

Recovering and replacing legacy financial file converters

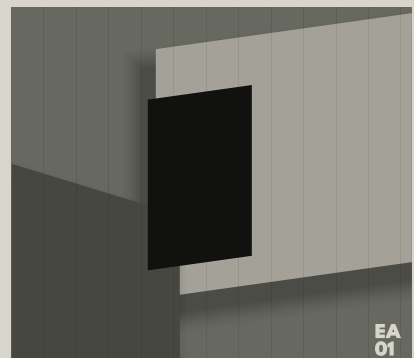
Reverse engineering MapForce executables and rebuilding Edlevo and SpeedAdmin conversion as a .NET 8 Frends integration

A contract-led method for preserving financial file behavior while replacing its execution model.

EDITION 1.0
19 OCTOBER 2026

CASE STUDY: 17 OCTOBER 2026
PACKAGE: 0.2.0 / BENCHMARK: 0.1.0

Ellie Augustini
Höglandsförbundet



Executive brief

Recover the contract before replacing the implementation.

This whitepaper documents the recovery of two undocumented financial conversion contracts and their replacement with an in-process .NET 8 library. It retains the full technical argument of the source case study and adds repository-verified details on row emission, evidence coverage, failure behavior and delivery ownership.

The engineering outcome is a reviewable implementation that reproduced 38 captured original outputs byte for byte in the recorded 0.2.0 replay. The historical synthetic unit run reports 58 passing cases. Neither result establishes independent financial correctness or production acceptance.

The earlier 0.1.0 benchmark measured eight supplied Edlevo files. For the largest case, one fresh original process took 2,778.8407 seconds; the replacement's warm median was 81.6151 milliseconds. These are different operational execution boundaries, stated alongside the results in Chapter 07.

Three conclusions govern use of the result:

1. Preserve selection, ordering, missing-value and byte-layout semantics before changing business policy.
2. Keep output agreement separate from finance approval, receiver acceptance and transport success.
3. Retain provenance and safe synthetic cases so future changes can be reviewed without reconstructing the lost visual project.

Audience. Principal engineers, integration architects, technical maintainers and financial-process owners evaluating or operating a legacy converter replacement.

Evidence status. Engineering evidence is recorded through 17 October 2026. This publication review inspected the repository and aggregate reports on 19 October 2026; it did not rerun the original executables, replay private invoices, execute a new benchmark, or establish deployment acceptance.

Contents

Executive brief	2
Abstract	4
01 Scope and evidence	5
02 Recovering the Edlevo contract	8
03 The SpeedAdmin parsing pipeline	17
04 The target byte contract	29
05 Configuration and architecture	32
06 Verification and assurance	40
07 Performance and its explanation	47
08 Data custody and reproducibility	51
09 Migration and operation	54
10 A reusable engineering method	57
Appendix A Output field dictionary	61
Appendix B Evidence and acceptance	65
Appendix C Sources and publication record	68

Abstract

Legacy file converters often survive long after the tools, projects and people that created them have disappeared. Their continued operation can conceal a substantial knowledge deficit. An executable accepts a file and produces another file; the organization may no longer know which fields matter, how invoices are grouped, which configuration values are active, or why execution takes minutes.

This project began with three owned MapForce executables, decompiled generated code, reference workbooks, a runtime parameter file, Friends integration code and a small collection of production examples. The original visual mapping projects were unavailable. The immediate requirement was understanding: recover the input and output contracts and explain the transformations. A subsequent implementation replaced the current Edlevo and SpeedAdmin executables with named, in-memory conversion code targeting .NET 8. The previous Edlevo executable remained useful historical evidence but was excluded from replacement scope.

The result demonstrates why this work requires several kinds of engineering at once. Recovering fixed columns is necessary, but collection semantics, grouping, text generation, missing values, numeric representation, configuration and execution environment all contribute to observable behavior. A rewrite can produce plausible invoices while differing materially from the old converter. Conversely, exact agreement with the old converter establishes compatibility for tested cases without independently establishing that the old financial rules are correct.

The current replacement passed 58 synthetic unit-test cases and reproduced 38 captured original-executable outputs byte for byte. A controlled benchmark of the earlier implementation compared eight Edlevo inputs with fresh original executions. On the largest supplied input, the original took approximately 46 minutes and 19 seconds; the replacement's warm median was approximately 81.6 milliseconds. These are measurements of two integration execution models with different boundaries, rather than an isolated comparison of parser instructions.

The transferable method is to recover a contract from several independent sources of evidence, preserve a reproducible executable oracle, make hidden semantics explicit, and separate compatibility work from business-rule changes. The same method applies to payroll exports, bank files, EDI adaptations, billing interfaces, administrative batch formats and other systems where a few hundred bytes carry disproportionate operational consequences.

01 Scope and evidence

Define the supported behavior, the surviving evidence and the boundary of the replacement.

SOURCE BASIS [01, 03, 06, 09, 10]

01.1 The actual problem being solved

The converters transform source data into fixed-width Raindance customer, invoice, detail and accounting records. Edlevo uses a fixed-position format with numeric record markers. SpeedAdmin uses a semicolon-delimited record format. Both eventually write the same physical target record families.

The business vocabulary needs care. The project is called IKE to Raindance, and IKE means interkommunal ersättning, or intermunicipal compensation. The implementation also contains childcare paths and SpeedAdmin culture-school fee behavior. Naming the entire input universe “IKE” would hide distinctions that affect references, wording and output naming. A useful first deliverable was therefore an inventory of families and versions, rather than a single universal format description.

The missing visual MapForce projects were a serious maintainability problem, but they did not leave a completely opaque system. The organization still possessed executable behavior, generated parser definitions, generated mapping code, an integration wrapper, configuration and examples. Each preserved a different part of the original design.

Table 1. Surviving evidence and its limits

Surviving artifact	What it can establish	What it cannot establish alone
Original executable	Observable behavior for a particular invocation	Business intent or complete behavior over all possible inputs
Decompiled parser and serializer definitions	Field order, widths, structural splits, encoding choices and output layout	Whether a declared field actually contributes to output
Decompiled mappings	Selectors, constants, predicates, grouping and transformations	The author's rationale or correctness of the business policy
Frends wrapper and surrounding integration code	Invocation, supplied settings, transport boundaries and cleanup behavior	An independently documented source-system or Raindance contract

Table 1. Surviving evidence and its limits (continued)

Surviving artifact	What it can establish	What it cannot establish alone
Reference workbooks	The supplied configuration values and their representation	A complete account-mapping model merely because the workbook is financial
Production examples	Real combinations of records and practical variation	Coverage of branches that never occur in those examples
Existing logs and intermediate files	Clues about stages and actual historical shapes	A reliable baseline when executable, settings and input provenance are unknown

The engineering objective was to replace an undocumented implementation while retaining the observable contract needed by its callers and receiver. That objective has a smaller, more testable scope than redesigning the entire billing process.

01.2 Establish the boundary before reading thousands of generated methods

The first useful question is what a conversion invocation actually includes. A file converter rarely depends only on the visible source file.

For this project, the effective inputs included source bytes, converter variant, company, mapping options, reference values and the execution environment. The local clock mattered as well. Output consisted of both the file body and a suggested delivery filename. The legacy process also modified its parameter file and produced temporary files and diagnostics.

A useful conceptual model is:

```
(body, suggested name, diagnostics, side effects)
  = converter(source bytes, configuration, implementation, environment, clock)
```

This is a model of the observable system, not the replacement's public method signature. It prevents a common error: comparing two bodies generated under different effective settings and attributing every difference to parsing.

The original Friends task already accepted source bytes and returned output bytes. Internally, however, it wrote a temporary input file, built an INI-style parameter file, started an executable through a Windows shell, read the result back and attempted cleanup. The surrounding workflow was responsible for transporting and delivering those bytes.

That distinction made the later architecture practical. File transport and executable orchestration were implementation machinery inside a

byte-oriented boundary. The replacement could retain the useful external shape while removing the process and intermediate-file requirements.

Before implementing anything, a similar project should resolve four boundaries:

1. **Compatibility boundary.** Which executable version and supported input families must be reproduced?
2. **Responsibility boundary.** Does the converter merely transform records, or does it also validate, number, route, deliver and acknowledge invoices?
3. **Evidence boundary.** Which examples are trustworthy baselines, and which are historical clues with incomplete provenance?
4. **Data boundary.** Where may production inputs, derived outputs, logs and diagnostic artifacts exist?

These decisions shape the implementation and test harness. They are difficult to retrofit after hundreds of comparison outputs have accumulated.

01.3 Preserve evidence and distinguish versions

A filename is a poor software identity. Assembly versions, mapping versions, framework targets and executable suffixes are separate identifiers. Current and previous Edlevo share an output layout but implement different transformations. The replacement explicitly selects current Edlevo and the supplied SpeedAdmin executable; the previous variant remains historical evidence outside the requested deployment scope.

Retain a baseline manifest connecting input, executable and output fingerprints with company, effective options and reference configuration. Record the harness version and timing boundary, and preserve raw original output in an authorized location. Where clock, timezone or culture can matter, record their values or execution policy. This is a recommended complete baseline design; the project's manifests do not claim every environmental input.

The replay harness validates input, executable and output hashes, options, company and the recorded reference-workbook fingerprint before reusing a capture. Verified reuse is valuable when a single original execution takes most of an hour; an old “passed” flag alone is insufficient.

Historical intermediate files are clues until controlled execution establishes their provenance. A familiar shape can belong to another version or option set. The same discipline applies to documentation: static analysis may be corrected by executable evidence. A field selection becomes a verified rule when source tracing and actual execution agree. Early investigation notes must remain distinguishable from current release evidence.

02 Recovering the Edlevo contract

Trace output dependencies through physical fields, collection operations and file-wide state.

SOURCE BASIS [02, 06, 16, 17, 18]

02.1 Read generated code as a graph of dependencies

Decompiled generated code is noisy for structural reasons. It contains a reusable runtime, generated enumerators, type descriptions, parser machinery and mapping-specific selectors. Reading every file in order gives the generic runtime the same attention as the actual business rules.

The efficient entry points in this project were the console entry point, parameter stage, input document definitions, preparation mappings, final mapping and output document definitions. Only then was it necessary to inspect runtime helpers that determined details such as trimming, first-value selection or numeric formatting.

Working backwards from an output field is often more productive than starting at the first input column. For example:

```
target R amount
  <- selected numeric value and separate sign predicate
  <- retained accounting data in the reduced representation
  <- source 63 amount
```

That dependency chain resolves an ambiguity that a field dictionary alone cannot. The source 62 record contains an obvious amount, quantity and unit price, but the current financial mapping uses the associated 63 amount. Copying the most visibly named amount field would have produced an elegant, incorrect replacement.

The same method applies to addresses, customer identities, VAT codes and references. Ask which source value reaches the serializer, under which predicate, and with which collection operation. Search for the destination field, follow its inputs, then inspect the helper semantics used along that path.

Generated collection operations deserve particular attention. “First present value,” “last selected value,” “any value equals zero,” “any value is negative” and “one result per grouped key” are different operations. They can become hidden business rules when a mapping tool expresses them through nested sequences.

This approach also avoids treating decompiled variable names as authority. A field called `levnr` is a useful trace label; it does not establish every business

meaning of that identity across all file families. Likewise, a source field named invoice date does not establish that the converter uses it.

The investigation used the decompiled sources supplied with the project; it did not recover the original visual design. Text search and source tracing established dependency paths. The .NET parity console and optional PowerShell oracle helper made those paths testable against executables. xUnit supplied repeatable synthetic cases. Stopwatch measured elapsed duration, SHA-256 connected artifacts to manifests, and archive inspection checked what packaging would distribute. Workflow linting and shell syntax checks addressed release machinery rather than conversion semantics. Each tool answered a specific question; none supplied an independent financial specification.

02.2 Recover the physical format before assigning meaning

An undocumented text file should first be treated as bytes with a grammar. Determine encoding, record delimiters, marker positions, declared widths, optional trailing fields and how malformed or short records are handled.

The recovered Edlevo input uses Latin-1 as its fallback encoding, with BOM detection inherited from its stream reader. In ordinary Latin-1 files, one character occupies one byte, so character positions and byte positions coincide. In a BOM-selected Unicode file, that equivalence no longer holds.

The replacement retains BOM-aware input decoding and Latin-1 output. It does not guess that a file without a BOM is UTF-8. That matters for Swedish characters: reading Latin-1 bytes as UTF-8 and then re-encoding the result can damage content before any mapping rule executes. Microsoft documents `Encoding.Latin1` as the ISO-8859-1 encoding. [Microsoft Encoding.Latin1 reference](#)

For fixed-width documentation, declare the position convention explicitly. The field dictionaries in this project use one-based inclusive positions, restarting on each physical line and excluding its line terminator. In C#, the corresponding substring starts one character earlier.

```
documented position p, width w
-> zero-based start p - 1
-> half-open interval [p - 1, p - 1 + w)
```

This translation belongs in one well-understood field-slicing mechanism. Repeating ad hoc offsets throughout mapping code makes review difficult and encourages off-by-one errors.

Declared record length also needs a precise interpretation. The old parser often sliced up to the available length rather than enforcing an exact length, and ignored content after the final declared field. A permissive parser is not evidence that truncated records are valid business input. It is evidence about what the implementation accepts.

Two useful artifacts emerge from this work. The physical schema records field positions, widths and parse behavior. A separate dependency table records whether each field is copied, transformed, used only as a condition or discarded. Keeping both prevents the source format's capacity from being mistaken for the converter's behavior.

