EDIFACT` see below
`RECORD-PATH = /order.items`	for JSON and XML the path to the source records, see below
`TEMP-TABLE.Name`	a **temporary table**, filled with `INSERT` or `SQL-LINK`, read with `WHILE NEXT`; see below
`SKIP-RECORDS = 1`	skips this many records at the start of the file. Intended for a **header line** that names the columns and is not a data record; without a value nothing is skipped
`RECORD.Name`	name of the source record
`FIELD.Name.Format`	the **first** field takes the first field of every source record and is used to check the record type, for example `IF SOURCE-DATA = "NAD"`; for EDIFACT this is the segment tag (`IF S-SEG = "LIN"`)
further `FIELD` lines	source fields at fixed positions; they are read, but are accessed via `GET_VALUE` or `VALUE`

An empty record at the end of the file, created by a trailing record separator, is not processed.

Fixed record length as source

Many ERP systems deliver a file without separators in which every field is at a fixed position. `FIXED-LENGTH` in SOURCE-CONF reads such a file:

```
::SOURCE-CONF
: RECORD-DELIMITER = X"0D0A"
: FIXED-LENGTH
: RECORD.INHOUSE
:    FIELD.RECORD-TYPE.X1              position  1- 1
:    FIELD.CUSTOMER-NO.N6              position  2- 7
:    FIELD.NAME.X20                    position  8-27
:    FIELD.DATE.N8                     position 28-35
```

- **The positions follow from the sizes** of the preceding fields; they are not stated separately.
- **Values are read by field name**: `IF RECORD-TYPE = "K"`, `NAD-CUSTNO = CUSTOMER-NO`. All fields from SOURCE-CONF are filled, not just the first. `GET_VALUE (n)` still returns the n-th field as well.

- **Trailing spaces** belong to the padding and are removed; leading ones stay, because that is where numbers carry their leading zeros.
- A record that **ends early** is not an error: the remaining fields are empty – like a missing field in a CSV source.
- `FIXED-LENGTH` together with `FIELD-DELIMITER` or `FORMAT = EDIFACT` is a finding; every field needs a size.

The **other direction** needs no switch: a target record in TARGET-CONF without separators between the fields already is a fixed-length record, because every field without `.TRIM` is brought to its size.

JSON and XML as source

`FORMAT = JSON` or `XML` reads a file that is a tree instead of a sequence of lines. Values are addressed there via **paths**.

```
::SOURCE-CONF
: FORMAT = JSON
: RECORD-PATH = /order.items

::RUN-CONF
: ORDERNO     = /order.number
: GLN-BY      = /order.partner[type = "BY"].gln
: LINENO      = line
: EAN         = article.ean
: QTY         = quantity
: TEXT        = texts[language = "EN"].text
: WRITE ROW
```

A path consists of names separated by dots:

Notation	Meaning
`/order.number`	from the **root** of the document – this is how header data is reached from any record
`article.ean`	in the **current source record**
`partner[type = "BY"].gln`	**selection by a neighbouring field**: the entry whose `type` is "BY"
`partner[@type = "BY"].gln`	the same in XML, where `type` is an **attribute**
`texts[1].text`	selection by **position**, counted from 1 as everywhere in the converter

Selection by a neighbouring field is the main way, not the position. It is the direct relative of the EDIFACT patterns the converter already knows:

EDIFACT	JSON / XML
`NAD+BY+VALUE`	`partner[type = "BY"].gln`
`IMD+*++*+:::VALUE::DE`	`texts[language = "DE"].text`
`DTM+137:VALUE`	`dates[kind = "137"].date`

Both say the same: *take the entry that is marked like this and read this field from it.* A position remains possible but is rarely what is meant – nobody can tell from the definition what is at position 1, and an additional entry in the file would silently make it read the wrong thing.

What a source record is is defined by `RECORD-PATH`: every match of this path is a record for which RUN-CONF runs once. Without a value the whole document is a single record – right for a file that contains only one transaction. There is no record type as with CSV; the structure of the file already says what a record is.

More:

- **Lists are not a level of their own.** `"items": [ {…}, {…} ]` becomes two entries named `items`; in XML two `<item>` elements are side by side. That is why the same path means the same thing in both formats.
- **Numbers arrive as they are in the file** – `12.50` does not become `12.5`. A `null` becomes the empty value.

- **An XML element without child elements is a value**, even if it carries attributes: `text[@language = "DE"]` returns the text right away. Explicitly this works with `#text` as the last step: `amount.#text`.
- **A namespace prefix is skipped when reading**: `rsm:ExchangedDocument` and `ExchangedDocument` match the same element.
- **A DTD is rejected** (`<!DOCTYPE …>`): the file comes from outside, and entities in it could multiply it while reading. The run stops with the message that the XML file cannot be read.
- **If a path does not match, the target field stays unchanged** – the same rule as for a pattern with `VALUE`.
- **If `RECORD-PATH` finds nothing**, the run aborts with a message and names what is at the top level of the file. A silent "zero records" would hardly be noticed.
- **A single name is a field first:** `EAN = QTY` takes over the field `QTY`. Only if the name would be the target field itself – `QTY = qty` – is the path into the file meant. A path with a dot or a selection (`article.ean`) is always a path.
- `FIELD-DELIMITER`, `GROUP-DELIMITER` and `FIXED-LENGTH` together with `FORMAT = JSON` or `XML` are a finding.

EDIFACT as source

`FORMAT = EDIFACT` reads the source as EDIFACT:

```
::SOURCE-CONF
: FORMAT = EDIFACT
: NO-RECORD-ID
: RECORD.S-SATZ
:    FIELD.S-SEG.X3
```

- **UNA** at the start of the file sets the separators and is not a source record itself. Without UNA the standard ones apply – segment terminator `'`, group separator `+`, field separator `:` – or whatever `RECORD-`, `GROUP-` and `FIELD-DELIMITER` specify instead.
- The **release character** is `?` or the one from UNA: `?+` is a plus and does not separate, `?'` does not end a segment, `??` is a question mark. In the values returned by `GET_VALUE` and `VALUE` the release is removed. A source without a group separator, such as CSV, has no release character.
- **Line breaks** (CR, LF) are only formatting and are skipped – after a segment, before the segment terminator and even in the middle of a segment if the file is wrapped after 80 characters. Only if UNA or `RECORD-DELIMITER` itself makes CR or LF the segment terminator do they separate.
- **Every segment needs a tag** of capital letters and digits, for example `UNH` or `BGM`. If a segment starts with something else – usually because the file is separated differently than the definition assumes – the conversion aborts with "Segment n is not an EDIFACT segment" instead of silently delivering nothing.
- **Values are read with patterns** that look like the message itself, see VALUE: `NAD+BY+VALUE`, `DTM+137:VALUE`, `QTY+21:VALUE`. A pattern checks its segment and its qualifier itself. A branch `IF S-SEG = "DTM"` is not needed for that.
- **`GET_VALUE` is a finding in an EDIFACT source.** It counts positions, and without EDIFACT knowledge one cannot tell what `GET_VALUE (3, 1)` reads. The finding names the equivalent pattern. If the assignment is inside a branch on a segment, it carries a suggestion that the editor applies on double-click. Every skipped position becomes a `*`, and the `*` should be replaced by the qualifier.

