Reconstructing the ISA Line

Once the four delimiters are trusted, rebuilding the canonical 105-byte ISA line is a total function: every non-fatal input maps to exactly one canonical line, everything else to a named fatal. There is no third outcome.

Engineering note x12-tidy / isa ~1,500 words PDF The method

← Those Pesky Delimiters

THE CLAIM

01 Non-fatal in, canonical line out — always

This is the third act of the x12-tidy method. The first note located the run; the second recovered the four delimiters from it. If nothing fatal came back, the delimiters are now **ground truth** — and this note is the payoff: rebuilding the ISA line to the byte-exact form the standard requires.

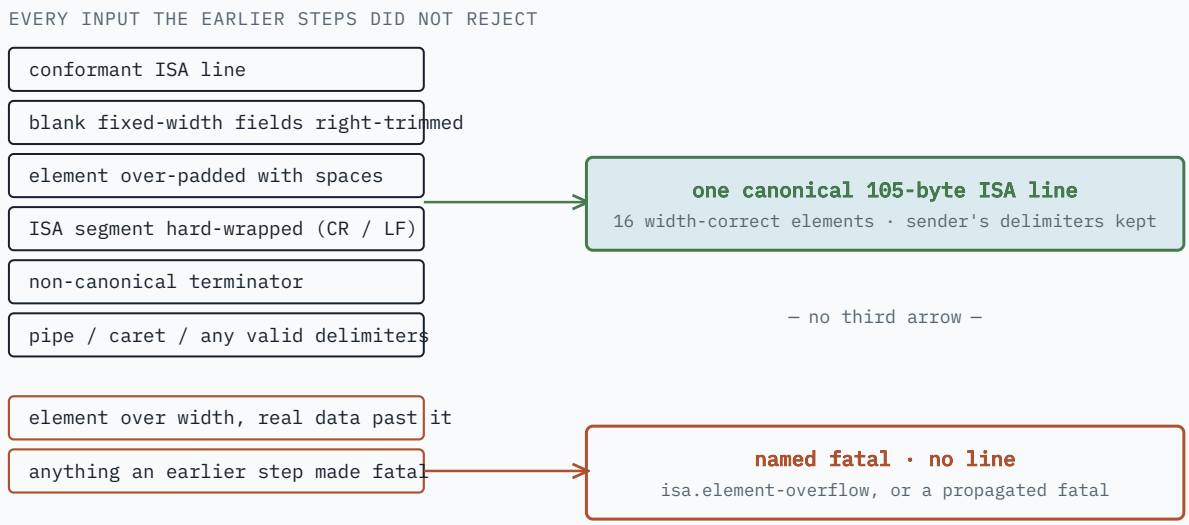
Reconstruction is a **total function** over the inputs the earlier steps did not reject. For any run whose delimiter parse raised no fatal:

- the run splits into exactly **16 elements** — the delimiter parse already guaranteed it
- each element is brought to its **fixed width**
- the line reassembles to exactly **105 bytes**

Every non-fatal input maps to exactly one canonical ISA line. Every other input maps to a fatal diagnostic that says why. **There is no “reconstructed but maybe**

wrong.” No silent truncation, no partial line. That closed set is the point of this step, and §5 walks every branch of it.

FIG. 1 — THE CLOSED SET OF OUTCOMES



Six input classes on the left are repaired to the same canonical line; two are refused by name. **Nothing lands between the two boxes.** That is what “total function” means here — the output is either the one right answer or an explicit refusal, never a plausible-looking wrong one.

WHAT CHANGED

02 Why width stopped being load-bearing

An offset parser treats the 105-byte length as structure: it reads `ISA13` at bytes 89–98 because that is where the standard puts it. A sender who right-trimmed a blank `ISA02` from ten spaces to nothing shifts every later field left, and that parser is now reading `ISA13` out of the middle of `ISA12` and `ISA14`.

x12-tidy never read a field at an offset. The delimiter step took the line apart with a single `split` on the element separator, anchored on the guaranteed separator count — not on any byte position. So by the time reconstruction runs, the sixteen element *values* are already in hand, whatever length they arrived at.