The first five supplied Edlevo files differ from the generated input declarations at several physical boundaries. The parser accepts short trailing fields and ignores surplus content beyond its declared fields. Consequently, validating only exact declared line lengths would reject historical inputs already accepted by the executable. The observed lengths below describe this corpus; they do not replace the recovered parser schema. All lengths exclude line terminators. [06]

Table 2. Observed and declared Edlevo source lengths

Source record	Observed characters	Declared characters
10	63	280
60	350	332
61	250	250
62	256	296
63	300	303
64	300	250
50	240	240

02.3 Decoding the long zero padded fields

The most visually intimidating Edlevo fragment was a long sequence of zero-padded digits immediately after the 62 marker. It looked like a single enormous value. It was a set of adjacent fixed-width fields with no separators.

The relevant prefix is:

Table 3. Edlevo 62 prefix: recovered field boundaries

One-based positions	Width	Field	Recovered role
1-2	2	posttyp	Record marker
3-12	10	priskod	Price code
13-27	15	belopp	Signed item amount
28-42	15	antal	Encoded quantity
43-54	12	apris	Signed unit price
55	1	moms	VAT indicator
56	1	palagg	Surcharge indicator
57-64	8	Period start	Date-like source text
65-72	8	Period end	Date-like source text

TABLE NOTE

Amount and unit price have independent signs. Quantity has its own width, and the following VAT digit can look like the end of unit price when the field boundaries are invisible.

This deliberately synthetic example separates the fields:

Table 4. Synthetic Edlevo field interpretation

Field	Source value	Interpretation
amount	+00000000012000	12000 encoded monetary units
quantity	000000000000300	3.00 units
unit price	+000000004000	4000 encoded monetary units
VAT	0	

The aggregate examination of 5,272 source 62 rows established the relationship:

$$\text{encoded amount} = (\text{encoded quantity} / 100) * \text{encoded unit price}$$

The relationship also held for the 75 zero rows in that analysis. It supports interpreting quantity as hundredths of a unit.

It does not independently establish the currency scale. If amount and unit price are both rescaled by the same factor, the equation remains true. In the synthetic example, $120.00 = 3 \times 40.00$ would be consistent with a cents convention, but deciding that those are the actual business units requires an independent source-system or receiver contract.

This is a useful general distinction. A width establishes representation. An arithmetic relationship establishes relative scaling. A currency definition establishes business units. Those are three different claims.

An equally important discovery was that the current converter discards the 62 amount, quantity, unit price and VAT indicator before final financial mapping. It uses the associated source 63 amount at positions 9-23. It does not reconstruct the amount by multiplying quantity and unit price.

Consequently, deciphering the dense prefix answered a human understanding problem without changing the compatibility implementation's financial source. Reverse engineering often produces exactly this outcome: a field becomes understandable and is then shown not to drive the observed result.

02.4 Signed amounts and financial meaning

For ordinary invoice lines, positive and negative values support a useful operational shorthand of charge and credit. The recovered target convention is more precise: serialize the absolute amount and put a minus sign in a separate credit-marker field when the controlling value is negative.

Table 5. Amount magnitude and sign fields

Target record	Magnitude positions	Sign position	Negative representation
R	63-77	78	Absolute numeric text followed by separate -
K	125-139	140	Absolute numeric text followed by separate -

A blank sign field represents the nonnegative branch. A blank amount is different from numeric zero: text-only R records have an absent amount.

Calling every leading plus a debit and every leading minus a credit can become misleading when extended to complete accounting entries or entire invoices. A signed detail, a credit note, an account's debit/credit role and the receiver's posting conventions are related concepts, but the prefix alone does not establish all of them.

The replacement uses decimal arithmetic for ordinary numeric values and invariant parsing and formatting. It removes unnecessary fractional zeroes rather than imposing two decimal places. Thus 123.4500 becomes 123.45 and 12500 remains 12500. It does not apply a general division by 100.

Using `decimal` avoids introducing binary floating-point approximation into ordinary decimal financial values. Microsoft specifically identifies `decimal` as suitable where decimal digits determine required precision and notes that values such as 0.1 are exactly representable there. That is a numeric representation choice, not a substitute for recovering the legacy rounding and formatting rules. The SpeedAdmin compatibility path even retains a special exponent-notation route through `double` before decimal conversion because that is how the recovered conversion behaves. [Microsoft C# numeric type reference](#)

Several financial checks that might seem natural are absent from the original mapping. It does not generally recompute prices, calculate VAT from rates or reconcile source trailer totals. SpeedAdmin's output accounting amount comes from its detail R rather than an input K amount. Recomputing those values during replacement would change the contract.

The target R and K amounts must also be reconciled independently. They represent invoice detail and its accounting representation. Adding both sets of amounts produces double counting.

For Edlevo, repeated associated 63 records reveal a particularly subtle collection contract. The first present amount provides the magnitude, an existential negative predicate can supply the minus sign, and any zero accounting amount can exclude the entire detail candidate. These operations are not equivalent to summing splits. They warrant explicit tests and later business review rather than a well-intentioned replacement with conventional arithmetic.

Financial-detail and accounting emission are independent

A retained Edlevo detail does not necessarily produce one financial R and one K. The converter first excludes a detail if any associated accounting amount is zero. For the remaining details, it evaluates separate emission conditions: financial R requires detail text, while K requires a present first accounting dimension. The company 300 FAKTKV path explicitly requires financial wording and fails if its source text is missing. Customer and invoice headers have already been emitted before detail filtering. [17]

Table 6. Independent financial-row emission

Condition	Financial R	Accounting K
Edlevo: any associated amount is zero	Omitted	Omitted
Ordinary Edlevo: text and dimension 1 present	Emitted	Emitted
Ordinary Edlevo: text absent; dimension 1 present	Omitted	Emitted
Ordinary Edlevo: text present; dimension 1 absent	Emitted	Omitted
SpeedAdmin: selected charge, including one without source K	Emitted	Emitted

For Edlevo, each emitted amount uses the first present associated amount, and the minus marker reflects whether any associated amount is negative. A missing amount remains blank; it is not numeric zero. SpeedAdmin uses its charge R to supply both output amounts. Reconciliation must therefore examine presence, amounts and dimensions independently, rather than impose a universal R-to-K count relationship. [17, 19]

02.5 Reconstruct the hierarchy and the state it carries

The intended Edlevo shape is straightforward when written out:

Table 7. Edlevo expected containment

Depth	Record	Expected structure
0	10	file header
1	60	customer block
1	61	supplementary header
2	62	item description
2	63	associated accounting detail
2		... repeated detail groups ...
1	64	extended customer and address
1		... repeated customer blocks ...
0	50	file trailer

Declared lengths include 280 characters for 10, 332 for 60, 250 for 61, 296 for 62 and 303 for 63, excluding terminators.

The generated parser is less strict than the diagram suggests. It assigns the first and last lines to header and trailer positions, splits body blocks at lines beginning with 60, splits detail sections at 62 and treats the final line of a customer block as its extended-address record. It does not independently validate every marker against a formal state machine.

A strict new grammar would improve some error messages, but it could reject files the original accepts. The replacement retained the recovered structural behavior rather than using the rewrite to redefine the accepted source language.

The source file's physical blocks are not the final invoice groups either. Current Edlevo groups by a retained supplier identity, falling back to a recipient identity when the supplier value is absent. A repeated key can merge nonadjacent blocks. First occurrence determines group order, and source order is retained within the group.

The retained grouping strings use exact ordinal comparison. Ordinary trailing spaces are removed during field extraction, but leading spaces are not automatically removed from these keys. Two visually similar identities can therefore be distinct. A block missing both candidate identities is excluded from grouping and output. That exclusion is a substantive completeness consideration, rather than merely an implementation detail about dictionary keys.

This has consequences beyond data structures. A source invoice number, source date or original block boundary is not automatically an output invoice boundary. First and last selections can take address and identity information from different blocks within a merged group. The mapping contains no universal agreement check requiring all those values to match.

Grouping also explains why a simple read-one-block/write-one-invoice stream is insufficient. A later block may belong to an already encountered key. Exact behavior requires retaining groups, deferring output or using an indexed spool that preserves first-appearance group order and source order within each group. Naively sorting by identity changes the contract. For the observed file sizes, an in-memory representation was the simpler compatibility choice.

Other state spans the complete conversion. Consecutive person headings can be suppressed across customer boundaries. That state is keyed by transformed display text rather than a unique person identifier. Resetting it at every invoice would appear cleaner, yet could add rows that the old output omits.

File-family flags have similar scope. They are derived from the filename embedded in the 10 record, not from the external filesystem name. Some flags are independent substring predicates: a name satisfying FAKTKV also satisfies a broader FAKT test. Replacing these with a mutually exclusive enum without tracing precedence would be a semantic change.

Validation precedes some exclusions

Edlevo constructs its source model before excluding customers without grouping identities or suppressing details associated with zero amounts. During construction, it parses every associated accounting amount and validates the selected counterparty against the customer-type table. Invalid values can therefore fail the call even when the corresponding customer or detail would later be excluded. [18, 17]

This ordering is part of the implemented behavior. A redesign that filters first and validates only retained records would change the accepted input domain, even if successful outputs appeared unchanged. Characterization should cover both returned bytes and failure behavior. This observation is established by source inspection; it does not claim exhaustive original-executable coverage of malformed inputs.

03 The SpeedAdmin parsing pipeline

Characterize the composed language: preprocessing can change ownership and financial meaning before CSV field parsing begins.

SOURCE BASIS [04, 06, 19]

Edlevo makes its difficulty visible immediately: rigid columns, numeric record markers and adjacent zero-padded fields that resemble one enormous number. SpeedAdmin supplies semicolons, quotes and readable rows. That familiarity encourages assumptions which its conversion pipeline does not support.

The maintenance difficulty comes from a sequence of parsing stages that disagree about what constitutes a field, a record and protected text. Some inspect raw physical lines, some split on record prefixes, and another finally interprets semicolon fields and quotes. A field can survive one interpretation and already have damaged another.

The semicolon delimiter itself is unremarkable. Hierarchical record exports are also legitimate formats. The difficult contract belongs to this supplied export and its legacy consumer together, including generated preprocessing and parser configuration. These findings should not be generalized to every SpeedAdmin export or every semicolon-delimited file.

Compare the two sources before sharing parser logic

Both converters ultimately produce the same Raindance layout. Their input languages and selection rules remain substantially different.

Table 8. Source contracts compared

Concern	Current Edlevo	Supplied SpeedAdmin contract
Field boundaries	Fixed character slices	Semicolon fields, after raw-text preprocessing
Record hierarchy	Numeric 10, 60, 61, 62, 63, 64 and 50 records	Letter O, S, H, R, K and C records
Customer aggregation	Repeated retained identities can merge nonadjacent source blocks	Containment beneath each S; no identity-based join across S blocks
Invoice boundary	Follows the recovered grouping identity	Multiple source H blocks beneath S flatten into one output H

Table 8. Source contracts compared (continued)

Concern	Current Edlevo	Supplied SpeedAdmin contract
Controlling financial value	Associated source 63 amount	Charge R column 3 with its column 4 credit marker
Misleading financial fields	Source 62 amount, quantity, unit price and VAT digit do not drive the current financial mapping	Input K amount/sign, R quantity/unit price and H debit flag do not supply the corresponding output values
Accounting multiplicity	Repeated 63 records participate in selection, sign predicates and zero filtering	Several K records collapse to one K with independently selected dimensions
Blank-field behavior	Whitespace-only fixed fields generally become absent	Empty CSV fields become absent; whitespace-only fields remain present
Reference configuration	VarRef and family/subtype rules	No reference workbook or VarRef lookup
Text hazards	Offsets, truncation, normalization and global heading state	Those semantic issues plus raw separators, quotes, hash truncation and prefix boundaries
Shared result	Latin-1 fixed-width S/H/R/K records	The same target layout, with different mapping semantics

Edlevo's difficulty is strongly positional. Once the correct slices and dependency paths are known, much of its source representation becomes mechanical. SpeedAdmin adds uncertainty about when a row acquires its meaning. An ordinary-looking empty field or terminal separator can influence the grammar before the actual field parser runs.

Hierarchy inside a delimited format

This is not one rectangular table with a shared schema. The first field selects different layouts: six columns for O, sixteen for S, ten for H, nine for R, eighteen declared for K and four for C. There is no column-header row.

The intended shape is approximately:

Table 9. SpeedAdmin expected containment

Depth	Record	Expected structure
0	O	opening envelope
1	S	customer
2	H	source invoice boundary
3	R	heading
3	R	charge
4	K	associated accounting
4	K	optional further accounting
3	R	another charge
4	K	associated accounting
2	H	optional further source invoice boundary
3		...
1	S	next customer
2		...
0	C	closing envelope

This diagram explains expected containment, rather than describing a strict validating grammar. A K belongs to the preceding charge's structural range. The converter does not join it to an R using an invoice number or line identity. Likewise, an H belongs beneath its surrounding S.

The distinction matters when inspecting files in a spreadsheet or generic CSV viewer. A row is not independently meaningful just because its columns can be displayed. Moving a row can change its owner. Sorting the apparent table can attach accounting to another charge or move charges beneath another customer.

There is another trap in the letters themselves: SpeedAdmin source R and K are semicolon-delimited source records. Raindance output R and K are different fixed-width records. Sharing a marker does not imply a direct copy or preservation of every source row.