written	suggested in the finding	better with qualifier
`GET_VALUE (3, 1)` in NAD	`NAD+**+VALUE`	`NAD+BY+VALUE`
`GET_VALUE (2, 2)` in DTM	`DTM+**:VALUE`	`DTM+137:VALUE`
`GET_VALUE (4, 1)` in LIN	`LIN+***+VALUE`	`LIN+*++VALUE`
`GET_VALUE (4, 4)` with language check in IMD	`IMD+*****:*:*:VALUE`	`IMD+***+:::VALUE::DE`

- **Explanation in the editor:** if you point the mouse at a pattern or at a segment tag such as `"NAD"`, the editor explains it from the UN/EDIFACT directory. It names the segment, the position that is read and the meaning of the

qualifiers. If a test file is selected, it also shows what the pattern reads in it. The directory (for example D.01B or D.02B) follows the UNH of the test file. The names are those of the official directory and therefore English.

- **Patterns from the sample file:** the "Sample file" tab shows the test file, one segment per line. If the cursor is on a value, the editor shows the pattern that reads it and what it matches in the whole file. "Insert" or a double-click puts it into the definition. Positions before it stay as fixed values if they are qualifiers, i.e. if the directory has a code list for their data element. This produces `NAD+BY+VALUE`, but `LIN+*++VALUE` for the line number in LIN. With "also check what follows" the short codes after the value are added, for example `IMD+F+ANM:*+:::VALUE::DE`. In a CSV source the result is `GET_VALUE (n)`.

Numbers from the source

The decimal mark of the source comes from UNA, otherwise from `SOURCE-DECIMAL-MARK`, otherwise from `DECIMAL-MARK`, otherwise it is the dot.

It is only interpreted where a number is needed: in the number formats `N`, `F`, `S`, `Z` and `-`, in `CALCULATE` and in comparisons. A text field keeps `12,5` as it came; this way a reference or an article text with a comma stays untouched.

UNA	Source value	Field `Z15(3).TRIM`	Field `X10.TRIM`
`UNA:+,? '`	`12,5`	`12.5`	`12,5`
`UNA:+,? '`	`12.5`	`12.5`	`12.5`
`UNA:+.? '`	`12,5`	`12,5` (not a number)	`12,5`

The output always uses the decimal mark of the output (`TARGET-DECIMAL-MARK`).

Temporary tables

A temporary table collects values during the run to process them again later – usually in END-CONF – for example the line items of several orders for a separate file. It only lives during one run and is never written itself.

```
::SOURCE-CONF
: RECORD.SOURCE
:    FIELD.RECORD-TYPE.X1
: TEMP-TABLE.ITEMS
:    FIELD.T-NO.X10
:    FIELD.T-EAN.X13

::RUN-CONF
: IF RECORD-TYPE = "P"
:    T-NO  = TMP-NO
:    T-EAN = GET_VALUE (2)
:    INSERT ITEMS
: END-IF

::END-CONF
: CHANGE TARGET-FILE = "ORD_{ZAEHLER}_{DATUM}_{ZEIT}.csv"
: WHILE NEXT ITEMS
:    R-NO  = T-NO
:    R-EAN = T-EAN
:    WRITE ROW
: END-WHILE
```

Part	Meaning
`TEMP-TABLE.Name`	declares a table in SOURCE-CONF. The following `FIELD` lines are the fields of a row, up to the next `TEMP-TABLE` or `RECORD`. It is therefore best placed after the fields of the source
`INSERT Name`	stores the current values of the table fields as a new row and resets the fields to their initial value, as `WRITE` does with a target record
`INSERT Name NO-INIT`	stores without resetting
`WHILE NEXT Name` … `END-WHILE`	runs once per row, in the order of insertion. Before each pass the values of the row are in the table fields. An empty table skips the loop

The table fields are addressed by their names like all fields. Their name may therefore not also be a field from TARGET-CONF, from SOURCE-CONF or from another table. Further findings are a table without fields, an `INSERT` or `WHILE NEXT` for a table that does not exist, an `INSERT` into the table the surrounding `WHILE` is currently running over, and two nested `WHILE` loops over the same table. The number of rows is fixed at the start of the loop.

Data from SQL Server (SQL-LINK)

`SQL-LINK` queries a **Microsoft SQL Server** and makes the result available as a temporary table – for example to fetch the items or the delivery address for a customer number from the input out of the ERP system. The connection is declared in a section of its own, `::LINK-CONF`, the table in SOURCE-CONF:

```
::LINK-CONF
: SQL-CONNECTION = "Server=sql01;Database=ERP;User Id=edi_reader;TrustServerCertificate=True"
: SQL-PASSWORD   = "dpapi:AQAAANCMnd8BFdERjHoAwE…"

::SOURCE-CONF
: RECORD.SOURCE
:    FIELD.CUSTNO.X10
: TEMP-TABLE.ITEMS
:    FIELD.I-EAN.X13
:    FIELD.I-TEXT.X35

::RUN-CONF
: SQL-LINK ITEMS = "SELECT EAN, Description FROM dbo.Items WHERE CustomerNo = {CUSTNO}"
: WHILE NEXT ITEMS
:    L-EAN  = I-EAN
:    L-TEXT = I-TEXT
:    WRITE LINE
: END-WHILE
```

Part	Meaning
`SQL-CONNECTION = "…"`	in LINK-CONF: the converter's connection as SQL Server understands it (`Server`, `Database`, `User Id`, `Integrated Security`, `Encrypt`, `TrustServerCertificate` …), **without the password**. Exactly one per converter
`SQL-PASSWORD = "…"`	in LINK-CONF: the password for the connection, **encrypted**. Without it, the converter signs in with the Windows account it runs under (`Integrated Security=True`)
`SQL-LINK table = "SELECT …"`	in START-, RUN- or END-CONF: runs the query and **replaces** the rows of the table with the result. `WHILE NEXT` then reads it like any temporary table

Columns and fields: The first column of the result fills the first field of the table, the second the second, and so on. Column names do not matter; their **number must match**, otherwise the run stops with a message. `NULL` becomes an empty value, numbers arrive as numbers, `bit` as `1` or `0`. A `date` arrives as CCYYMMDD, a `time` as HHMMSS, other timestamps as CCYYMMDDHHMMSS – depending on the column type, not on the value. The query itself can deliver another form, e.g. with `FORMAT(Date, 'dd.MM.yyyy')`.

