TidyEDI

X12 EDI · METHOD

The x12-tidy Method

Don't derive the four X12 delimiters from the structure. Earn them first — from the weakest facts that pin them down — and then the structure is ordinary parsing plus repair.

The spine x12-tidy / isa ~1,100 words PDF Read this first

The engineering notes were written as related-but-separate lessons — locating the ISA line, reading its delimiters, rebuilding it. The through-line is a single architectural idea, and it was implicit. This note makes it explicit. Read it once; each note then picks up one act of the same argument.

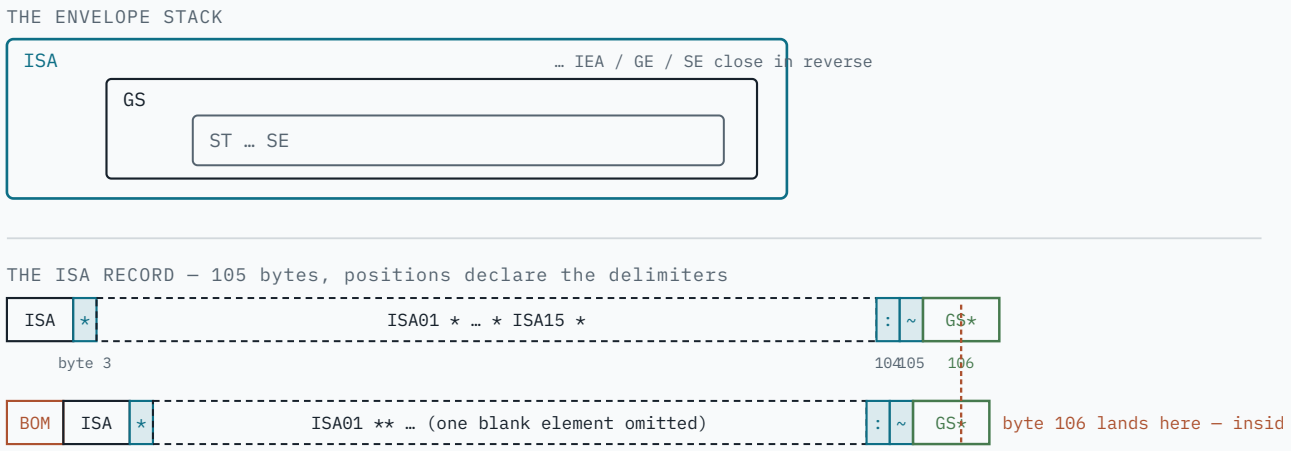
THE SETUP

01 The problem, stated once

An ANSI X12 interchange is a stack of envelopes: `ISA` wraps one or more `GS` functional groups, each wrapping `ST` transaction sets, closed in reverse by `SE`, `GE`, `IEA`. The first segment, `ISA`, is a fixed 105-byte record whose byte positions *declare the delimiters* for everything after it — element separator at byte 3, component separator at byte 104, segment terminator at byte 105, and (version `00403`+) the repetition separator in `ISA11`.

The standard is strict about this, so the obvious reader is offset arithmetic: slice the delimiters out at 3, 104, 105; check `GS` sits at 106; parse the rest. It works on conformant files and fails on a large share of real ones, because the senders x12-tidy exists for **strip empty elements, prepend bytes, append newlines, and re-encode**. Three bytes of BOM and one omitted element are enough to put byte 106 in the middle of a field. The byte positions are not an invariant.

FIG. 1 — THE ENVELOPE STACK, AND THE RECORD THAT DECLARES ITS PUNCTUATION



The dashed line is byte 106. In the conformant record it falls exactly on `GS`. Prepend three bytes of BOM and drop one blank element and it falls two fields short. **An offset parser has already failed here** — before it has read a single value.