The order of interpretation

The effective pipeline is closer to the following than to a single CSV read:

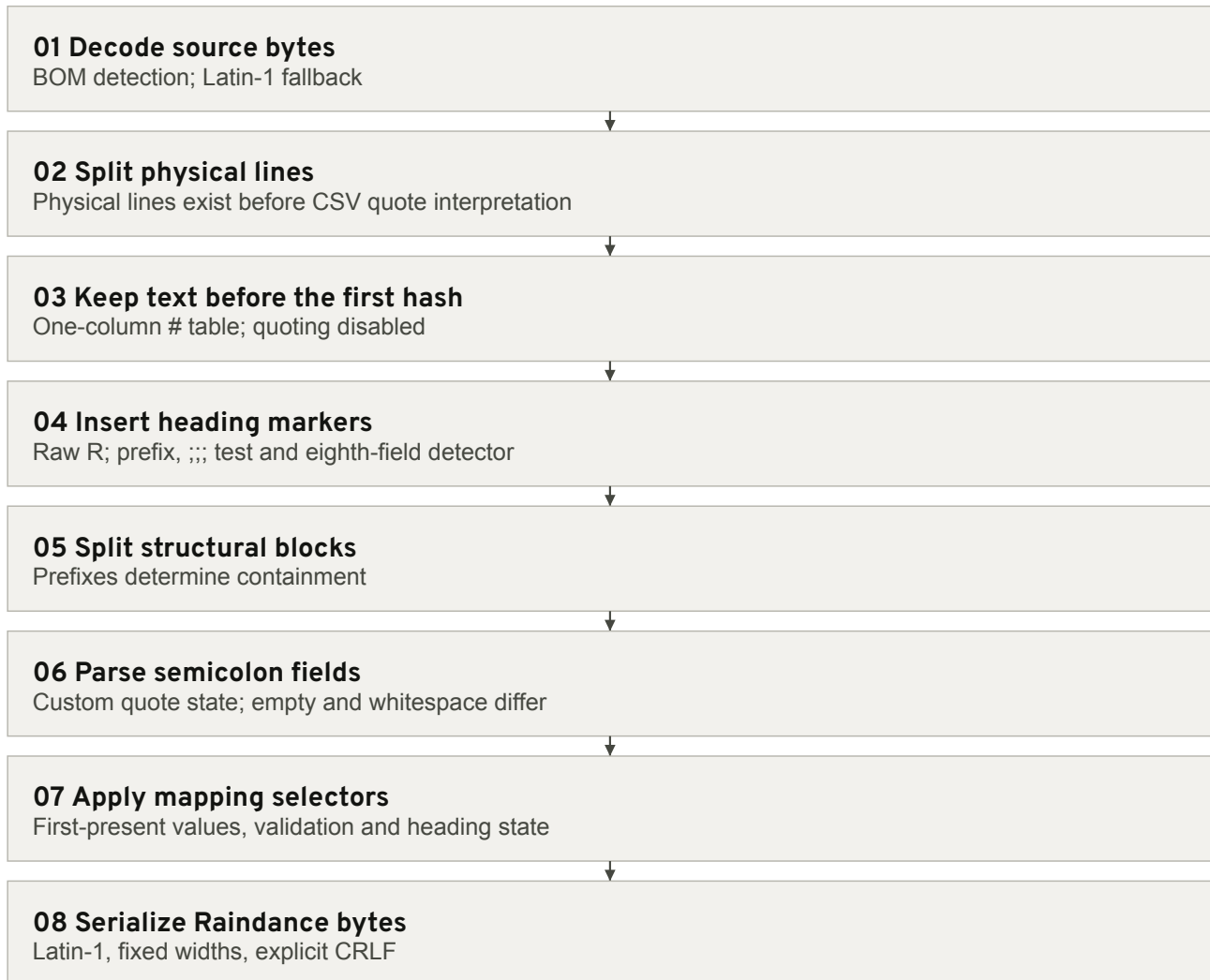


Figure 1. SpeedAdmin interpretation order. This conceptual sequence exposes the stages that act before quote-aware field parsing. [04, 19]

The ordering is the problem. Quote-aware field parsing occurs after preprocessing and structural splitting. It cannot recover text already discarded or restore a boundary already assigned to the wrong block.

A useful investigation therefore asks separately what physical-line preprocessing sees, what structural splitting sees and what the field parser sees. Testing only the final CSV reader misses failures introduced earlier.

A unit price can be grammar while supplying no price

The preprocessor inserts a synthetic `line_split;` line before an R when all three conditions hold:

1. The raw line starts with the exact two characters `R;`.
2. The raw line contains `;;;` somewhere.
3. Its raw eighth-field detector reports an empty result.

The inserted marker starts a heading block. The first R in that block becomes heading text; subsequent Rs become charge records. An R selected as a heading does not supply its amount or VAT to a financial detail.

The eighth source field is named unit price. It does not populate output unit price, multiply the quantity or calculate the output amount. Nevertheless, its apparent emptiness helps decide whether the row is a heading at all.

This is why a transformation inventory needs a category for “predicate only” or “structural.” Calling an output-disconnected field “unused” can be dangerously incomplete.

The detector is not a quote-aware request for CSV column eight. It performs seven raw substring-after-semicolon operations, then takes text before the next semicolon. Missing separators yield an empty result. The literal triple-semicolon condition also examines raw text, rather than a parsed run of empty fields.

Consequently, classification can be wrong even when a later field parser would have understood the data perfectly.

The trailing semicolon that changes what an invoice row means

Consider these synthetic fragments:

```
R;Tuition;125.50;;MF;;;77;  
R;Tuition;125.50;;MF;;;77
```

The difference is one final semicolon. A field reader supplied with nine output slots obtains the same effective values from both: amount 125.50, VAT MF, eighth field 77 and an empty or absent ninth field.

The raw heading detector behaves differently:

Table 10. Terminal separator and heading classification

Fragment	Text after seven semicolons	Next semicolon	Heading marker inserted
With terminal semicolon	77;	Present after 77	No
Without terminal semicolon	77	Missing	Yes

TABLE NOTE

In the second case, “text before the next delimiter” evaluates as empty because the delimiter is missing. The detector therefore reports an empty eighth field despite the visible 77.

In an otherwise appropriate surrounding file, the inserted boundary can turn the row into a heading and cause its amount and VAT to be ignored in that role. A CSV cleanup tool that removes a seemingly redundant terminal separator can therefore change financial output.

The isolated preprocessing and field-parser behavior in this example was verified against the current compiled replacement using synthetic strings. It is not an additional fresh run of the slow original executable. It demonstrates the recovered rule and provides a focused candidate for extending original-executable characterization.

Quotes provide local protection

A quote-aware CSV parser normally distinguishes a separator inside quoted text from a separator between fields. This pipeline's raw classifier does not.

For example:

```
R;"Label;with;several;parts";100;;;;;2;
```

The semicolon field reader sees the quoted label as one field and sees 2 in column eight. The earlier raw detector counts the label's internal semicolons as delimiters, lands on an empty raw field and inserts a heading boundary.

The two parsers disagree about where column eight is. A correct quote implementation in the later stage does not make their composition correct.

There are three further quote hazards.

First, the initial one-column table uses # as its delimiter with quoting disabled. It retains only the text before the first hash. This is not a conventional CSV comment rule; it is a consequence of that earlier table definition.

```
Source:
R;"Lesson#2";125.50;;MF;;;77;

Text remaining after the first stage:
R;"Lesson
```

Hash truncation precedes quote interpretation. The closing quote and financial fields have already disappeared. Later stages cannot recover them, whether the eventual outcome is altered mapping or an error.

Second, the recovered semicolon parser toggles quote state on unescaped quotation marks, with NUL configured as its escape control. Doubled quotes toggle state twice and disappear:

Table 11. Quote interpretation

Interpretation	Value
Source field:	"Ensemble ""A"""
Ordinary CSV interpretation:	Ensemble "A"
Recovered field reader:	Ensemble A

RFC 4180 describes doubled quotes as an escaping convention and allows line breaks inside quoted fields. The relevant distinction here is the escape and record handling, rather than the choice of semicolons instead of commas. A general CSV library can implement those conventions correctly and still differ from this consumer. [RFC 4180](#)

Third, logical CSV records and physical lines are not interchangeable. The low-level reader can maintain quote state over joined text, but preprocessing has already split the original input into physical lines. Structural parsing then tests those lines for prefixes.

A quoted text value containing a newline followed by `s;` can therefore expose a new apparent customer boundary before any CSV quote state protects it. Ordinary top-level records are also passed to their field parser one physical line at a time. Multiline support in a helper is not evidence of end-to-end multiline support.

Quoting all fields is another plausible interoperability failure. A standard writer may treat `R` and `"R"` as equivalent first-field values. The heading preprocessor checks the raw prefix `R;`, and the structural splitters inspect raw initial letters. Quoting record markers changes the language those earlier stages see.

Prefixes stand in for a grammar

The structural splitters test prefixes such as S, H and R, rather than requiring a parsed first field equal to the expected marker. A physical line beginning with a longer word that starts with R can therefore create an R boundary in the relevant range.

The opening and closing envelopes are selected by first and final position, rather than validated O/C markers. A block splitter consumes its first physical line even when that line does not carry the requested prefix. Detail processing can discard a line as an internal marker even when malformed input did not actually supply one.

Edlevo also has positional envelope selection and permissive prefix-based structure. SpeedAdmin compounds that behavior with a preprocessing stage that manufactures boundaries from raw field content. A delimiter problem can become a hierarchy problem, which then becomes a missing or reassigned financial row.

This explains why successful field parsing and successful conversion are weak integrity signals by themselves. The C record contains customer counts, row counts and a batch sum, but the mapping does not reconcile them. Source O and C do not become target control records.

Empty and whitespace are different data

The SpeedAdmin field model omits zero-length values. A field containing spaces remains present. Missing trailing fields become absent slots, and fields beyond the declared count are ignored by the field reader.

That presence rule interacts with selection order. The dimension helper chooses the first present value and only then normalizes whitespace:

Table 12. First-present selection before whitespace normalization

Selection step	Value
First associated K responsibility:	" "
Later associated K responsibility:	"200"
Choose first present value:	" "
Normalize it:	" "
Output responsibility:	blank

Trimming fields before empty-value omission and first-present selection would instead choose 200. That apparently helpful cleanup would change accounting output.

Edlevo's fixed-field extraction generally treats whitespace-only values as absent and removes ordinary trailing spaces. A shared “clean text” helper across both source formats would therefore erase a real semantic difference.

The distinction also affects failure behavior. An empty amount can be absent; spaces are a present string and can fail numeric parsing. An empty counterparty is skipped; a spaces-only counterparty normalizes to empty and then fails the lookup.

Normalization is not uniformly applied even to successful values. A padded recognized counterparty can validate after normalization, while its raw form is copied into S/H and its accounting dimension is normalized. Broadly normalizing every output string would lose that distinction.

Blank physical lines have their own rule. Empty lines and lines consisting entirely of hashes are skipped by preliminary processing. Whitespace-only lines are not the same case. They may remain available to influence positional structure.

Invoice and accounting boundaries collapse

The apparent source hierarchy promises more detail than the target mapping preserves. Several H blocks under one S produce one target H. None of the source H business fields populates that target header: dates, references, debit flag and source description do not control their corresponding output values.

The target subject comes from configuration, and the target header's identity and counterparty information come from other source locations. Moving from a source model that contains invoice headers to a target model that preserves all of them would change compatibility.

Accounting collapse is more subtle than “take the first K.” Each selected dimension independently takes its first present value across the associated K collection.

Table 13. Independent first-present accounting selections

Associated source K	Account	Responsibility
First	3010	Absent
Second	4020	200
Resulting single output K	3010	200
TABLE NOTE The resulting dimension combination need not occur in any single source accounting record. It is neither a copy of the first K nor a sum of accounting splits.		

The positional mapping is also selective:

Table 14. Selected SpeedAdmin accounting columns

SpeedAdmin source column	Target use
K2	Accounting dimension 1
K3	Accounting dimension 2
K6	Accounting dimension 3
K7	Accounting dimension 4
K8	Accounting dimension 8 and counterparty-related selection
K14 amount and K15 sign	Ignored for output financial amount and sign
K16 text and K17/K18 period fields	Not copied into the corresponding output values

TABLE NOTE

A charge generates one output K even when there is no source K. Its dimensions may be blank, while amount, sign and text still come from the charge R.

For comparison, Edlevo's ordinary accounting dimensions begin at source 63 position 51. BETIK uses its counter-account range beginning at position 204. Each range holds eight ten-character dimensions, and BETIK does not automatically fall back to the ordinary range. Both converters need a deliberate mapping; neither supports an assumption that identically named account slots simply copy through.

The source R controls both financial amounts

SpeedAdmin's charge R supplies the amount for both target R and target K. Its source K amount and sign do not participate. Source R quantity and unit price do not calculate the charge, and the H debit flag does not determine its sign.

The signed text is built by concatenation:

```
signed amount text = R column 4 credit marker + R column 3 amount
```

Thus a marker - with amount 125.50 parses as negative. A marker - with an already negative amount produces --125.50 and fails. Treating two negatives as a positive or removing one would introduce a new policy.

Both output records use the resulting absolute magnitude and separate sign. MF becomes K00 after VAT uppercasing; other VAT strings pass through the recovered logic and remain subject to the target's three-character field width. There is no general VAT calculation.

Edlevo takes its financial value from accounting record 63 and has an existential zero-accounting filter. SpeedAdmin takes it from R and has no equivalent Edlevo zero-accounting suppression rule. A common invoice abstraction that hides where the controlling value came from would make those differences harder to review.

Evaluation can exceed what is eventually serialized

S/H retain the first present descendant K counterparty, but every present descendant counterparty is validated. A recognized first value followed by an unknown value fails conversion. Stopping as soon as the first usable value is found would preserve that output slot while changing failure behavior.

Different recognized counterparties are not rejected merely for differing. S/H can retain the first, while individual detail K records use their own selected dimensions. That is a recovered selection behavior, not proof of a consistent financial grouping policy.

The shared customer-type table is exact: 860y maps to UTL, whereas 860Y is unknown. A universal uppercase pass would change accepted input.

Heading suppression adds another scope dependency. Consecutive equal normalized headings are suppressed across customer and header boundaries within the conversion. A sequence A, B, A emits all three; this is not a global distinct-name set. Two adjacent customers sharing a heading can receive different visible heading treatment solely because of their order.