Values from the definition are written as placeholders in braces: `{FIELD}` for a field or variable, `{~~CONSTANT}` for a constant, including `{~~PARTNER-GLN}`. A placeholder is sent to the server as a **parameter** and never becomes part of

the query text, so a quote or semicolon in the data cannot change the query. For the same reason a placeholder is not put in single quotes: `LIKE '%' + {SEARCH} + '%'`, not `LIKE '%{SEARCH}%'`.

Read only. `SQL-LINK` runs nothing but **a single** `SELECT`. When the definition is read, the following are findings:

- a query that does not start with `SELECT` – including `WITH` and `EXEC`;
- a semicolon, a comment (`--`, `/* */`) and a variable with `@`;
- any word that writes, changes permissions, executes code or reaches other servers or files: `INSERT`, `UPDATE`, `DELETE`, `MERGE`, `TRUNCATE`, `INTO`, `DROP`, `ALTER`, `CREATE`, `EXEC`, `EXECUTE`, `DECLARE`, `SET`, `BEGIN`, `COMMIT`, `ROLLBACK`, `SAVE`, `WAITFOR`, `GO`, `USE`, `GRANT`, `REVOKE`, `DENY`, `SHUTDOWN`, `KILL`, `DBCC`, `BACKUP`, `RESTORE`, `RECONFIGURE`, `CHECKPOINT`, `ENABLE`, `DISABLE`, `BULK`, `WRITETEXT`, `UPDATETEXT`, `OPENROWSET`, `OPENDATASOURCE`, `OPENQUERY`, `OPENXML`.

Texts in single quotes and names in square brackets are exempt. If a column is named like one of these words, put it in square brackets: `SELECT [Update] FROM …`. During the run the same check is repeated before every query; the query also runs in a transaction that is **always rolled back**, and if the server reports changed rows, the run stops. The safest setup is still a database user that may only read (role `db_datareader`).

The password is first typed in plain text in the editor – `SQL-PASSWORD = "secret"`. On **saving** and before a **test run** the editor replaces it with `"dpapi:…"`: encrypted with Windows (DPAPI) and bound to the computer, like the mail password. Editor and converter read it under every user account of this computer. A definition copied to another computer does not carry the password; it is entered and saved again there. Until then a plain-text password is a finding, as is a `Password=` or `Pwd=` in `SQL-CONNECTION`.

Further findings and aborts:

Case	Result
`SQL-LINK` without `SQL-CONNECTION`, for a table that does not exist, inside `WHILE NEXT` over the same table, with an unknown placeholder	finding
`SQL-CONNECTION` or `SQL-PASSWORD` twice, empty or without quotes, in TARGET-CONF instead of LINK-CONF; `SQL-PASSWORD` without `SQL-CONNECTION`; any other entry in LINK-CONF	finding
server unreachable, sign-in or query fails, number of columns does not match, password cannot be decrypted on this computer	conversion error (return code 7); the message names the table and the server's text
more than 100,000 rows or longer than 60 seconds	conversion error; a condition in `WHERE` narrows the query

Every `SQL-LINK` queries the server again; in RUN-CONF that is once per source record. A value that is the same for all records is better fetched once in START-CONF.

3.5 TARGET-CONF

```
: RECORD.SEG-NAD
:    ~~FILL-10                          = "NAD+BY+"
:    FIELD.NAD-ILN.N13
:    ~~FILL-10                          = "+"
:    FIELD.NAD-NAME.X35.TRIM
:    ~~SEG-END
```

Entry	Meaning
`RECORD.Name`	start of a target record (segment)
`FIELD.Name.FormatSize[(decimals)][.attribute][ = initial value]`	a field of the record
`~~TEMPLATE [= value]`	a constant from a template (`TEMPLATE`); without a value its initial value applies. It goes into the output **unchanged**
`FORMAT = CSV` / `JSON` / `XML`	structure of the target file; without a value `CSV` , i.e. records one after another
`FORMAT = LAYOUT`	XML from the output layout in LAYOUT-CONF, see below
`NAMESPACE prefix = "URI"`	with `FORMAT = XML` : declares a prefix for the paths; without a prefix the default namespace, see below
`… -> path`	for a tree target: where the record or field belongs, see below

A record is assembled in the order of its parts. Fields appear in the form their format and attribute require; constants unchanged.

JSON and XML as target

A tree has no order from which it would follow where a field belongs. That is why with `FORMAT = JSON` or `XML` every record and every field says so with an **arrow**:

```
::TARGET-CONF
: FORMAT = JSON
: RECORD.HEADER   -> /order
:    FIELD.H-NUMBER.X35       -> number
:    FIELD.H-DATE.N8          -> date
: RECORD.ITEM     -> /order.items[]
:    FIELD.I-LINENO.Z6        -> line
:    FIELD.I-EAN.N13          -> article.ean
:    FIELD.I-ARTNO.X35        -> article.no
:    FIELD.I-QTY.Z15(3)       -> quantity
:    FIELD.I-REMARK.X70.IF-CHANGED  -> remark
```

results in

```
{ "order": {
    "number": "780307082681", "date": "20260804",
    "items": [
      { "line": "1", "article": { "ean": "5996227007944", "no": "50091040010" },
        "quantity": "24", "remark": "urgent" } ] } }
```

The paths are the same as on the source side:

- **On the record** it says where every `WRITE` attaches an entry. The last step is the name of the entry; before it is where it belongs; missing intermediate levels such as `article` are created automatically.
- **`[]` at the end** says that there is a **list**. Then a list is created even for a single line item – otherwise the reading system would get something different for an order with one line item than for one with two. Without `[]` a single entry is created, as for the header record.
- **On the field** the path within the entry is stated. With a slash it starts from the root – that is where header data that exists only once ends up. If the same path is described several times, the last value applies.
- **In XML a path with `@` becomes an attribute**: `-> @no` writes `<item no="1">`. If the tree has several entries at the top level, an element `daten` is put around them so that the file is valid XML.
- **`#text` is the text of an element.** If one path ends with `#text` and another with `@` at the same element, it carries both: `-> ram:TaxTotalAmount.#text` and `-> ram:TaxTotalAmount.@currencyID` give `<ram:TaxTotalAmount`

`currencyID="EUR">19.00</ram:TaxTotalAmount>`. `#text` may only be the last step of a field path. It may also be left out: `-> ram:TaxTotalAmount` and `-> ram:TaxTotalAmount.@currencyID` give the same, in either order.