FIG. 2 — OFFSET VS. THE PIECES ALREADY IN HAND

Same line, two readings. The offset parser's window has slid off the field it names. x12-tidy already has the field — it just needs to be the right length. **Width is no longer a fact to trust; it is an output requirement to satisfy.**

FROM **DELIMITERS.PY** — THE SPLIT HAPPENED ONCE, THERE

```
# IsaDecomposition, from split_isa_line
decomposition.elements   # (ISA01, ISA02, ..., ISA16) — raw, not width-checked
```

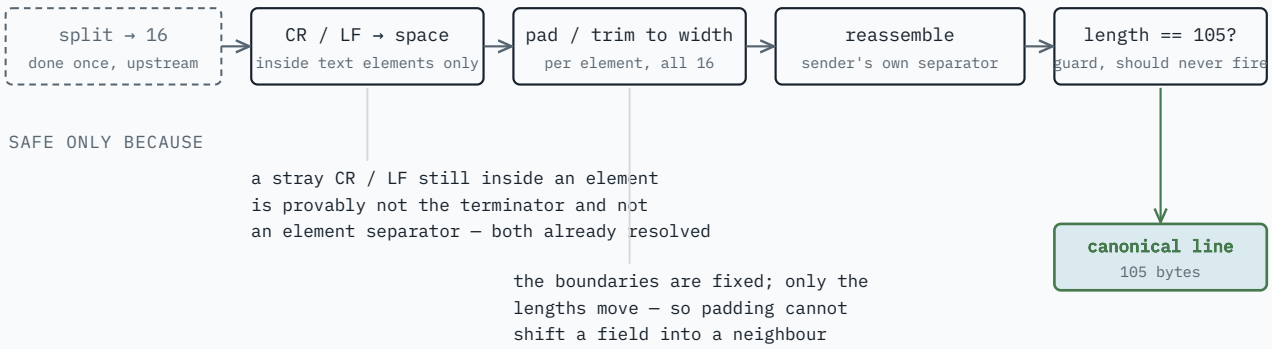
> Pad what is short, and the line the sender made unreadable becomes readable again.

THE CORRECTNESS ARGUMENT

03 The order of operations is the proof

Each transformation is safe *only* because of what has already happened. Reorder any two and it breaks.

FIG. 3 — THE PIPELINE, AND WHAT MAKES EACH STEP SAFE



SAFE ONLY BECAUSE

a stray CR / LF still inside an element
is provably not the terminator and not
an element separator — both already resolved

the boundaries are fixed; only the
lengths move — so padding cannot
shift a field into a neighbour

The greyed first stage is the delimiter step's work, consumed directly — reconstruction never splits again. Every stage after it depends on the one before. **Order is the correctness argument, not a style choice.**

3.1 Split into 16 — already done, once

The delimiter parse split the run on the element separator to *find* the delimiters, and it kept the pieces. Reconstruction consumes `decomposition.elements` directly. One split, one source of truth for where the element boundaries are.

3.2 Carriage returns inside an element → spaces — and only now

Some senders hard-wrap the ISA segment across lines, leaving a `\r` or `\n` inside an element value. Turning those into spaces is correct — a wrapped field is really space padding a line break mangled — **but it is only safe here.**

Earlier, a `\r` or `\n` could *be* the segment terminator (`\r\n`, or a bare `\r`), or in a pathological file a delimiter. Blank those out early and you destroy a real delimiter. By this point the terminator has been identified and split off, and the element separator has already done its work, so a `\r` / `\n` still sitting *inside* an element is provably neither.

Two elements are still excluded, by position, because their *value is a delimiter*: `ISA16` (always the component separator) and `ISA11` when it carries the repetition separator (version `00403`+). A sender is free to choose `\r` for either.

FROM RECONSTRUCT.PY

```
is_delimiter_element = index == 16 or (index == 11 and carries_repetition_separator)
if not is_delimiter_element and (b"\r" in value or b"\n" in value):
    value = value.replace(b"\r", b" ").replace(b"\n", b" ")   # isa.element-embedded-
```

3.3 Each element to its fixed width

The sixteen widths are fixed by the standard:

```
ISA01  2   ISA05  2   ISA09  6   ISA13  9
ISA02 10   ISA06 15   ISA10  4   ISA14  1
ISA03  2   ISA07  2   ISA11  1   ISA15  1
ISA04 10   ISA08 15   ISA12  5   ISA16  1
```

They sum to 86; with the `ISA` identifier (3) and the sixteen element separators (16) that is the canonical 105 bytes. Per element:

- **shorter than its width** → space-pad on the right. `isa.element-width`
- **longer, but only by trailing spaces** → trim. `isa.element-width`
- **longer, with real data past the width** → refuse (§4)

> `ISA.ELEMENT-WIDTH` IS AN ERROR, NOT A WARNING
>
> A padded value is unchanged in meaning — but until it is padded, the ISA line is not 105 bytes, and conventional VAN services and every fixed-offset parser downstream cannot read the interchange at all. That is not advisory.