The source R count, target R count, number of charges and number of headings therefore describe different quantities. A generic “all rows imported” assertion is insufficient.

Why replacing the CSV library is not the repair

The practical failure modes form a chain:

Table 15. Preprocessing changes with financial consequences

Seemingly harmless operation	What this consumer may change
Remove a terminal separator	Charge-versus-heading classification
Put semicolons inside correctly quoted text	The raw eighth-field calculation
Put a hash inside quoted text	Truncation before quote parsing
Escape a literal quote by doubling it	Loss of the quote character
Quote every field, including record markers	Raw marker recognition

Table 15. Preprocessing changes with financial consequences (continued)

Seemingly harmless operation	What this consumer may change
Wrap a long quoted value over physical lines	Structural prefix boundaries
Trim all cells	Presence, first-value selection and error behavior
Sort spreadsheet rows	Customer, heading and accounting ownership
Preserve each H or K as a separate target object	Invoice count and accounting allocation

The engineering task is to characterize the composed pipeline, then choose which behaviors are compatibility requirements and which deserve a separately approved correction. A standards-compliant CSV parser can be part of a future redesign once the producer, consumer and financial expectations agree on one grammar.

For this replacement, the accepted legacy behaviors remain explicit and testable. The original-output corpus includes cases for multiple headers, repeated accounting and missing accounting, as well as ordinary financial paths. The isolated hash, quoting and trailing-separator demonstrations explain additional parser hazards; they should not be misrepresented as newly executed original-EXE comparisons.

The SpeedAdmin compatibility problem is therefore precise: familiar punctuation conceals several competing definitions of a record, followed by mappings that discard or combine apparently meaningful source structure. Edlevo demands careful byte positions. SpeedAdmin demands that care plus an account of which parser gets to interpret the text first.

Boundaries of the supplied-file evidence

The two supplied SpeedAdmin files exercise the ordinary PRIV/K00 path: every observed K counterparty is 860, every detail VAT code is MF, and no detail has a negative credit marker. Neither file contains quotes or #. Each observed K row has 17 fields, including its trailing empty field, against 18 declared fields. These files establish the common production shape; source tracing and constructed comparisons provide the separate evidence for credits, multiple or missing accounting records, multiple invoice headers, and parser edge semantics. Not every malformed or quoted-input combination has executable coverage. [04,06,10]

04 The target byte contract

Preserve the physical output and the text rules that make it meaningful to the receiving process.

SOURCE BASIS [02, 04, 16]

04.1 The target file is a byte contract

Both source families produce the same four fixed-width record kinds:

Table 16. Fixed-width target record lengths

Record	Purpose	Data bytes	Bytes including CRLF
S	Customer and address	305	307
H	Invoice header	209	211
R	Explanatory text or financial detail	105	107
K	Accounting dimensions and amount	184	186

Output is Latin-1 without a BOM, with explicit CRLF terminators. Those choices remain necessary on a Linux host. Using the platform's default newline would make a superficially harmless portability change visible in every line.

The shared writer centralizes widths, space filling and truncation. Ordinary fields are left aligned and padded on the right. Amounts look right aligned because the amount formatter adds leading spaces before the shared writer receives the value.

This distinction is useful when reproducing a generated serializer. “All numeric fields are right aligned” is an inaccurate schema claim if alignment is performed by mapping expressions rather than the serializer.

A financial target R can be viewed as:

Table 17. Financial R record: field positions

One-based positions	Field
1	marker
2	reserved space
3-62	row text, width 60

Table 17. Financial R record: field positions (continued)

One-based positions	Field
63-77	amount magnitude, width 15
78	credit marker
79-81	VAT code
82-89	quantity slot
90	quantity sign slot
91-105	unit price slot

The presence of quantity and unit-price slots does not mean this mapping populates them from similarly named source fields.

Likewise, the H schema contains dates, invoice number and several identifiers that the mappings can leave blank. A replacement should not start populating those fields simply because it now has a convenient source model. A blank may select behavior in the receiver that is outside the recovered implementation.

The R record is polymorphic in practice. Initial descriptions, child or person headings, period text, income text and amount-bearing details share its physical layout. Therefore an R count is not a charge count.

R text allows 60 characters, whereas K text allows 30. The accounting description can be truncated when the invoice description is still complete. Reserved fields also occupy real positions. Splitting these records on whitespace destroys the information needed to distinguish empty slots, padding and embedded spaces.

The serializer preserves legacy truncation rather than wrapping. That includes numeric text after it has been formatted. A future hardening decision should consider rejecting numeric width overflow explicitly; blindly truncating an overlong amount is not a sound financial policy merely because a generic fixed-width writer can do it. That change should have its own accepted-input decision and tests.

Latin-1 also imposes a character repertoire. The current writer uses normal encoding replacement fallback for unrepresentable text, rather than promising arbitrary Unicode fidelity or rejecting every unsupported character. Custom configuration text and new source-system encodings require deliberate validation.

The replacement does add a guard against CR or LF inside a scalar output field. Without such a guard, a field value could create extra physical records while satisfying an ordinary string API. That is an intentional boundary rule, distinct from copying every historical failure mode.

04.2 Small text rules can carry substantial business behavior

An invoice converter is partly a document generator. Its prose determines what the recipient understands about a charge, and the text logic can encode branch conditions that are financially relevant.

Several recovered Edlevo rules illustrate the range:

- The ordinary initial “Avser” explanation and BETIK’s period text have separate conditions.
- Person names containing a comma are reordered by string operations; this is not a general name-parsing algorithm.
- A company-specific FAKTKV branch produces school and term wording using the previous month. January requires the preceding year.
- Month names are Swedish constants rather than a host-locale date presentation.
- A service-text match, normally for `busskort`, chooses K06 instead of the default K00 VAT code. The current mapping does not simply copy the 62 VAT digit.
- Income handling detects a configured label case-insensitively and then extracts it case-sensitively. The inconsistency affects output.
- A protected-address predicate requires the relevant address and postal fields to be present before applying its marker tests.
- VAT registration text can be constructed by adding a prefix and suffix to an identity, without proving a valid registration.

Whitespace normalization is also specific. The replacement reproduces XML-style normalization for space, tab, CR and LF, rather than treating every Unicode whitespace character identically. The XPath definition of `normalize-space` specifies trimming and collapsing XML whitespace. A broad regex using `\s` or a different generic trimming rule can change non-breaking spaces and therefore field content. [W3C XPath string functions](#)

The general rule is to recover text transformations with the same care as numeric transformations. A regex that “cleans up” whitespace, a culture-dependent uppercase operation or a reset heading counter can alter receiver-visible behavior without changing a single total.

05 Configuration and architecture

Replace hidden execution machinery with explicit, reviewable responsibilities and a documented public boundary.

SOURCE BASIS [05, 07, 08, 09, 20]

05.1 Recover configuration meaning before choosing its syntax

The reference workbooks were initially described as account mappings. Inspection showed that their relevant populated values selected “our reference” codes, such as KR02 and K22 subtype codes. They were not a general translation table for ledger account dimensions.

This matters architecturally. Output K dimensions come from source fields and mapping rules. Moving the workbook's values to JSON should not introduce a new account-mapping abstraction that the old configuration never supplied.

All six supplied company workbooks had identical populated values. That establishes equivalence of those supplied configurations, not a permanent rule that all companies must share references. Company and reference configuration remain explicit, independent inputs.

The first replacement representation preserved a multiline K22 string. It was compatible but inconvenient: one JSON string embedded several separate codes and their parenthesized labels. Maintaining it required understanding a second format inside the first.

The 0.2.0 configuration expresses the actual data model:

```
{
  "K02": "KR02",
  "K22": {
    "VUX": "KR2201",
    "Gymnasiet": "KR2202",
    "Anpassad gymnasieskola": "KR2203",
    "Grundskolan inkl fritidshem": "KR2204",
    "Förskola": "KR2205"
  }
}
```

The decisive improvement is structure, rather than the choice between JSON and YAML. Each subtype now has a distinct key and a single-line value. A reviewer can see and change one mapping without editing an embedded mini-language.

JSON fit the Friends object boundary and .NET's existing serializer, avoiding an additional runtime configuration parser. The production converter receives a typed object. It does not open a JSON file, parse YAML or read a workbook. An external caller may deserialize stored JSON before invoking it.

The subtype lookup accepts known labels case-insensitively, rejects unknown labels and case-duplicate entries, rejects null values and CR/LF in references, and permits partial mappings. Missing or empty selected references remain blank. Values are normalized and uppercased, and the lookup is copied into call-local state without rewriting the caller's configuration.

These rules deliberately avoid inventing fallback behavior. In particular, KJ1's final reference remains blank even when K02 is configured, because the recovered final selector has no KJ1 subtype branch. BETIK's preschool reference choice depends on a file-wide condition: one matching preschool detail can affect the reference used by relevant headers elsewhere in the file.

The public shape changed between 0.1.0 and 0.2.0, but the resulting file bodies were replayed against the verified original captures. This is a useful separation: configuration usability can improve through an explicit API migration while invoice semantics remain under an unchanged output comparison.

Missing null and empty are different states

The surrounding text options preserve another legacy convention: absence, null, empty, whitespace and an explicit suppression sentinel can mean different things.

For settings that use default-value handling, the effective states include:

Table 18. Defaulted text-option semantics

Value received by the task	Defaulted Edlevo text option	Defaulted SpeedAdmin text option
Explicit null	Suppress the value	Select the default
Empty string	Select the default	Select the default
Exact literal {override_null}	Suppress the value	Suppress the value
Whitespace-only string	Supplied text; later normalization may make it blank	Supplied text; later normalization may make it blank
Other nonempty string	Use the supplied value	Use the supplied value

An omitted property during deserialization generally retains its model initializer, which is a separate issue from explicitly assigning null. Reference-object

properties have their own validation: explicit null references are rejected. These rules should not be generalized from one option family to every property.

A conventional rewrite such as `value ?? defaultValue` would change nullable Edlevo text behavior. Applying a shared “null or whitespace means missing” helper would change it again. The implementation keeps the distinction because it can alter invoice wording and even filename components.

For future APIs, a dedicated optional-value model could make these states more explicit. The compatibility task instead documents and tests the inherited convention so callers can migrate predictably.

05.2 Design the replacement around explicit responsibilities

The replacement is a library with two synchronous Friends task methods:

```
ConversionResult Main.ConvertEdlevo(  
    EdlevoInput input,  
    EdlevoOptions options,  
    CancellationToken cancellationToken = default);  
  
ConversionResult Main.ConvertSpeedAdmin(  
    SpeedAdminInput input,  
    SpeedAdminOptions options,  
    CancellationToken cancellationToken = default);
```

Both receive original bytes, perform conversion in memory and return output bytes, a suggested filename and record counts. Exceptions signal failures. The production assembly has no external package dependency, launches no executable, reads no workbook and creates no intermediate conversion files.

The separation is intentionally modest:

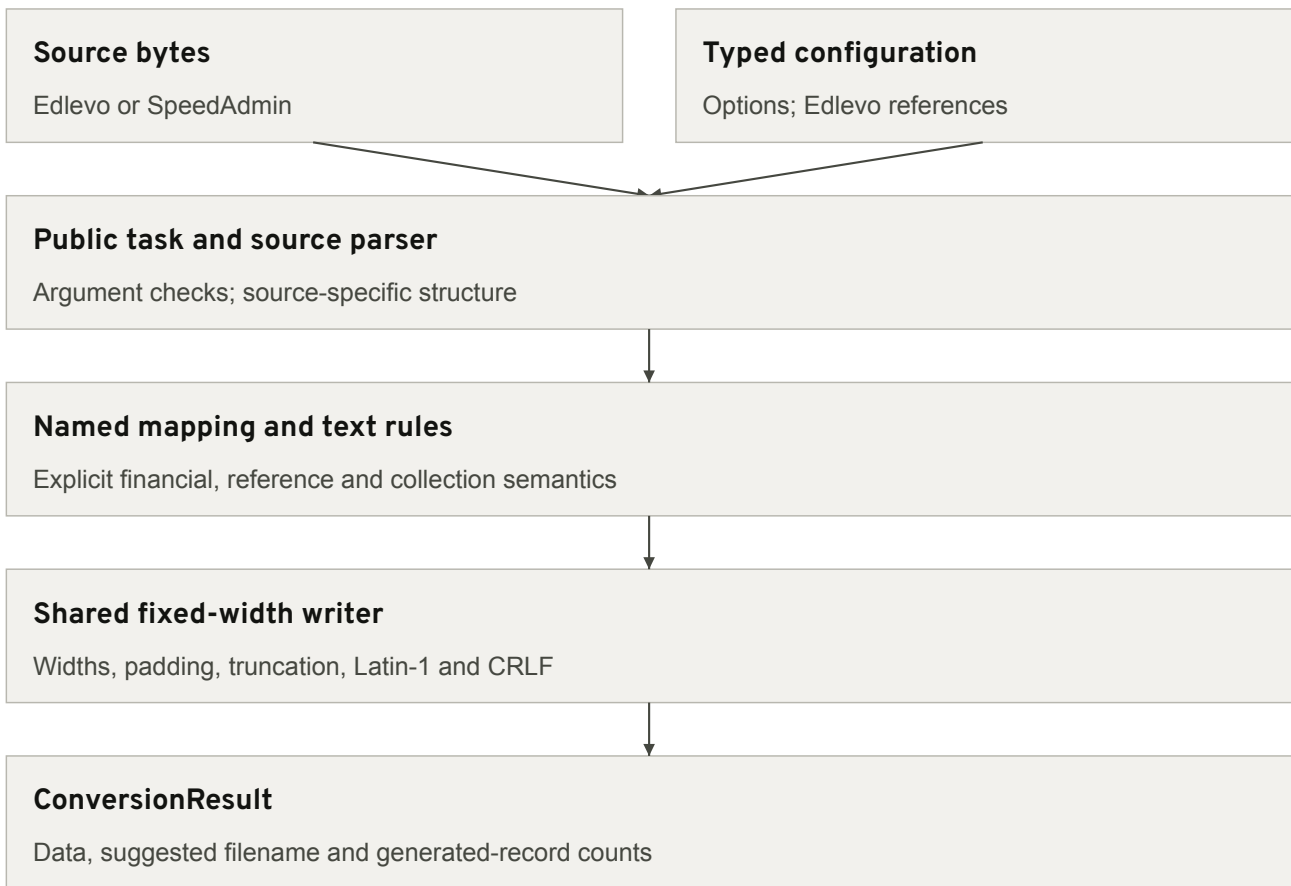


Figure 2. Replacement responsibility boundaries. Transport and receiver acceptance follow the returned result and are owned by the surrounding process. [07, 08, 20]

The parser recovers the source structure. Named mapping methods express the historical branch rules. A shared writer controls physical output. The public task coordinates those components and produces the delivery-name suggestion. Tests, executable-oracle tooling and workbook adaptation live in separate development projects.