Namespaces

Many XML formats – such as ZUGFeRD/XRechnung (CII) – require every element to belong to a namespace. `NAMESPACE` declares a prefix for it, which is then used in the paths:

```
::TARGET-CONF
: FORMAT = XML
: NAMESPACE rsm = "urn:un:unece:uncefact:data:standard:CrossIndustryInvoice:100"
: NAMESPACE ram = "urn:un:unece:uncefact:data:standard:ReusableAggregateBusinessInformationEntity:100"
: NAMESPACE udt = "urn:un:unece:uncefact:data:standard:UnqualifiedDataType:100"
: RECORD.HEADER -> /rsm:CrossIndustryInvoice.rsm:ExchangedDocument
:    FIELD.H-NUMBER.X35       -> ram:ID
:    FIELD.H-DATE.X8          -> ram:IssueDateTime.udt:DateTimeString.#text
:    FIELD.H-DATEFORMAT.X3    -> ram:IssueDateTime.udt:DateTimeString.@format
```

gives

```
<rsm:CrossIndustryInvoice xmlns:rsm="urn:…:CrossIndustryInvoice:100" xmlns:ram="urn:…" xmlns:udt="urn:…">
  <rsm:ExchangedDocument>
    <ram:ID>RG106</ram:ID>
    <ram:IssueDateTime>
      <udt:DateTimeString format="102">20261026</udt:DateTimeString>
    </ram:IssueDateTime>
  </rsm:ExchangedDocument>
</rsm:CrossIndustryInvoice>
```

- The declarations are placed on the outermost element, with the prefixes from the definition.
- `NAMESPACE = "URI"` **without a prefix** declares the default namespace. It applies to every element without a prefix; an attribute without a prefix belongs to no namespace, as usual in XML.
- The prefix `xml` (e.g. `@xml:lang`) is reserved by XML itself and needs no declaration.
- Elements appear in the order in which they are written. A schema that prescribes the order – as CII does – therefore determines the order in which the records must be written.

Short names, blocks and fixed values

In deeply nested formats such as CII the paths get long and repeat themselves. Four notations keep TARGET-CONF readable; they change nothing in the file produced:

Entry	Meaning
`PATH NAME = /path`	a short name for a path from the root. At the start of any path – also in another `PATH` – `$NAME` then stands for it
`NODE path` … `END-NODE`	a block within a record: every path in it without a slash is attached to the path of the block. Blocks may be nested
`FIX path = "value"`	a fixed value without a field name of its own, e.g. `FIX ram:TypeCode = "VAT"`
`path[@attribute="value"]`	at the end of a field path: a fixed attribute at the element of the field. It only appears if the field appears

```
: PATH INVOICE     = /rsm:CrossIndustryInvoice
: PATH TRANSACTION = $INVOICE.rsm:SupplyChainTradeTransaction

: RECORD.ITEM -> $TRANSACTION.ram:IncludedSupplyChainTradeLineItem
:    FIELD.I-LINENO.X6                   -> ram:AssociatedDocumentLineDocument.ram:LineID
:    NODE ram:SpecifiedTradeProduct
:       FIELD.I-GTIN.X14.IF-CHANGED      -> ram:GlobalID[@schemeID="0160"]
:       FIELD.I-NAME.X100                -> ram:Name
:    END-NODE
:    NODE ram:SpecifiedLineTradeSettlement.ram:ApplicableTradeTax
:       FIX ram:TypeCode                 = "VAT"
:       FIELD.I-RATE.Z6(2)               -> ram:RateApplicablePercent
:    END-NODE
```

- A path with a slash starts at the root inside a block too; so does a block with a slash (or `$NAME`). That way a record also writes header data elsewhere.
- Without a GTIN, `ram:GlobalID` is left out together with `schemeID` – no helper field for the attribute and no `IF … NOT EMPTY` in RUN-CONF is needed.
- A record ends with the next `RECORD`; a block must be closed with `END-NODE` by then.
- For a ZUGFeRD invoice the **output layout** (below) is the simpler way: the order of the schema is then in the layout, not in RUN-CONF. The samples `Samples\ZUGFERD-CSV.DEF` and `Samples\ZUGFERD-INVOIC.DEF` work like that.

There is no padding in a tree target. In a JSON or XML file the target system reads values, not columns; a customer number as `"0000000000024"` or a name with trailing spaces would be wrong there. The size still applies as a **maximum** – too long remains an error – and decimal places are rounded as usual. A field with `.IF-CHANGED` keeps its meaning: if it is still in its initial state, it does not appear at all. But it is not padded any more than any other – a date in an `N8.IF-CHANGED` appears as `20260804` in the file, not as `0020260804`.

Findings: a record or field without an arrow, an arrow without `FORMAT = JSON` / `XML`, something after the arrow that is not a path, and a constant (`~~TEMPLATE`) in a tree target – there are no separators and record terminators there. Likewise a prefix without `NAMESPACE`, a prefix declared twice, `NAMESPACE` without `FORMAT = XML`, and `#text` that is not the last step of a field path. With the short notations: an unknown or duplicate `PATH`, a `PATH` that does not start at the root, `END-NODE` without `NODE`, a `NODE` still open at the end of a record, `NODE` or `FIX` before the first `RECORD`, and a selection in square brackets that is not a fixed attribute at the end of a field path.