3.4 Reassemble

`ISA` + the sixteen width-correct elements, joined on the sender's *own* element separator — unchanged, because any valid non-alphanumeric byte is

conformant. The segment terminator is likewise the sender's own byte: which character serves as a delimiter is the sender's choice, X12 does not dictate it, so a `\n` terminator stays `\n` and is never rewritten to `~`. It is not one of the 105 bytes — the reconstructed line carries none — and is returned alongside for the eventual whole-interchange rejoin. When the sender left the terminator out entirely x12-tidy refuses rather than guess `~` — a wrong terminator would break the split of every following segment. A final assertion checks the length is 105; it cannot fail (sixteen fixed-width elements plus the separators sum to 105 by construction), so it is a tripwire for a future change, not a reachable outcome.

THE ONE REFUSAL

04 The one thing reconstruction refuses

An element longer than its fixed width with **real, non-space data** in the overflow — a 17-character sender ID in `ISA06`'s 15-byte field:

```
...*ACMEWIDGETSCORP01*ZZ*...
        └ ISA06, 17 bytes, "01" past the width
```

The tool cannot know what the sender meant. Maybe an element separator was dropped and `ISA06` has swallowed part of `ISA07`. Maybe the sender simply overran the field. Truncating to 15 bytes to “fix” it would silently change an identifier; splitting it would invent a boundary. So it does neither: `isa.element-overflow`, fatal, with both possible causes named. This is the method's refusal rule — permissive parsing never invents.

EVERY BRANCH

05 The closed set of outcomes

INPUT	OUTCOME
blank fixed-width fields right-trimmed	padded back — `isa.element-width` per field
element over-padded with spaces	trimmed — `isa.element-width`
ISA segment hard-wrapped (`\r` / `\n` in a text element)	line breaks → spaces — `isa.element-embedded-newline` , then re-measured
non- `~` terminator (`\n` , bare `\r`)	**kept as-is, no finding** — which byte ends a segment is the sender's choice, not a deviation
trailing bytes after the terminator (`~\r\n` , `~`)	the real terminator kept; the trailing `\r\n` /spaces stripped (`isa.trailing-newline` / `isa.trailing-junk`)
terminator omitted entirely (GS follows ISA16)	`~` supplied — nothing to preserve — `isa.segment-terminator-stripped`
pipe / caret / any valid delimiters	kept as-is; the delimiters are the sender's choice
conformant ISA line	returned unchanged — `was_clean` — the one row with nothing to repair
element over width with real data	`fatal — isa.element-overflow`
anything the delimiter step made fatal	`propagated; no line`

No row produces a line that is not exactly 105 conformant bytes. That is what “total function” means here.

THE NEXT RUNG

06 Where value validation takes over

Reconstruction validates and repairs *structure* — widths, delimiters, the terminator, the length. It does not judge element *values*: whether `ISA05` is a real interchange-ID qualifier, `ISA09` a real date, `ISA15` a legal usage indicator, or whether `ISA13` matches the `IEA02` that closes the interchange. That is the next phase — the GS/ST envelope and control-number checks — and it is a genuine backstop, not an afterthought.

See the ladder: shape, then delimiters, then structure here, then values.

PROVING IT

07 Reconstruct, then re-parse

"Clean" is not a flag someone sets. It is a property you can test: **feed a reconstructed line back through the whole pipeline and nothing this step repairs is still outstanding.**

THE ROUND-TRIP ACCEPTANCE TEST

```
for every non-terminal corpus input:
    first    = clean_isa_line(dirty)
    reparsed = clean_isa_line(wrap(first.isa_line))

    assert reparsed.isa_line == first.isa_line          # a fixed point
    assert reparsed.elements == first.elements
    assert no isa.element-* / isa.leading-bytes / isa.trailing-*
           in reparsed.diagnostics                      # nothing left to repair
```

The reconstructed line is a **fixed point** — cleaning it again returns the identical bytes. Value-level findings (`isa.version-unrecognized` , `isa.isa11-not-standards-id`) are out of scope for this step and are allowed to survive a round trip; a structural one is not.

The corpus behind this is every case from the two earlier steps plus a “carriage return anywhere in the line” family, truncation, and byte-mutation fuzz — the same partition the decompose sweep proves: clean refusal, or lossless account. Never a wrong answer.

THE PATTERN

Once you can trust the delimiters, stop treating width as structure and start treating it as an output requirement. Apply each repair only after the step that makes it safe — order is the proof. Refuse the one case where the sender's intent is genuinely unknowable, and name both readings. Then show the result is total: enumerate every input class and check each lands on a canonical line or a named fatal, with nothing in between.