This architecture makes it possible to discuss a rule without understanding a generated enumerator graph. A future maintainer can find reference selection, person headings, income text or amount formatting by responsibility.

It also avoids an unnecessary generic mapping framework. There are two source families and one specific target contract. A new abstraction should earn its place by simplifying real shared behavior. The shared writer and normalization helpers meet that test; a configurable universal transformation engine would reintroduce complexity that this replacement is intended to remove.

The methods are synchronous because the production conversion performs computation over already available data. There is no asynchronous filesystem

or network operation inside the task that would benefit from an async interface. Scheduling, concurrency and transport remain the integration host's responsibilities.

Call ownership and concurrency

The converter does not mutate the source byte array. Reference values are validated and copied into a call-local lookup. Heading suppression and other conversion state belong to the individual invocation rather than static counters shared by concurrent calls.

The test suite exercises simultaneous conversions, including a 32-call scenario. That supports state isolation, but does not make mutable arguments magically immutable. A caller should not alter an input array, options object or dictionary while a call is using it. A dictionary snapshot still has to enumerate the original dictionary safely.

Cancellation is cooperative. Checks occur before work, within conversion loops and during final result counting. Cancellation throws `OperationCanceledException`; it is not a promise of immediate interruption at every instruction. The legacy-oracle process runner has a separate obligation to stop its child process on timeout or cancellation.

Memory instead of intermediate files

The implementation holds decoded source text, source models, accumulated output text and the final output byte array in memory. It is not a streaming API.

For the supplied few-megabyte files, that choice produced simple, reviewable code and very low observed runtimes. Global grouping and file-wide predicates already require more than a single independent customer block. Avoiding temporary files also removes naming collisions, directory permissions, cleanup races and repeated serialization boundaries.

Memory use still needs a deployment policy. Several simultaneous files multiply the live working set, and decoded .NET strings occupy more memory than a Latin-1 source byte array. Output text and output bytes can coexist during encoding. No peak-memory benchmark or universal maximum input size has been established for this project.

A later streaming design should start with measured memory pressure and a model of the actual global dependencies. It may require bounded queues, indexed spooling or a preliminary pass. Replacing one large allocation with a more complicated, partially buffered implementation is not automatically an operational improvement.

The public call still depends on its environment

The new architecture is largely an in-memory mapping, but it is not a mathematically pure function of bytes and options.

Each call samples `DateTime.Now` once. That value produces the delivery timestamp and participates in Edlevo's expansion of six-character periods. The period rule uses the current century, with a special next-century case around a year ending in 99. The clock can therefore affect body content at a century boundary as well as filename metadata.

Numeric formatting is invariant. However, SpeedAdmin VAT uppercasing explicitly uses the current culture. Tests compare English and Swedish execution, rather than claiming equivalence across every possible host culture.

These dependencies should be visible during migration. A Linux container may have a different local timezone or culture from a dedicated Windows agent. An injectable clock or a newly fixed culture policy could improve future testability and determinism, but each should be evaluated as an explicit behavior change.

Failure behavior and responsibility

The public methods synchronously return a complete `ConversionResult` or throw. Their conversion path neither publishes partial output nor performs delivery. A caller that receives an exception has no returned batch to transfer. [20]

Table 19. Public failure contract

Condition	Public behavior
Null required object or data	<code>ArgumentNullException</code>
Empty data, blank company or invalid structured references	Argument validation exception
Unsupported Edlevo family or invalid Edlevo mapping value	<code>ConversionException</code>
Invalid SpeedAdmin amount or unknown present counterparty	<code>Contextual FormatException</code>
CR/LF in a scalar output field	<code>ConversionException</code>
Cancellation observed at a checkpoint	<code>OperationCanceledException</code>

A nonempty envelope-only input can return an empty body with zero counts. The calling process must enforce any requirement for at least one invoice. Conversion success, durable delivery and receiver acceptance remain separate states, with separate retry and recovery policies. [08]

Cancellation is cooperative, without a latency guarantee

The public task checks cancellation before conversion, within selected conversion loops and while counting output records. A requested cancellation becomes an exception when execution reaches a checkpoint. Whole-input decoding, some collection materialization, SpeedAdmin's inner CSV character parser and output encoding are not individually interruptible operations. [18, 19, 20]

The automated public-contract test establishes pre-canceled-token behavior; it does not establish a maximum cancellation delay during a large conversion. Target-host acceptance should measure that delay using the supported input envelope, realistic concurrency and container CPU limits. No bounded mid-flight cancellation latency is claimed here.

05.3 Separate compatibility from deliberate improvements

A successful replacement does not need to reproduce every incidental defect of the old hosting arrangement. It does need a clear account of what is retained and what is changed.

The original wrapper depended on a Windows shell, physical parameter files, executable-relative temporary directories and a shared debug-log path. It waited without an implemented timeout, did not inspect the executable's exit code as its primary success check, and treated certain returned keys and a minimum output length as evidence of completion. Its INI helper also mishandled values containing additional equals signs.

Those behaviors explain operational fragility. They are not invoice-mapping rules that a new byte-oriented library should recreate.

The distinction can be stated explicitly:

Table 20. Compatibility and intentional boundary changes

Behavior	Treatment in the replacement	Reason
Fixed target bytes, record order and field selection	Retained and compared	Receiver-visible compatibility
Nonadjacent grouping and global heading suppression	Retained	They alter output invoices and text
Source 63 amount and legacy zero/sign selection	Retained	They determine financial rows
KJ1 blank reference and file-wide BETIK selection	Retained	Existing reference behavior

Table 20. Compatibility and intentional boundary changes (continued)

Behavior	Treatment in the replacement	Reason
Shell invocation, INI rewriting and intermediate files	Removed	Internal execution machinery
Shared debug file and cleanup quirks	Removed	No equivalent requirement in the new model
Null or empty required inputs	Explicitly rejected	Clear new task boundary
CR/LF inside scalar output or reference values	Explicitly rejected	Prevent malformed physical output
Unknown or ambiguous reference labels	Explicitly rejected	Configuration errors become visible
Empty converted result	Returned with counts where conversion permits it	Caller policy replaces the old wrapper's arbitrary ten-byte threshold

A nonempty envelope-only input can therefore differ at the API boundary from the old wrapper, even though accepted comparison cases produce identical bodies. “Compatible” should be scoped to the supported conversion contract and recorded evidence. It should not mean identical exception types, process side effects and malformed-input behavior over every conceivable byte sequence.

Business changes deserve a separate queue. Summing repeated accounting records, reconciling control totals, filling missing dates, fixing quoting or populating a previously blank reference may be worthwhile. Combining those with the initial replacement would make every output difference ambiguous.

A practical change record should state whether a rule is observed legacy behavior, an intentional public-contract change or a proposed business correction. For a correction, retain a before-and-after example and obtain an agreed expected result from the people responsible for the financial process.

06 Verification and assurance

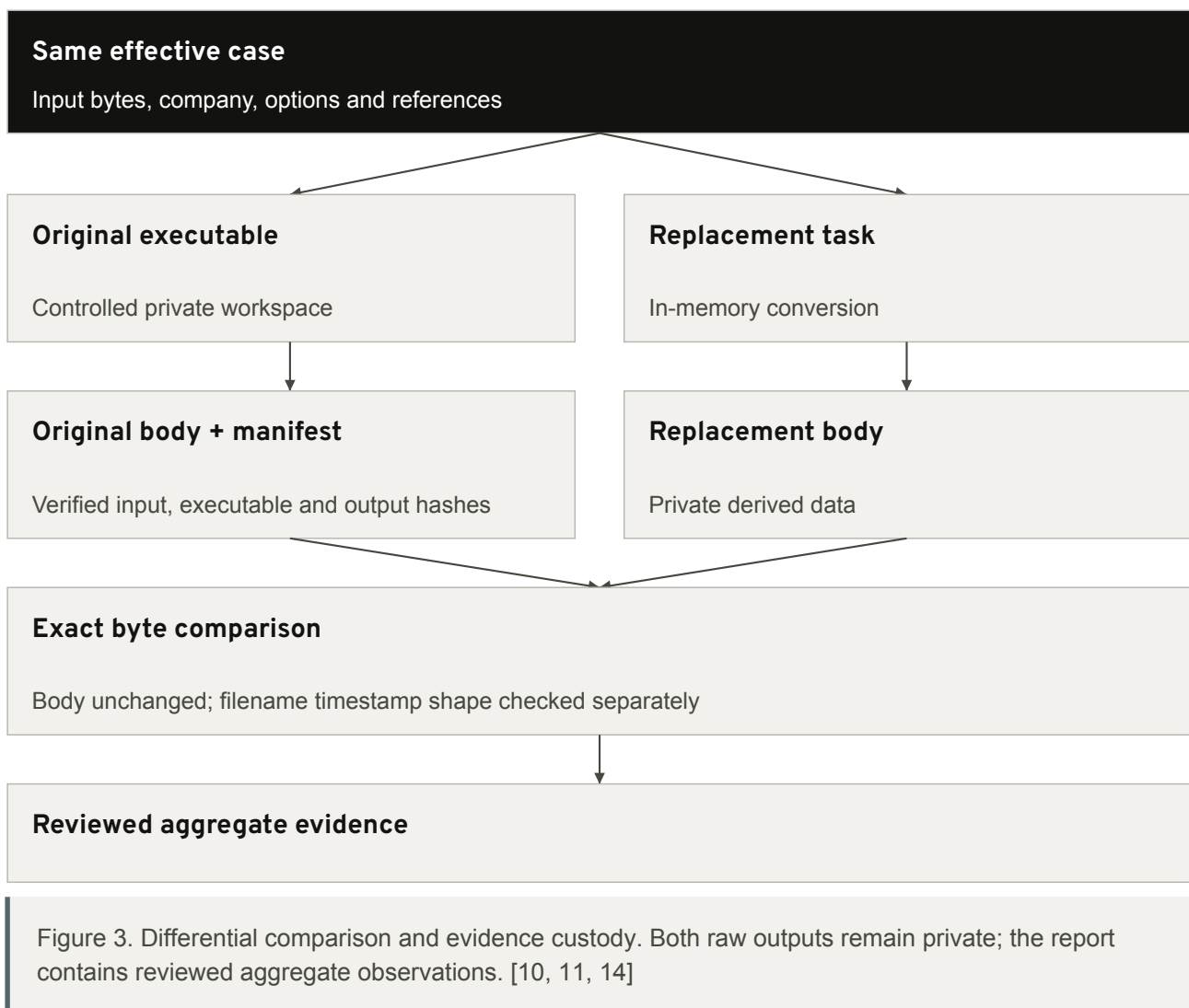
Use original-executable evidence, synthetic cases and independent financial validation for their distinct purposes.

SOURCE BASIS [06, 10, 11]

06.1 Build an executable oracle before optimizing

The old executable is an oracle for historical behavior. It is not necessarily an oracle for business correctness.

A differential harness runs the same effective case through old and new implementations and compares their results. The effective case includes bytes, company, options and reference values, not merely a path to a source file.



The harness should control working directories, isolate temporary names, record exit status, retain diagnostics appropriately and apply explicit timeouts. It should distinguish a conversion failure from a missing result path or a failure reading the result. Long-running original programs make this discipline more important, because losing provenance after a 46-minute execution is expensive.

The project also encountered source files being corrected during the investigation. The harness checks the source fingerprint after an original run as well as before it. If bytes change while a slow process is running, a nominally correct comparison may otherwise relate the output to the wrong version of the input.

Compare exact bodies

The actual comparison examines the complete byte arrays. It covers encoding, CRLF, trailing spaces, field widths, order, text, identities, signs, amounts and accounting records.

SHA-256 fingerprints accompany that check for tracking and cache validation. They are useful compact evidence, but the harness does not rely solely on matching totals or a visual diff.

The only output normalization is a timestamp-shaped token in the suggested filename. For the tested filename configurations, this masks the generated timestamp and leaves the remaining filename characters to compare. The implementation uses a timestamp-pattern regex rather than a separately parsed filename structure, so future custom prefixes containing the same pattern would require additional scrutiny. The file body is not normalized.

This is a crucial choice. Trimming every line, converting all newlines, sorting records or ignoring text differences can make a comparison pass by erasing the very behavior a fixed-width receiver depends on. Normalize a field only when its nondeterminism has been independently identified and its compatibility requirement is separately checked.

When a difference occurs, byte offset, record number, field position and hexadecimal expected/actual values are often more useful than a large raw-text diff. Such diagnostics can avoid printing entire citizen records, although even small excerpts and exception strings still need a data policy.