Output layout (FORMAT = LAYOUT)

With an XML target using arrows, the file is built in the order of the `WRITE`s. If a schema prescribes the order of the elements – as CII does for ZUGFeRD and XRechnung – RUN-CONF then has to follow the schema rather than the input. An **output layout** turns this around: it is the target file itself, with placeholders. RUN-CONF only collects the values; the layout is filled **once at the end** of the run. The order in which the records arrive in the input does not matter.

```
::TARGET-CONF
: FORMAT = LAYOUT
: RECORD.RECHNUNG
:    FIELD.KO-NUMMER.X35
: RECORD.POSITION
:    FIELD.PO-NR.X6
:    FIELD.PO-NAME.X100

::RUN-CONF
: IF SATZART = "K"
:    KO-NUMMER = GET_VALUE (2)
: END-IF
: IF SATZART = "P"
:    PO-NR     = GET_VALUE (2)
:    PO-NAME   = GET_VALUE (3)
:    WRITE POSITION
: END-IF

::LAYOUT-CONF
<rechnung xmlns:lenic="urn:lenic:edikonverter:layout">
  <nummer>{KO-NUMMER}</nummer>
  <position lenic:for-each="POSITION" nr="{PO-NR}">
    <name>{PO-NAME}</name>
  </position>
</rechnung>
```

Layout and definition

The layout belongs to the definition: TARGET-CONF with records and fields, LAYOUT-CONF with the XML, plus SOURCE-CONF and RUN-CONF for the source – all in one converter. What it outputs is therefore in the converter itself; it does not name another converter. A name after `FORMAT = LAYOUT` is a finding.

If a second source gets the same invoice, its definition takes over TARGET-CONF and LAYOUT-CONF. The fields are the interface: if they are the same, the LAYOUT-CONF can be swapped too, for example for another profile.

Records

TARGET-CONF describes records and fields with their format – without arrows, `PATH`, `NODE`, `FIX`, `NAMESPACE` and constants. A record is one of two kinds:

Record	recognized by	`WRITE`	in the layout
single record	no `lenic:for-each` names it	not allowed (finding)	its fields with the value they have at the end of the run
repeating record	an element carries `lenic:for-each="RECORD"`	one entry per `WRITE`	the element once per entry, with its values

A `WRITE` keeps the record's values and resets it as usual. **A field that was never assigned is empty – a number field as well.** The zeros for empty number fields do not apply to a layout, otherwise no element would ever be left out; if you want a 0, assign it or give the field a default.

LAYOUT-CONF

- The layout starts at the first line beginning with `<`; separator lines and comments before it are allowed. From there on every line counts as it stands, without `:` in front, up to the next section. Comments are XML comments.
- It is **well-formed XML** – otherwise a finding with its line.
- The control attributes belong to the namespace `urn:lenic:edikonverter:layout`, usually with the prefix `lenic`. Declaration and attributes do not appear in the output; all other namespaces stay as they are in the layout.

Notation	becomes
`{FIELD}`, `{RECORD.FIELD}`	the field's value – as in a tree target: size as a maximum, rounded, not padded; `&`, `<`, `"` are escaped for XML
`{~~CONSTANT}`	the value of a constant, such as `{~~PARTNER-GLN}`
`{{`, `}}`	a curly brace

Placeholders go into the text of an element and into attribute values, also several of them and mixed with fixed text.

Control attributes

Attribute	Effect
`lenic:for-each="RECORD"`	the element once per written entry, in the order of the `WRITE`s; without entries not at all. Inside, the fields of this record, of the outer loops and of all single records are available. A record may repeat in several places
`lenic:where="condition"`	in a loop inside a loop: which outer entry an inner one belongs to, see below
`lenic:distinct="FIELD"`	with `for-each`: only the first entry per distinct value of the field; empty values do not count – for example a list of the delivery notes of all line items
`lenic:if="condition"`	the element only if the condition is true; with `for-each` per entry
`lenic:keep-empty="true"`	the element stays even when empty

A condition is built as in `IF`; text goes in single quotes (`lenic:if="KO-ART = '384'"`), and instead of `<` write `<` in the attribute or turn the comparison around.

Which outer entry an inner one belongs to: Without `where`, to the one written **last before it** – an allowance belongs to the line item before it. If none was written before, the run stops (return code 7). With `where`, the condition decides, no matter when the two were written: `lenic:where="PR-POSITION = PO-NR"`. It compares fields of the inner entry with those of the **next** outer loop and the single records. If an entry matches no outer entry or several, the run stops with a message naming it – an allowance does not silently disappear.

Leaving out

An element that is empty after filling is left out – no text other than whitespace, no child element. Attributes do not count: without a GTIN, `<ram:GlobalID schemeID="0160">{PO-GTIN}</ram:GlobalID>` disappears entirely. This continues outwards; an element whose children have all been left out is left out as well. An attribute that consisted only of empty placeholders is left out too. `lenic:if` is therefore only needed where fixed content would never let an element become empty – such as `<udt:Indicator>false</udt:Indicator>`.

Output

The XML declaration names the character set of the output file, as with an XML target; a declaration in the layout is replaced. Indentation is redone. A run produces one document; END-CONF already sees all values and is the place for checks across the whole document. Output type ZUGFeRD PDF works as with any XML target, and the profile in the PDF metadata comes from the XML.

Findings

When reading a layout: not well-formed XML; a placeholder that names neither a field from TARGET-CONF nor a declared constant; an open `{`; `for-each` with an unknown record or on the outermost element; a field of a repeating record outside its loop; `where` without an outer loop or with a field of a loop further out; `where` or `distinct` without `for-each`; `distinct` with a field of another record; a condition that is not one; an unknown control attribute; arrows, `PATH`, `NODE`, `FIX`, `NAMESPACE` or constants in TARGET-CONF.

Also: a name after `FORMAT = LAYOUT`; LAYOUT-CONF without `FORMAT = LAYOUT`; `WRITE` of a single record; `CHANGE TARGET-FILE`.

Shipped are two definitions with their own layout: `ZUGFERD-CSV.DEF` from an in-house CSV (profile EN 16931) and `ZUGFERD-INVOIC.DEF` from an EANCOM INVOIC (profile EXTENDED). TARGET-CONF and LAYOUT-CONF of both profiles also come as **copy templates**, `Samples\ZUGFERD-VORLAGE-EN16931.LAYOUT` and `Samples\ZUGFERD-VORLAGE-EXTENDED.LAYOUT` – to take over into a definition, not as a converter of their own.

Field names

A field is addressed by its name. A preceding record name is allowed but not evaluated: `SEG-NAD.NAD-ILN` and `NAD-ILN` mean the same field. **Field names must therefore be unique across all records.**

Field formats

The size is a size in bytes. Where decimal mark and sign are shown, they count.