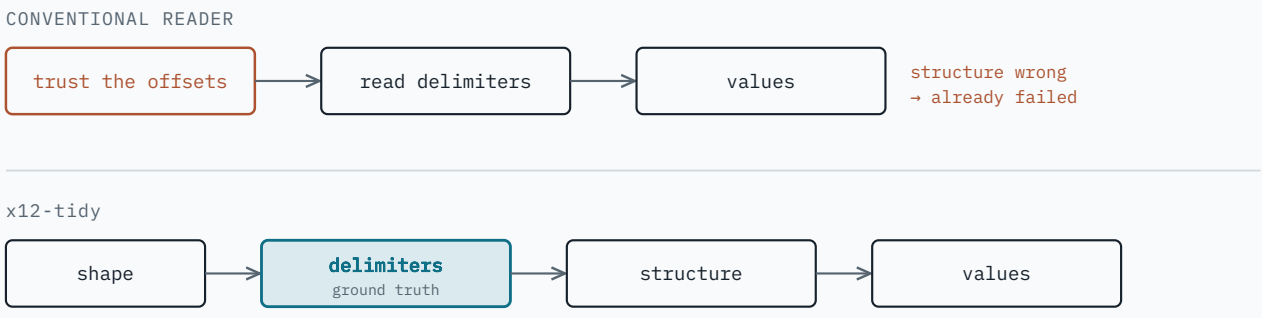
THE IDEA

02 The inversion

Every conventional X12 reader derives the delimiters *from* the structure — it trusts the offsets, reads the bytes there, and moves on. When the structure is wrong, it has already failed.

x12-tidy goes the other way. **The only things it must be certain of are the four delimiters.** So it earns those first, from the weakest structural facts that still pin them down — never from an offset or a width — and only then looks at anything else. Once the delimiters are known and nothing fatal was found, they are ground truth: every later step (split the elements, check the widths, rebuild the line, walk the body) is ordinary parsing plus repair.

FIG. 2 — TWO DIRECTIONS THROUGH THE SAME LINE



Same bytes, opposite order. The conventional reader needs the structure to be right before it can read the punctuation. x12-tidy needs only the punctuation — and gets it from facts a broken structure leaves intact. **That is the whole method; everything else is a consequence.**

EACH PART OF THE PARSE

03 Earn the delimiters first

This reframes what each part of the parse is *for*:

Locating the ISA line is not the goal — it is the precondition for a clean delimiter parse. Its minimum bar (begins with `ISA`, ends immediately before `GS` + the element separator, holds *exactly* 16 element separators) is exactly the set of facts sufficient to guarantee the delimiters can be read and the line split into 16 elements without ambiguity. No more, no less. A run that fails the bar is fatal and terminal — not because locating failed, but because the delimiters could not be trusted out of it.

Reading the delimiters anchors on the one offset that cannot move (byte 3, before the first variable-width field) and the one boundary the sender cannot shift (the `GS` header), and recovers the other three from a single `split`. A finding here is fatal only if a delimiter the whole interchange needs is unusable.

Reconstruction is where the payoff lands: with the delimiters trusted, element width stops being load-bearing, and a right-trimmed blank field — fatal to an offset parser —

is just padding to restore.

THE SHAPE OF THE PIPELINE

04 The ladder: shape → delimiters → structure → values

Each phase validates exactly one more thing and hands the next a stronger guarantee. Nothing checks a property the phase below it has not already established.

PHASE	ESTABLISHES	HANDS FORWARD
locate	the run has the *shape* of an ISA line (identifier, `GS` boundary, 16 separators)	a run the delimiters can be parsed from
delimiters	the four delimiter bytes, and which are usable	ground-truth delimiters
structure	16 elements at fixed width, one canonical 105-byte line, the sender's delimiters kept	a conformant ISA line
values (later)	`ISA05` is a real qualifier, `ISA09` a real date, `ISA13` matches `IEA02`	a validated interchange

A note that says “that is the next step's problem” is pointing down this ladder.

THE REFUSAL RULE

05 Never guess the sender's intent

Every terminal decision in the system is the same move. The tool *could* guess, but a wrong guess silently corrupts an identifier — a sender or receiver ID, a control number — so instead it refuses and names the fault:

- **more than 16 element separators** → the line has no unambiguous parse (a separator inside `ISA06` / `ISA08` data, or a false `GS` match). Terminal.

- **an element longer than its width with real data in the overflow** → a dropped element separator merged two fields, *or* the sender overran one. Both are plausible; the tool will not pick. Terminal.

- **`ISA11` holding something other than `U` on a pre-`00403` version** → a wrong value in an informational field, not a repetition separator the sender "meant." Reported, not used.

Permissive parsing tolerates a lot. It never invents.

PROVING IT

06 Prove it against hostile input

Each step ships an adversarial sweep — tens to hundreds of thousands of mutated, truncated, mis-encoded inputs — checked against a single invariant: no crash, and either a clean refusal (a fatal diagnostic, no output) or a result that satisfies the phase's contract. There is no third outcome — no silent wrong answer, no partial parse. The notes each end with the sweep that backs them.

THE PATTERN

Find the one thing you must be certain of. Earn it from the weakest assumptions that pin it down, anchored on what the adversary cannot move. Treat it as ground truth from then on. Build the rest as a ladder where each rung only checks what the rung below already guaranteed. Refuse — loudly, by name — rather than guess. And prove the whole thing against inputs written to break it.