Trace a byte mismatch back to its selector

Consider a synthetic comparison where an R amount magnitude agrees but column 78 differs: the original contains hexadecimal 2D, a minus sign, and the replacement contains hexadecimal 20, a space.

The first question is which field owns the byte. Column 78 is the separate credit marker, so changing amount precision or padding would be a poor first response. Trace the marker to its sign predicate, then identify which source collection that predicate evaluates.

In current Edlevo, checking only the first amount would miss the recovered “any associated amount is negative” behavior. A constructed detail with one positive and one later negative amount distinguishes the predicates without needing a production record. The fix belongs at the selection rule, and the new test should explain that rule.

Likewise, a mismatch in an accounting dimension should first identify whether the input family selects ordinary or counter-account dimensions. A mismatch in text may originate in grouping or heading state rather than in the writer. A different record count may indicate a zero filter or missing group identity before it indicates a serializer problem.

Because output record lengths differ, mapping an absolute byte position back to a column requires the actual preceding records. For a zero-based absolute offset:

```
offset = sum(previous record widths + 2 CRLF bytes)
         + (one-based column - 1)
```

This technique makes a large diff tractable. Locate the earliest meaningful divergence, assign it to a schema field, trace its selector and construct the smallest distinguishing input. Later differences may be consequences of the first missing or extra record.

Keep slow originals reusable

Rerunning every original executable after every internal refactoring is unnecessary when the input and its effective settings are unchanged. A verified capture is a reusable oracle.

The current 0.2.0 reference-shape change was checked against saved original outputs. Every capture was validated against its input, executable, output and settings manifest before use. The replay ran the new conversion again; it did not claim to have freshly executed the old program.

This distinction preserves both efficiency and truthful evidence. A cache replay establishes that the new build reproduces a particular verified original result. It does not supply a new old-executable runtime measurement.

06.2 Use several complementary kinds of tests

No single test style covers this problem adequately.

Production comparisons reveal combinations and data variation that the engineer did not invent. Synthetic tests isolate difficult branches and permit safe CI. Structural checks catch broken widths independently of a complete oracle. Business reconciliation and receiver validation answer questions that an old executable cannot.

The implemented evidence developed in stages:

Table 21. Validation chronology

Stage	Passing unit cases	Distinct original-output comparison cases	Evidence
Initial 0.1.0	35	35	Seven supplied files and 28 constructed cases compared with actual original executions
Larger-file 0.1.0 work	35 at that stage	38 cumulatively	Three additional supplied Edlevo files, plus fresh benchmark repetitions of the original five Edlevo inputs
Current 0.2.0	58	38	New structured-reference implementation replayed against verified original captures

The 38 distinct comparison cases comprise eight supplied Edlevo inputs, two supplied SpeedAdmin inputs, 22 synthetic Edlevo cases and six synthetic SpeedAdmin cases. They are not 38 production files.

The initial production corpus was skewed: its five Edlevo files were all KJ1, and its two SpeedAdmin examples exercised ordinary positive PRIV/MF charges. Those examples did not establish the other Edlevo families or exceptional SpeedAdmin behavior. Constructed oracle cases supplied that branch evidence. The three later Edlevo captures increased observed scale and data volume; their addition should not be mistaken for proof of every production family.

The 0.2.0 replay performed zero fresh original-executable executions. All 38 complete bodies and filename shapes matched. The Release solution built with zero warnings and errors.

Synthetic fixtures should reveal assumptions

The unit fixtures are constructed examples rather than copied production invoices. Their field layouts are declared independently of the converter's parser. That helps avoid a circular test where a parser and its fixture generator share the same wrong offset.

The synthetic unit suite and constructed executable comparisons together cover more than ordinary positive charges: identity selection, grouping, zero filtering, repeated accounting records, company and year rules, text options, encoding and BOMs, decimal formatting, cancellation, buffer ownership and concurrent calls.

Structured-reference tests cover each K22 route, file-wide BETIK selection, KJ1's blank reference, partial and empty mappings, case-insensitive labels, duplicate or unknown labels, null and multiline values, configuration non-mutation, JSON round-tripping and rejection of the former shape.

A useful fixture usually varies one important condition at a time. For example, placing the same customer key in two nonadjacent blocks distinguishes grouping from simple sequential processing. Placing equal headings on opposite sides of a customer boundary distinguishes global suppression from per-invoice suppression.

The synthetic corpus should also include combinations selected to expose collection semantics: an absent amount versus zero, a first positive amount followed by a negative one, a present empty-looking field versus an omitted one, and different values in repeated accounting rows.

Additional tests should be chosen for the unresolved risk

Property-based or generated testing could extend this project by producing many valid structural combinations, then comparing old and new behavior within a controlled domain. Fuzzing could target truncation, delimiters and malformed records. Those are possible extensions, not tests claimed as completed here.

Metamorphic tests can help when an exact expected body is expensive to construct. Examples include verifying that changing an unused source field leaves the body unchanged, or that a configuration change affects only its documented output slots. A proposed relation must itself be justified: reordering customer blocks is not generally invariant when grouping order and global heading suppression matter.

Mutation testing can assess whether tests detect a removed zero filter or changed sign predicate. Again, a percentage from such a tool would be additional evidence rather than a substitute for understanding which business cases are represented.

Test counts are useful inventory, not a coverage theorem. Fifty-eight well-chosen tests can still omit an entire production family. A larger corpus should be selected by families, counterparties, exceptional credits, protected-address handling, multiple-detail patterns, dates, configuration combinations and source-system versions rather than raw file count alone.

Compatibility and business correctness answer different questions

Exact body agreement is strong evidence that the replacement reproduces the old mapping for those cases. It also means that any latent rule defect shared by both remains shared.

Independent financial validation should ask whether customer selection, invoice grouping, signs, charge totals, account dimensions, VAT treatment and references agree with the intended billing process. Receiver validation should ask whether Raindance accepts the import and creates the expected financial documents.

The following questions identify proposed independent validation work and the authority needed to answer it:

Table 22. Independent financial validation

Question	Independent evidence
Are the encoded monetary units kronor, öre or another scale?	Source export specification together with receiver configuration
Which source identities and blocks should constitute one invoice?	Approved billing policy and representative expected invoices
Should repeated accounting records be selected or summed?	Finance-owned allocation rules
Are signs and VAT codes correct for charges and credits?	Agreed exceptional cases and receiver posting results
Is a blank reference intentional for a particular route?	Reference policy and the receiver's treatment of blanks
Are all intended charges represented after filtering and grouping?	Source control totals interpreted with documented exclusions
Does protected-address behavior satisfy the process requirements?	Authorized privacy and billing rules applied to safe test scenarios

An aggregate total is necessary in many reconciliations but is insufficient alone. Moving an amount from one customer to another can preserve the batch total. Dropping a positive and equal negative amount can preserve a net total while losing both transactions. Reconciliation should therefore use the appropriate grouping and sign dimensions as well as the overall sum.

The replacement's output counts are conversion facts. They are not acknowledgments from Raindance, nor proof that citizens received correct invoices.

The current evidence does not include a live Raindance import, a Friends package import or a deployed Kubernetes run. The code and configured CI support those next steps; they do not make them historical facts.

Corpus coverage by evidence type

The 38 distinct original-output comparisons comprise ten supplied files and 28 constructed cases. All eight supplied Edlevo inputs exercise KJ1/K02. The three larger captures increase size and customer-volume coverage; K22-family coverage remains synthetic. Counts describe the validation corpus, not exhaustive coverage of the accepted input language. [06,10,11]

Table 23. Original-output comparison corpus

Converter	Supplied files	Constructed cases	Total comparisons
Edlevo	8	22	30
SpeedAdmin	2	6	8
Total	10	28	38

07 Performance and its explanation

Measure the operational execution model and identify the repeated work the implementation removes.

SOURCE BASIS [12, 13]

07.1 Measure the execution model honestly

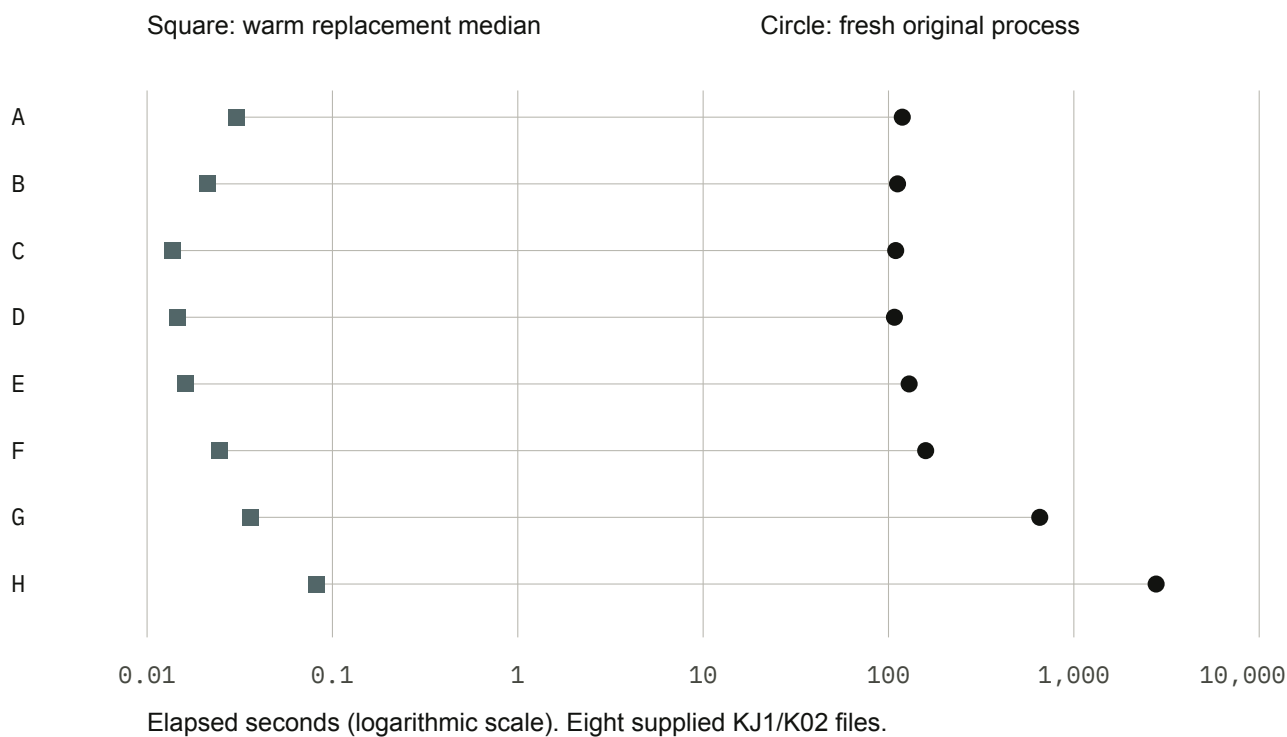


Figure 4. Observed execution time, version 0.1.0. Each original point is one process execution; each replacement point is the median of 15 warm calls after three warmups. Different timing boundaries; no end-to-end delivery or import measurement. [12, 13]

The benchmark compared fresh original-executable runs with warm calls to the replacement task. It measured eight supplied Edlevo inputs using the 0.1.0 implementation on .NET 8.0.32.

The table uses anonymous batch labels and aggregate sizes. Every complete output body was identical.

Table 24. Controlled Edlevo timing results, version 0.1.0

ID	Input bytes	Output customers	Original seconds	New median milliseconds	Observed ratio
A	952,227	440	118.6793	30.2218	3,927×
B	966,623	446	111.9017	21.1507	5,291×
C	978,993	451	109.3571	13.7549	7,950×
D	981,711	454	107.6558	14.5786	7,385×
E	1,103,741	449	129.2266	16.0417	8,056×
F	1,192,465	563	158.6994	24.6806	6,430×
G	2,185,101	1,009	654.6798	36.3487	18,011×
H	4,132,439	1,882	2,778.8407	81.6151	34,048×

TABLE NOTE

Batch H produced 3,145,881 output bytes: 1,882 S records, 1,882 H records, 12,815 R records and 4,300 K records. Its original runtime was 46 minutes 18.8407 seconds. The replacement's 15 measured calls ranged from 71.4853 to 105.9968 milliseconds, with the median shown above.

These measurements remain labeled 0.1.0. Version 0.2.0's reference redesign passed output replay, but its replay timings are not a new controlled benchmark.

Original timing boundary

Each original input ran in a fresh child process. The stopwatch began immediately before process start and stopped after process exit. It included process startup, input and parameter reading, workbook access, conversion, intermediate-file operations and output writing.

Preparing the executable workspace and INI was outside the timer. Reading the final output for comparison, persisting captured logs, hashing and reporting were outside it as well.

There was one completed original measurement per input. That is an observation for each case, not an estimate of the original's median or variability.

Replacement timing boundary

The replacement ran in an already loaded .NET process. Input bytes and reference values were prepared before timing. Each input received three unmeasured warmups followed by 15 measured public task calls.

The timed call included parsing, configuration lookup preparation, mapping, output byte creation and result counting. It excluded file retrieval, workbook-to-object adaptation or JSON deserialization before the call, output persistence, comparison and reporting. Garbage collection was not forced between iterations; any collection during a call contributed to its elapsed time.

The runs were serial, without competing converter jobs or builds. The first five files used an original-process timeout of 1,800 seconds. The three larger files used 3,600 seconds. An incomplete larger-file attempt was restarted with that longer timeout and contributes no completed timing result.

The machine reported Windows 10.0.26100, X64, an Intel Core i7-12850HX and 24 available logical processors. The replacement was built in Release for net8.0.

What the speedup means

The ratio measures the two integration execution approaches used in this case. Removing process startup, workbook loading and disk intermediates is part of the improvement. It does not isolate only the mapping algorithm.