Format	Contents	Example	Result
`X`	text	`X10` with `Smith Ltd`	`Smith Ltd␣`
`U`	text, converted to capital letters	`U10.TRIM` with `Smith Ltd`	`SMITH LTD`
`N`	digits only, leading zeros, no sign	`N6` with `1`	`000001`
`F`	like `N`, with a virtual decimal point: decimals without separator	`F6(2)` with `1.1`	`000110`
`S`	like `F`, sign in the first position	`S14(4)` with `145.75`	`+0000001457500`
`Z`	number with visible decimal mark, minus for negative values, padded right-aligned with spaces	`Z8(3)` with `-123.456`	`-123.456`
`-`	like `Z`, the sign is always shown	`-8(3)` with `123.456`	`+123.456`

Further rules:

- **Excess length is an error.** Nothing is cut off; the conversion aborts with a message that names record, field, value and permitted size.
- **Too many decimal places are rounded commercially**, away from zero.
- A negative value in `N` or `F` is an error.
- Whatever cannot be read as a number goes out unchanged in number formats. Leading and trailing spaces are tolerated. The decimal mark of the source is understood, see "Numbers from the source".

Field attributes

Attribute	Meaning
none (or `.F`)	fixed length – **the default**; the field always has its size
`.TRIM`	variable: leading and trailing spaces are removed; with `Z` and `-` also trailing zeros and a decimal part that is zero. The size is only the maximum. Corresponds to EANCOM `an..35`
`.TRIM-END`	trailing spaces are removed
`.LEFT`	left-aligned, also for numbers
`.IF-CHANGED`	the field only appears if its value differs from the initial state

With `N`, `F` and `S` the leading zeros are part of the value; `.TRIM` finds nothing to remove there. For EDIFACT segments `X….TRIM` and `Z….TRIM` are therefore the usual forms, for example `FIELD.NAD-NAME.X35.TRIM`.

Initial value and initial state

Every target field has an initial state: the initial value after the equals sign, otherwise depending on the type the number `0` for `N`, `F`, `S`, `Z` and `-` and empty text for `X` and `U`. In the output this becomes for example `00000` for `N5` or five spaces for `X5`.

- At the start all target fields are in their initial state.
- **After every `WRITE` it applies again to the fields of the written record**, except with `WRITE … NO-INIT`.
- **An empty number field is written with zeros.** In the formats `N`, `F`, `S`, `Z` and `-` an empty value counts as `0` – regardless of whether the field was never filled or was explicitly assigned an empty value, for example with `FIELD = EMPTY` or by a `GET_VALUE` on an empty source field. This way a number field always contains a readable number, and a fixed-length field keeps its width.
- An empty **text field** (`X`, `U`), on the other hand, goes out empty and is at most padded with spaces to its fixed length.

3.6 START-CONF, RUN-CONF, END-CONF

START-CONF runs once before the first source record, RUN-CONF once per source record, END-CONF once after the last one. All three work on **the same fields**: a counter from START-CONF lives on until END-CONF.

The output is controlled exclusively by `WRITE`: a target record is created exactly when the definition writes it.

Assignments

Form: `field = expression`. The expression is exactly one of the following forms:

Expression	Result
`"Text"`, `X"hh"`, `~~CONSTANT`, integer	the value
`EMPTY`	empty value
`other-field`	the value of the other field
`GET_VALUE (n)`	the n-th field of the current source record of a CSV source; counted from 1, empty if missing. **A finding in an EDIFACT source**; there a pattern with `VALUE` applies, see 4
`GET_VALUE (g, n)`	the n-th field of the g-th group; a CSV source has exactly one group. A finding in an EDIFACT source
`CONCAT part part …`	the parts joined without separator; a part is `"Text"` or a field. **The parts are separated by spaces**, not by commas. As with `WARNING` they may be in parentheses: `CONCAT (A " " B)` is the same as `CONCAT A " " B`
`CALCULATE formula`, `CALCULATE (formula) …`	calculation with `+ - * /`, parentheses, numbers and fields; multiplication before addition. **Operators stand alone, with spaces around them**, because field names themselves contain hyphens. Rounding only happens on output, to the decimal places of the target field. Division by zero is an error
`DATE ()`, `DATE (format)`	today's date; without format `CCYYMMDD`. Formats: `CCYYMMDD`, `YYYYMMDD`, `YYMMDD`, `DDMMYY`, `HHMM`, `HHMMSS` – no free format texts
`TIME ()`, `TIME (format)`	time of day; without format `HHMMSS`, otherwise `HH`, `MM`, `SS` in the format text are replaced: `TIME (It is HH:MM)`
`CHECK_DIGIT (field)`	GS1 check digit (modulo 10) over the digits of the field
`SUBSTRING (start, length, value)`	excerpt from position `start` (counted from 1), at most `length` characters. The value is a field, a `"Text"`, a `~~CONSTANT` or `GET_VALUE (…)`: `SUBSTRING (5, 2, GET_VALUE (3))`. If the value does not reach the end of the excerpt, whatever is there is returned; if it starts beyond it, the result is empty. Any other form or start 0 is a finding
`FORMAT-DATE (value, "output")`, `FORMAT-DATE (value, "input", "output")`	converts a date. The value is a field, a `"Text"`, a `~~CONSTANT` or `GET_VALUE (…)`. Placeholders, upper or lower case: `CCYY` / `YYYY` / `JJJJ` year, `YY` / `JJ` two-digit year, `MM` month, `DD` / `TT` day, `DDD` / `TTT` day of year (001–366), `WW` calendar week according to ISO 8601, `HH` hour, `MI` minute, `SS` second; everything else appears in the output as it is. Instead of a format an EDIFACT format code works too: `101`, `102`, `203`, `204`, `616`. If the input format is missing, it is detected: 6, 8, 12 or 14 digits as `YYMMDD`, `CCYYMMDD`, `CCYYMMDDHHMI`, `CCYYMMDDHHMISS`, plus `DD.MM.CCYY` and `CCYY-MM-DD`. Example: `FORMAT-DATE (H-DATE, "DD.MM.CCYY")`, `FORMAT-DATE (GET_VALUE (3), "DD.MM.CCYY", "102")`. If the value does not match the input format exactly, only the digits count: `06.10.2026` is also read with `DDMMCCYY`, `06102026` also with `DD.MM.CCYY`. An empty value stays empty; a value that does not match even then or is not a valid date aborts the conversion. A format without placeholders is a finding
`LEN (value)`	length of the value in characters, as a number – for example for `IF` or `CALCULATE`. The value is a field, a `"Text"`, a `~~CONSTANT` or `GET_VALUE (…)`: `LEN (GET_VALUE (3))`. The content as it currently is in the field is counted, spaces included – not the field size from TARGET-CONF. An empty field has length 0. Any other form is a finding
`TRANSLATE list1 TO list2`	translates the target field's **own** value: the n-th entry of list1 becomes the n-th of list2; if list2 has only one entry, it applies to all. The longer entry takes precedence; an inserted value is not translated again
`REMOVE list`	the characters of the list are removed from the target field's own value
`pattern VALUE`	see below

Whatever does not match one of these forms is already a **finding** when the definition is read, not just an abort in the middle of the conversion. The finding names what is missing:

Line	Finding
`EAN = QUANITY`	there is no field or variable with this name – text belongs in quotation marks
`SENDER = ~~SENDR`	there is no constant with this name
`TOTAL = A + B`	calculations use `CALCULATE`: `CALCULATE A + B`
`PRICE = 3.5`	a number with decimal places is written in quotation marks or calculated with `CALCULATE`
`VALUE1 = UPPER (A)`	the function `UPPER` does not exist

A name is known if it appears in TARGET-CONF or SOURCE-CONF, or is assigned somewhere, counted with `INC` or used as the counting variable of `FOR`. With a JSON or XML source, a name that is not a field is a path in the file.