Warm task timings do not include agent startup, first-call JIT compilation, source transport, downstream delivery or Raindance import. There is no controlled contemporary SpeedAdmin performance table equivalent to the Edlevo benchmark.

Small-file timings are not strictly monotonic with file size. They reflect data shape and ordinary execution variability as well as byte count. A benchmark report should preserve the samples rather than fitting a persuasive curve to a small dataset.

The original and new boundaries are appropriate for understanding the operational replacement, provided they are stated. A parser microbenchmark would require a different experiment: comparable in-memory inputs, warmed runtimes and isolated parsing or mapping operations. It would answer a narrower question.

07.2 Understand the cause before attributing the result

The generated Edlevo final mapping repeatedly opens and parses the complete intermediate document while emitting individual customers. Two customer-level paths contain this behavior. The underlying text-document parser calls `ReadToEnd`, even where a later selector only needs a header-level fact.

For a document of size N and C customer groups, a repeated complete-document operation has work proportional to $C \times N$. If customer count grows with document size, that component can grow approximately quadratically. This describes an identified operation, rather than a formally derived complexity bound for every part of the original program.

The replacement performs one structural Edlevo parse, computes file-family and reference decisions outside customer-level repetition, and works over the parsed groups. It removes repeated full-document disk reads and parsing from those customer paths.

The public method still performs some additional linear work. It decodes and splits the source again to derive the delivery family, and scans the generated result to count record kinds. Describing the replacement as “touching every byte exactly once” would be false. The relevant improvement is eliminating repeated global reconstruction inside per-customer work.

This is an example of a substantial optimization that does not require changing financial policy. Hoisting stable file-level facts and retaining an explicit source representation changes where work occurs, while byte comparison constrains what it produces.

No profiler was used to allocate percentages of the original runtime among parsing, allocation, workbook I/O and other costs. The code provides a concrete explanation consistent with the measurements; it does not justify attributing every millisecond to one helper.

It would also be unfair to generalize this result into a claim that all generated MapForce mappings perform poorly. This executable's generated dependency paths and data access pattern are the evidence. Other mappings can have very different shapes.

08 Data custody and reproducibility

Keep private data outside the distribution boundary and connect results to an identifiable build.

SOURCE BASIS [09, 14, 15]

08.1 Privacy is part of differential testing

Using real production invoices creates a data-handling problem before it creates a testing advantage. The output is not a harmless derivative: it can repeat names, identifiers, addresses, descriptions and relationships from the source.

The project exposed a concrete boundary mistake. The original Test inputs remained outside the integration folder, but early comparison tools wrote raw original outputs, replacement outputs and intermediate files beneath an ignored artifacts directory inside it. A folder copied or shared as a unit would still have carried those production-derived files.

The correction preserved the three raw artifact families in a sibling `PrivateValidationArtifacts` directory outside the integration's distribution boundary. The original Test inputs were unchanged. Both oracle tools now default to the external location and reject internal raw-output paths. The parity runner rejects internal report paths too, since failure reports can contain source values.

An ignore file reduces accidental version-control inclusion. It does not protect a directory copied to a colleague, indexed by another tool, included in a support bundle or archived wholesale. The private directory's name is not encryption or an access-control policy.

The subsequent audit examined 625 integration files and 420 expanded archive entries, using 70,716 in-memory variants derived from ten source fixtures. It found no matching production names, identity numbers or street addresses in the remaining audited integration scope. The matches were generic confidentiality markers. The audit covered source, documentation, tests, ignored build outputs, caches and packages.

That was a scoped exact-seed audit, with normalization variants, rather than a universal proof that no possible personal information could exist. Compressed embedded source inside Portable PDB blobs was not separately decompressed; corresponding project sources and ordinary archive contents were checked.

The practical lesson is to design data lineage alongside the oracle:

Table 25. Data custody across the validation pipeline

Artifact	Why it may contain production data	Appropriate treatment
Original source	Direct citizen and invoice information	Authorized private fixture store
Original and replacement output	Transformed copies of the same information	Same custody as source
Intermediate files	Often retain fields omitted from final output	Same custody as source
Logs and exception messages	Can interpolate field values or complete records	Review, minimize and restrict
Diff reports	Can expose the exact field that failed	Private by default; sanitize public summaries
CI test results	Can contain failure arguments and assertion values	Use synthetic fixtures in ordinary CI
Packages and symbol archives	Can accidentally include fixtures, reports or source constants	Inspect archive contents before distribution
Counts, timing and fingerprints	Usually suitable for reviewed engineering evidence, but still contextual metadata	Retain only what the audience needs

A file hash is useful for provenance but should not be treated as an anonymization technique for personal identifiers. Nor is a batch filename automatically non-identifying. Public evidence needs review at the metadata level as well as the body level.

Synthetic fixtures are the long-term maintainable boundary for ordinary development and CI. They should reproduce structural and semantic edge cases without copying an individual's record. Real production replay remains valuable in a controlled validation environment.

08.2 Build and release are part of reproducibility

The production library targets .NET 8 because that is the agents' requested runtime. C# 12, nullable analysis and warnings as errors support explicit implementation and review. The production dependency graph consists of framework libraries; test tooling is kept outside the production assembly.

Target framework, build SDK and execution runtime are separate facts. Local verification used installed SDK 10.0.402 to compile against .NET 8 reference

assemblies, and executed on .NET 8.0.32. That does not make the resulting package require .NET 10.

The adapted GitHub workflows explicitly install and select SDK 8 using a job-local `global.json`. This prevents a newer preinstalled runner SDK from silently determining the build. It also avoids imposing a new machine-wide SDK setup on local developers.

The push and pull-request workflow is configured to build and run the synthetic suite on Windows and Ubuntu. It builds the parity console but does not run original executables or consume production invoices. The Linux job packages the library, and both jobs retain unit-test results.

The release workflow follows the supplied RemoteFS release pattern while making its contract explicit: published release, exact-tag checkout, validated version, build and tests, then package publication and release assets. The workflow filename does not define its trigger; its event configuration does.

Several details matter even in a small pipeline. The assembly and NuGet package must receive the same release version. Tag text is untrusted shell input and must be validated. Credentials belong in temporary feed configuration, outside uploaded artifacts, with cleanup. Ordinary CI needs less repository permission than a job that attaches release files.

The .NET 8 NuGet push path also required using its supported configuration behavior rather than copying an SDK 9-era option. A pipeline can look correct and still rely on a newer CLI feature than the selected SDK provides.

The workflows passed local linting and script checks, and local build, tests and packaging succeeded. A configured Windows/Ubuntu matrix is not evidence of an executed GitHub-hosted or Linux run. No live feed publication or deployment is claimed by this case study.

The integration folder is intended to become the repository root. GitHub will not discover its workflows if the entire folder remains an arbitrary nested directory in a larger repository. Repository layout is therefore part of the release handover.

09 Migration and operation

Treat deployment, delivery, acknowledgements and rollback as explicit integration responsibilities.

SOURCE BASIS [08, 09, 10, 15]

Removing the Windows-only executable dependency makes operation in the normal Kubernetes-based integration environment feasible. It does not, by itself, complete that migration.

The potential operational value extends beyond conversion latency. The organization may be able to retire dedicated converter-agent processes, remove executable installation and temporary-directory requirements, and use its ordinary integration scheduling and deployment model.

The operational estimate was that roughly eight dedicated agent processes could be retired. That is an opportunity rather than a measured capacity or cost reduction already achieved. Actual consolidation depends on the surrounding workflows, concurrency, transport, availability requirements and validation process.

There is also a less obvious consequence of a very large speedup: downstream pressure changes. A converter that naturally spaced output batches by minutes can begin producing them in milliseconds. Transport limits, duplicate prevention, import scheduling and monitoring need to cope with that new rate.

Keep conversion success separate from delivery success

At least three operational events should be distinguished:

1. Source bytes were successfully converted.
2. The output was durably transferred to the intended receiving location.
3. Raindance accepted and processed the intended batch.

The library establishes the first event. The surrounding integration owns the others.

A retry after an ambiguous transfer can create a duplicate import even when conversion is perfectly deterministic. The suggested filename has second precision and does not reserve a path. It is unsuitable as the sole idempotency mechanism.

An operational batch identity should bind the source, relevant configuration and intended business execution. A source hash alone may not identify the intended run if the same bytes can legitimately be processed with different

settings. A new generated timestamp alone changes on retries and can hide duplicates.

These are delivery-design considerations, not features claimed as implemented inside the converter. The useful boundary is to keep the conversion deterministic enough to inspect while making transport acknowledgment and retry state explicit outside it.

Suggested filenames require delivery policy

Filename is delivery metadata. The public task composes it from company, family or configurable SpeedAdmin prefix, local timestamp and extension, then normalizes XML whitespace. Company validation rejects blank values but does not establish a filesystem-safe naming vocabulary. [20]

The delivery adapter must apply destination-specific filename validation and path confinement before storage. It must also manage uniqueness and idempotency: the suggested name has second precision and reserves no path. These responsibilities belong beside transport and acknowledgment handling, rather than being inferred from successful conversion. [08, 20]

A staged rollout preserves evidence

A practical next validation sequence is to gather representative files by business family, replay old and new under matched settings, review financial meaning independently, execute the synthetic suite on the target runtime and host, and test the packaged Friends task in an isolated process.

Shadow conversion can then compare results without delivering both outputs. Shadow comparison must not submit both generated batches. The delivery path should have a single authoritative producer at each stage.

An agreed rollout should include stopping conditions, such as a new unsupported family, a byte mismatch, an unexpected zero-invoice result or a receiver rejection. The old executable and its configuration should remain recoverable during the transition.

Rollback is also a state question. Switching code versions does not undo an invoice already imported by the receiver. The integration's record of delivered and acknowledged batches is essential to recovering safely from an incident.

Kubernetes deserves its own execution evidence

The implementation avoids Windows shell and path requirements and explicitly emits legacy CRLF and Latin-1 bytes. Target-host testing should still cover timezone, culture, cancellation, CPU limits, memory limits, concurrent calls and transport handling.

Warm local execution time does not establish performance under a constrained container CPU quota. A few-megabyte in-memory conversion may be inexpensive, but concurrency and retained outputs can affect memory more than a single median runtime suggests.

Monitoring can remain useful without recording personal data. Examples include input and output byte counts, record counts, conversion duration, package version, effective configuration fingerprint, batch status and a correlation identifier. Error channels need the same scrutiny as successful-path logs because exceptions may contain actual source values.

10 A reusable engineering method

Leave an explainable contract that the next engineer can inspect, reproduce and operate.

SOURCE BASIS [01-20]

10.1 A practical starting method for another undocumented converter

The following sequence is intended for a project that has an operational converter but incomplete design knowledge. It separates investigation from replacement so each phase leaves useful evidence.

Preserve and inventory

Freeze copies of the executables, configurations and supporting files. Fingerprint them. Identify what is deployed rather than trusting suffixes such as “new” or “previous.” Record whether examples are production, synthetic or of uncertain provenance.

Define the authorized data locations before the first oracle run. Keep the source repository and its distribution folder free of raw production copies.

The useful deliverables are an artifact inventory, a version map and an effective-invocation description.

Establish one reproducible successful run

Find the actual invocation path, working directory, parameter generation and result discovery. Run one known input with controlled settings and capture output and provenance. Observe which files are created or modified.

Do not start with the largest slowest file unless that is the only available evidence. A small successful case makes it easier to distinguish setup problems from conversion behavior.

The useful deliverable is a repeatable oracle case, including its private raw evidence.

Recover the target and then the source

Extract output schemas first when they are easy to locate. They provide stable destinations for tracing. Recover input schemas and structural splitting next.

For each important output field, trace its dependency back through intermediate representations. Record blank behavior, normalization, first/last selection, grouping and predicates.

The useful deliverables are a physical data dictionary, a transformation matrix and a list of unresolved business meanings.

Challenge assumptions with small probes

Construct cases that distinguish plausible interpretations. To distinguish grouping from sequential processing, repeat a key nonadjacently. To distinguish first-value from sum semantics, supply different repeated values. To distinguish a missing field from zero, test both.

Preserve every meaningful discovery as a synthetic case with an explanation. A surprising result without a test is likely to be “fixed” by the next maintainer.

The useful deliverable is a branch-oriented characterization corpus.

Implement the smallest explicit model

Separate parsing, mapping, serialization and orchestration. Name rules after their responsibilities. Keep stable file-level facts outside per-record loops. Preserve the output contract before optimizing convenience.

Prefer straightforward structures until measurements justify greater complexity. In particular, avoid selecting streaming, parallel mapping or a generic rule engine before understanding ordering and global state.

The useful deliverable is a reviewable implementation whose rule locations correspond to the recovered documentation.

Compare before measuring

Run the same effective cases through both implementations. Resolve every unexpected difference. Keep nondeterminism narrow and explicit. Distinguish a newly discovered old rule from an intentional improvement.

Only then collect performance evidence with stated boundaries, serial runs where appropriate, warmup policy and retained raw samples.

The useful deliverables are a compatibility report and a reproducible benchmark, each tied to implementation fingerprints.

Validate the financial process and hand it over

Bring in the people who own the billing and receiver configuration. Check the intended financial meaning, not just legacy agreement. Test the actual package and host. Define delivery status, retry handling and rollback responsibilities.

The useful deliverable is a handover that lets another engineer explain a particular output byte, reproduce a result and operate the process without relying on the original investigator.

10.2 Decisions worth making explicit

Several choices recur across this kind of project. Their value lies in the reason and boundary, rather than in a universally correct answer.

Table 26. Design decisions and tradeoffs

Decision	Choice in this project	Tradeoff
Behavioral baseline	Current Edlevo and supplied SpeedAdmin executables	Clear scope; previous Edlevo remains outside the implementation
Physical fidelity	Exact Latin-1, CRLF, widths, spaces and order	Receiver compatibility; legacy representation remains visible
Financial transformation	Preserve observed selection and sign rules	Reduces migration differences; retains rules requiring separate business review
Input strictness	Retain recovered structure with selected boundary guards	Accepts historical shapes; does not become a comprehensive source validator
Runtime model	In-process .NET 8 conversion	Removes process and disk machinery; uses application memory
Memory model	Whole-file representation	Simple global grouping; requires concurrency and size policies
Configuration	Typed object with structured JSON representation	Reviewable and dependency-free; subtype vocabulary becomes an API
Testing	Synthetic suite plus private executable oracle	Safe repeatable CI and real behavior evidence; corpus remains finite
Slow regression baselines	Verified original-output cache	Fast replay; provenance checks become essential

Table 26. Design decisions and tradeoffs (continued)

Decision	Choice in this project	Tradeoff
Performance evidence	Fresh original process versus warm public task	Relevant operational comparison; does not isolate the algorithm alone
CI data	Synthetic fixtures only	Portable and shareable; private production replay remains separate
Deployment evidence	Local Windows verification and configured cross-platform CI	Strong preparation; target-host and receiver acceptance remain separate work

10.3 What the next maintainer should inherit

The enduring output of reverse engineering is an explainable contract. Source code is one part of that contract; field dictionaries, synthetic examples, executable provenance, compatibility reports and operational boundaries are the rest.

A future engineer should be able to answer why an amount came from 63 rather than 62, why two source blocks became one invoice, why a reference stayed blank, why a heading disappeared, and why an apparently harmless serializer change broke byte parity.

They should also be able to identify a different category of question: whether that historical rule is still the desired business policy. Preserving the distinction lets the organization modernize behavior deliberately after it has modernized the implementation.

The large performance improvement matters because it changes what can be operated economically and where it can run. The more durable improvement is that the organization can now inspect and test the transformation without reconstructing a lost visual project or depending on opaque executable behavior.

Appendix A Output field dictionary

The 65 declared fields below describe the supplied converters' shared physical output layout. This is a recovered implementation contract, not an independently supplied official Raindance specification. Field capacity does not establish whether a mapping populates a field. [16]

Positions are one-based and inclusive, exclude the CRLF terminator, and restart on each physical record. For representable Latin-1 output, character and byte positions coincide. Values are space-filled, left aligned and truncated on the right by the writer; amount expressions add leading spaces before serialization.

A.S Customer

Table 27. S record: complete declared field layout

Start	End	Width	Generated field name	Meaning
1	1	1	postmarkering	Record marker: S, H, R, or K.
2	2	1	Field	Reserved space.
3	14	12	kundidentitet	Customer identifier field. Mapping may leave it blank.
15	54	40	namn_1	Name line 1.
55	94	40	namn_2	Name line 2.
95	134	40	adress	Street/address line.
135	143	9	postnummer	Postal code field.
144	144	1	Field2	Reserved space.
145	174	30	ort	Town/city.
175	190	16	momsregistreringsnr	VAT registration number.
191	194	4	Field3	Reserved space.
195	206	12	person_orgnr	Personal/organization identifier.
207	209	3	Field4	Reserved space.
210	211	2	landskod	Country code field.
212	214	3	Field5	Reserved space.
215	224	10	motpart	Counterparty.
225	234	10	kundtyp	Customer type.
235	249	15	actor_id	Actor ID field.

Table 27. S record: complete declared field layout (continued)

Start	End	Width	Generated field name	Meaning
250	259	10	plusgiro	Plusgiro field.
260	269	10	bankgiro	Bankgiro field.
270	285	16	bankkonto	Bank account field.
286	289	4	Field6	Reserved space.
290	302	13	gln	GLN field.
303	304	2	Field7	Reserved space.
305	305	1	sekretessmarkering	Confidentiality marker field.

A.H Invoice header

Table 28. H record: complete declared field layout

Start	End	Width	Generated field name	Meaning
1	1	1	postmarkering	Record marker: S, H, R, or K.
2	2	1	Field	Reserved space.
3	14	12	kundidentitet	Customer identifier field. Mapping may leave it blank.
15	20	6	fakturadatum	Invoice date field; width does not establish that it is populated.
21	26	6	bokforingsdatum	Posting date field.
27	32	6	forfallodatum	Due date field.
33	34	2	Field2	Reserved space.
35	64	30	var_ref_tabell	Our reference / reference table code.
65	94	30	er_referens	Your reference.
95	134	40	faktura_avser	Invoice subject/description.
135	144	10	motpart	Counterparty.
145	156	12	person_orgnr	Personal/organization identifier.
157	179	23	Field3	Reserved space.
180	199	20	reskontraposttyp	Receivable entry type field.
200	209	10	fakturanummer	Invoice number field.

A.R Invoice row

Table 29. R record: complete declared field layout

Start	End	Width	Generated field name	Meaning
1	1	1	postmarkering	Record marker: S, H, R, or K.
2	2	1	Field2	Reserved space.
3	62	60	radtext	Row text. K text is shorter than R text.
63	77	15	radbelopp	Invoice row amount magnitude. Interpret with the separate credit marker.
78	78	1	kreditmarkering	Separate negative/credit marker, populated with - for a negative amount in these mappings.
79	81	3	momskod	VAT code.
82	89	8	antal	Quantity.
90	90	1	kreditmarkering2	Separate quantity sign field.
91	105	15	a_pris	Unit price field.

A.K Accounting row

Table 30. K record: complete declared field layout

Start	End	Width	Generated field name	Meaning
1	1	1	postmarkering	Record marker: S, H, R, or K.
2	2	1	Field	Reserved space.
3	12	10	koddel_1_konto	Dimension 1: account.
13	22	10	koddel_2_ansvar	Dimension 2: responsibility.
23	32	10	koddel_3_verksamhet	Dimension 3: operation/service.
33	42	10	koddel_4_aktivitet	Dimension 4: activity.
43	52	10	koddel_5_objekt	Dimension 5: object.
53	62	10	koddel_6_projekt	Dimension 6: project.
63	72	10	koddel_7_frikod	Dimension 7: free code.
73	82	10	koddel_8_motpart	Dimension 8: counterparty.
83	124	42	filler2	Reserved filler.
125	139	15	belopp	Accounting amount magnitude. Interpret with the separate credit marker.
140	140	1	kreditmarkering	Separate negative/credit marker, populated with - for a negative amount in these mappings.
141	144	4	Field3	Reserved space.
145	174	30	radtext	Row text. K text is shorter than R text.
175	184	10	periodisering	Periodization field.

Appendix B Evidence and acceptance

B.1 Evidence register

The evidence was recorded on 17 October 2026 in separate implementation and measurement stages. Fresh original-executable runs and later replay of verified captures answer different questions and must remain distinguishable. [10–13,15]

Table 31. Evidence register

Evidence stage	Version	Recorded result	Evidence boundary
Initial unit suite and parity	0.1.0	35 unit cases; 35 comparisons pass	Seven supplied files and 28 constructed cases
Controlled Edlevo benchmark	0.1.0	Eight comparisons pass	One fresh EXE run per input; three warmups and 15 timed public-task calls per input
Structured-reference replay	0.2.0	38 comparisons pass	Verified original captures reused; zero fresh EXE executions
Current unit suite	0.2.0	58 executed; 58 passed	Stored local Windows test results; no failures, errors, or timeouts
Build and release preparation	0.2.0	Local build, test, package and workflow checks recorded	Hosted CI, Linux execution, feed publication and deployment are not established by these records

TABLE NOTE

The five original Edlevo inputs were measured again during the benchmark and are counted once in the cumulative corpus. The benchmark added three distinct supplied files, increasing the comparison corpus from 35 to 38. Single-call replay timings for 0.2.0 are not a second controlled benchmark. [10–13]

The original-output comparison cases and unit cases are separate inventories. A replay does not create a new original-executable run, and a configured CI matrix does not establish an executed target-host result.

B.2 Implementation identity

The portable reports identify the measured or replayed implementation and reference configuration with SHA-256 fingerprints. These values establish artifact identity; they do not independently establish financial correctness or deployment status. [11,13]

Measured 0.1.0 library

4e48c53e733cb534928a3a596d2946228384e4ff141bf9d0b0e388d72f553eaa

Replayed 0.2.0 library

935737badfcaf3aa410215f2ef0848a9b29725c69fb9e41fd353a0adedfb8e22

0.2.0 structured reference JSON

249c28d5e3d5b65ae4172f3b71e87c881e30979ad30f1470467d11479b260e7c

Legacy reference workbook

a162a55e7acbdcdf0dd6b8ddd412510c439da9193edc7ac640ba145b3f845a81

B.3 Proposed migration acceptance gates

These gates describe remaining acceptance work, not completed implementation features or an approved production rollout.

Table 32. Proposed migration acceptance gates

Gate	Required evidence	Accountable role
Family coverage	Matched old/new replay for each production family, exceptional signs and configuration combination in use	Integration engineering
Financial meaning	Agreed identities, grouping, currency scale, VAT, references and separate R/K reconciliation	Financial-process owner
Runtime and package	Successful Friends package import and execution on the intended .NET 8 host; culture and timezone verified	Platform engineering
Resource envelope	Peak memory, concurrency and cancellation latency measured at supported size and container CPU quota	Platform engineering
Delivery integrity	Destination filename validation, path confinement, durable transfer, batch identity and duplicate prevention exercised	Integration operations
Receiver acceptance	Raindance accepts safe representative batches and produces the intended financial documents	Receiver owner and finance
Recovery	Stop conditions, authoritative producer, acknowledgement ledger and rollback exercise agreed	Service owner

Appendix C Sources and publication record

Repository references identify the supplied project files as reviewed on 19 October 2026. Numbered source groups at each chapter entrance connect its claims to these records. The source package includes SHA-256 fingerprints for the article and referenced files. No raw production input or derived invoice output is included.

The replacement repository was clean at inspection, at commit 642354d17cfa09f9f780a70e973ec44f5e4d5392. The surrounding investigation folder is not itself a Git repository. A commit identifies the checkout; the evidence reports separately identify the measured and replayed assemblies.

Path key: Investigation docs means `Docs/`; Replacement docs means `Friends.HIT.IKEToRaundance/docs/`; Library source means `Friends.HIT.IKEToRaundance/Friends.HIT.IKEToRaundance/`. Paths are relative to the supplied investigation folder.

[01] Runtime and configuration

Investigation docs: 01-runtime-and-configuration.md

[02] Current Edlevo format

Investigation docs: 02-edlevo-current.md

[03] Previous Edlevo and differences

Investigation docs: 03-edlevo-previous-and-differences.md

[04] SpeedAdmin format

Investigation docs: 04-speedadmin.md

[05] Reference workbooks

Investigation docs: 05-reference-workbooks.md

[06] Samples and validation

Investigation docs: 06-samples-and-validation.md

[07] Replacement architecture

Replacement docs: compatibility-and-architecture.md

[08] Public task contract

Replacement docs: task-contract.md

[09] Build and migration

Replacement docs: build-and-migration.md

[10] Validation evidence

Replacement docs: validation.md

[11] Structured-reference replay results

Replacement docs: structured-reference-validation-results.json

[12] Runtime comparison

Replacement docs: runtime-comparison.md

[13] Aggregate runtime report

Replacement docs: runtime-results.json

[14] Data handling

Replacement docs: data-handling.md

[15] CI and release

Replacement docs: ci-and-release.md

[16] Shared Raindance output layout

Investigation docs: 07-raindance-output-layout.md

[17] Edlevo financial row emission

Library source: Converters/Edlevo/EdlevoConverter.cs

[18] Edlevo source parsing and validation

Library source: Converters/Edlevo/EdlevoSource.cs

[19] SpeedAdmin mapping and parsing

Library source: Converters/SpeedAdmin/SpeedAdminConverter.cs

[20] Public task orchestration

Library source: Main.cs

C.1 External technical references

Microsoft. Encoding.Latin1 property. .NET API reference.

<https://learn.microsoft.com/en-us/dotnet/api/system.text.encoding.latin1>

Microsoft. Floating-point numeric types. C# language reference.

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types>

Y. Shafranovich. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. October 2005. Informational.

<https://www.rfc-editor.org/rfc/rfc4180>

W3C. XML Path Language (XPath), Version 1.0, section 4.2: normalize-space. 16 November 1999.

<https://www.w3.org/TR/xpath-10/#function-normalize-space>

C.2 Editorial and production record

Source. Adapted from *Recovering and replacing legacy financial file converters*, case study dated 17 October 2026. This edition preserves its substantive argument, reframes informal headings, adds source-verified technical qualifications and includes the complete output-field dictionary.

Identity. Ellie Augustini Identity Manual 2.0.1, 5 September 2026, maintained in Outline. The supplied stacked signature and complete EA01 vector artwork are used as masters. League Spartan ExtraBold provides display type, Overpass Bold provides functional headings, Arial provides reading text, and IBM Plex Mono provides code and metadata. Fonts are embedded in the reading PDF; font binaries are not redistributed with the source package.

Format. A4, mirrored 22 mm inner and 18 mm outer margins, 141 mm prose measure and 170 mm evidence measure. Main text is 11.5/16 pt. Signal Red marks orientation; interior reading pages use an unprinted white field. This is an RGB reading edition, not a supplier-approved press file.

Navigation and accessibility. The PDF contains selectable text, bookmarks, linked contents, document language, structural tags, table header associations and text alternatives for informative figures. A reflowable HTML edition is included. No PDF/UA certification or complete assistive-technology audit is claimed.

Review. Technical review reconciled the article with implementation, recovered schemas, API documentation, stored test results and aggregate replay/benchmark reports. Publication checks cover rendered pages, table and code fit, source references, output-field widths and metadata. Historical execution evidence remains distinct from this publication review.