A variable is empty until it is assigned for the first time. This makes it possible to do something exactly once, for example at the first record of a kind:

```
: IF TMP-FIRST = EMPTY
:    WRITE PARTNER
:    TMP-FIRST                          = "J"
: END-IF
```

VALUE

`field = pattern VALUE` looks for the pattern **at the start of the current source record** and returns whatever stands from the position of `VALUE` up to the next field, group or record separator. A `*` in the pattern stands for any number of characters, as few as possible.

The pattern ends with a separator immediately before `VALUE`: `NAD+BY+VALUE`, `QTY+21:VALUE`. A name such as `TMP-RETURN-VALUE` is therefore never a pattern but always a field.

The pattern may continue after `VALUE`. Then this part must match too, otherwise the pattern does not match. This way a value is only read if the right thing follows, for example the article text only in German:

Source record	Statement	Result
`IMD+F+ANM::9+:::Wiesbauer Grillmix::DE`	`TEXT = IMD+*++*:::VALUE::DE`	`Wiesbauer Grillmix`
`IMD+F+ANM::9+:::Grillmix (FGY) 360g::HU`	ditto	does not match

Source record	Statement	Result
`UNB+UNOC:3+4012345000009:14+4000000000003:14+…`	`SENDER = UNB+*+VALUE`	`4012345000009`
ditto	`RECIPIENT = UNB+*++*+VALUE`	`4000000000003`
`DTM+137:20260911:102`	`DATE = DTM+*:VALUE`	`20260911`
`LIN+1++4012345000009:EN`	`EAN = LIN+2++VALUE`	does not match

If the pattern does not match, the field stays unchanged. This way several patterns can be applied to the same field one after another without a later failure erasing an earlier match:

```
: ADDRESS = NAD+CN+VALUE
: ADDRESS = NAD+DP+VALUE
```

A released separator (`?+`) does not end the value, and the release is removed in the value.

Statements

Statement	Meaning
`WRITE record`, `WRITE (record)`	writes the target record and resets its fields to the initial state
`WRITE record NO-INIT`, `WRITE (record) NO-INIT`	writes without resetting
`INC (field)`, `INC (field, n)`	increments a field by 1 or by n
`IF … [ELSE …] END-IF`	branch, nested as deeply as needed
`WARNING (text …)`	warning to the run log and the log file; the text is built as with `CONCAT`, placeholders such as `{FIELD}` insert values, see below. The conversion continues, the output is written
`ERROR (text …)`	error to the run log and the log file, text as with `WARNING`. The conversion continues so that all errors are reported, but **no output file is written**
`INSERT table [NO-INIT]`	stores the values of the table fields as a row in a temporary table, see 4
`WHILE NEXT table … END-WHILE`	loop over the rows of a temporary table, see 4
`SQL-LINK table = "SELECT …"`	fills a temporary table with the result of a query to SQL Server, read only, see 4
`FOR (I = start; condition; step) … END-FOR`	counting loop, see below
`CHANGE TARGET-FILE = "name"`	only in END-CONF and at most once, not inside a loop (`WHILE` or `FOR`): whatever is written afterwards goes to an **additional target file**, see below

Counting loop

```
: FOR (I = 1; I <= TMP-COUNT; I++)
:    FTX-NO = I
:    WRITE SEG-FTX
: END-FOR
```

The parentheses contain, separated by semicolons:

Part	Meaning
`I = start`	the counting variable gets the start value at the beginning: a number, a field or a `~~CONSTANT`
condition	a comparison as with `IF`, also with `AND` and `OR`. It is checked **before every pass**; if it is not true, execution continues after `END-FOR`. If it is not true at the start, the body never runs
step	after every pass: `I++` counts up by 1, `I--` down by 1, `I += n` and `I -= n` by n. n is a number, a field or a constant

`FOR (I = 0; I < TMP-COUNT; I++)` therefore runs with 0, 1, … up to one below TMP-COUNT. The counting variable is an ordinary field: the body can read it, name it in `WARNING` as `{I}` and also change it – if it sets it to the end, the loop is over after this pass. Loops can be nested, each with its own counting variable.

If a loop has not finished after 100,000 passes, the conversion aborts with a message instead of running forever – usually the step then counts in the wrong direction. A step that changes a variable other than the one being counted is already a **finding** when the definition is read.

Additional target file

`CHANGE TARGET-FILE = "name"` in END-CONF creates a second file. Everything written before goes to the output file of the configuration, everything after it to the additional file. It has the same format (`FORMAT` in TARGET-CONF) and the same character set as the output file.

- The name carries the same placeholders as the output file, see 7: `{ZAEHLER}`, `{ZAEHLER:6}`, `{DATUM}`, `{ZEIT}`. An unknown placeholder is a finding. A field or a constant may be used instead of the text. Its value may come from the input and therefore yields **only a file name**: `\`, `/`, `:` and other characters that Windows does not allow in a name become `_`, and so does a value consisting only of dots. Only a text in quotation marks can name a folder.
- A name without a folder, or with a relative folder, starts from the folder of the output file.
- `{ZAEHLER}` carries the same number as the output file so that both belong together: `OUT_17.csv` and `ORD_17.csv`. If the output file has no number, the additional file draws one of its own.
- An existing file is not overwritten. Then **neither** of the two files is written, and the run ends with return value 6. Likewise neither is created after an `ERROR`.
- The check against the EDIFACT directory (`VALIDATE`) applies to the output file.
- In the editor's test run the additional file appears below the output, with its name.

Placeholders in WARNING and ERROR

In the text of `WARNING` and `ERROR`, `{NAME}` inserts the current value of a field or variable, also with the record before it such as `{SEG-BGM.BGM-NR}`. `{~~NAME}` inserts the value of a constant. Any number of them may be used:

```
: WARNING ("Customer {TMP-CUST-NO} without name, order {SEG-BGM.BGM-NR}")
: ERROR ("Address {CUST-NO} missing for {~~PARTNER-NAME}")
```

A name that exists in the definition neither as a field nor as a filled variable or constant is a finding with a line. A field that has not been filled in the run yet results in empty text. Replacement only happens in the text of the definition; curly braces in the value of a field stay as they are. The notation as with `CONCAT`, `WARNING ("Customer without name: " TMP-CUST-NO)`, still works.

Conditions

```
: IF SOURCE-DATA = "NAD"
: OR SOURCE-DATA = "NAD2"
:    …
: END-IF
```

Comparison	Meaning
`A = B`	equal, see below
`A <> B`, `A != B`, `A NOT = B`	not equal, see below
`A < B`, `A > B`, `A <= B`, `A >= B`	comparison as numbers; whatever is not a number counts as 0
`A = EMPTY`	A contains nothing or only spaces
`A NOT EMPTY`, `A NOT = EMPTY`	the opposite

On the right of the comparison there is a field, a text in quotation marks, a `~~CONSTANT` or a number.

What is compared must be a name or a value. If one side contains something else – a misspelt `NOR` instead of `NOT`, a comment after the condition – that is a **finding**. Otherwise the rest of the line would count as text of its own, the condition would silently always be false, and the record after it would never be created. A comment therefore belongs on its own line: a `*` in the middle of a line is not one.

The name must also exist. A name is a field from TARGET-CONF or SOURCE-CONF, a variable that is assigned somewhere or counted with `INC`, or a constant `~~NAME`. A misspelt name such as `P-TXT` instead of `P-TEXT` is a **finding**; so is a text without quotation marks.

Equal and not equal depend on what is on the right:

- **Text in quotation marks:** it is compared character by character. `"+1"` is not `"1"`, and `"NAD"` is always the text, even if there is a field with that name.

- **Number, field or constant:** if both sides are numbers, the value counts. `0.00 = 0` and `+1 = 1` are true. Otherwise the comparison is character by character. An empty value is not a number, so empty is not equal to `0`.

A text is only equal if it is completely equal; texts of different length are never equal. Several comparisons are combined with `AND` and `OR`, also across continuation lines that start with `AND` or `OR`. They are evaluated **from left to right without precedence**.

3.7 Calling and return values

The converter is only called with the ID of a saved configuration: `LeNic.EdiKonverter.App.exe <ID>`. Direction, partner, input and output file and the converter are stored in the configuration. The return values 0 to 14 are listed in the user manual. Every run is logged in the database, with return value, files, number of source and target records and the complete message including `WARNING` and `ERROR`. The editor shows the log on the "History" page.

The name of the output file is a pattern with placeholders:

Placeholder	Result
`{ZAEHLER}`	running number of the configuration (counter): `1`, `2`, …
`{ZAEHLER:6}`	ditto, padded with zeros to 6 digits: `000001`
`{DATUM}`	today's date `20260911`
`{ZEIT}`	time of day `100507`
`{RECHNUNGSNUMMER}`, `{RG-NO}`	the number of the invoice – only for EDIFACT INVOIC (`BGM`) or ZUGFeRD/CII XML (`ExchangedDocument/ID`)

Example: `C:\EDI\out\INVOIC_{DATUM}_{ZAEHLER:6}.edi`. With `{ZAEHLER}` the output is always created – the number is only assigned after a successful conversion, and a number already in use is skipped. Without `{ZAEHLER}` an existing file is not overwritten.

The invoice number is read from the finished output, not from a field of the definition – no matter what the definition calls its fields. If the output is not an invoice, carries no number or contains several invoices with different numbers, the run ends with return value 7 and nothing is written. Characters not allowed in a file name become `_`.

Log of a run

Every run for a known configuration writes a log file: the details of the run with return value and message, the texts from `WARNING` and `ERROR` and the **IF statistics**. For every `IF` statement it names section and line and counts how often the condition was true and how often false; those never reached appear with 0. If the conversion aborts, it contains what was counted up to then. The log is written in the language that is set.

Setting of the configuration	Meaning
Log folder	empty: `temp` in the base directory; a relative path starts from the base directory
Keep logs (days)	older `.LOG` files in the log folder are deleted during the run; 0 means forever

The name follows the output file. With `{ZAEHLER}` the log is named like the written output, for example `INVOIC_000123.LOG`, and an existing one is overwritten. Otherwise it carries a timestamp, `INVOIC_20260913_101500.LOG`; if nothing was written with `{ZAEHLER}`, it is named after the input file.

Deleting only in the base directory

The converter only deletes files in the base directory, the common installation folder. Old logs outside stay where they are, and the run reports this. The only exception is the archive, which usually lies next to the input file. Files there are only deleted if "Also delete archive files outside the base directory" is ticked and confirmed in the configuration.

Granting and revoking the permission and every file deleted outside are recorded with time and Windows user in the deletion log of the database (table `Loeschung`). Only the half-finished `.tmp` file of a failed write is removed by the run anywhere, because it created it itself.

Concurrent runs

Every call is a process of its own; different configurations run side by side. If a configuration is started that is currently running, or another one with the same input file, the second run waits in the queue until the first is finished, at most 10 minutes. After that it is rejected with return value 14 without converting anything. This also applies between the Task Scheduler and the editor.

3.8 Complete example

The editor creates this definition for a new converter.

```
::GLOBAL-CONF
:  TARGET-DECIMAL-MARK = "."
:  TEMPLATE ~~SEG-END FIELD.SEG-ENDE.X1.TRIM = X"27"
:  TEMPLATE ~~FILL-10 FIELD.FILL.X10.TRIM

::TARGET-CONF
: RECORD.SEG-BGM
:    ~~FILL-10                          = "BGM+380+"
:    FIELD.BGM-NR.X35.TRIM
:    ~~SEG-END

::SOURCE-CONF
: RECORD-DELIMITER = X"0D0A"
: FIELD-DELIMITER  = ";"
: NO-RECORD-ID
: RECORD.SOURCE-RECORD
:    FIELD.SOURCE-DATA.X50

::RUN-CONF
: IF SOURCE-DATA = "BGM"
:    BGM-NR                             = GET_VALUE (2)
:    WRITE SEG-BGM
: END-IF
```

The input line `BGM;INV2026-12345` results in `BGM+380+INV2026-12345'`.