

The HL7 report (b3u.dev/docs/ccs_java_hl7.pdf) covered the clinical message; the X12 report (b3u.dev/docs/ccs_x12.pdf) covered the billing message; this one covers **the record the mainframe holds between them**. Same platform, same claim:

ONE PARSE. FIVE RUNTIMES. NO RE-PARSING AT ANY LANGUAGE BOUNDARY.

Here the artifact parsed once is the **copybook itself**: compiled by a grammar-generated frontend into an SCB — a *Syntax-Controlled Binary*, the parsed schema persisted as bytes — which Java, Python, JavaScript, Rust, and C++ each read natively and reproduce **byte-identically**. The schema stops being N re-implementations and becomes one compiled artifact with a cross-runtime oracle. On top of it, this report demonstrates the layer where mainframe data work actually lives or dies: a record decode — EBCDIC, COMP-3 packed decimal, zoned overpunch, REDEFINES — proven identical in all five runtimes against a hand-computed oracle.

Ground rules: every copybook in the evidence corpus is public (Apache-2.0); all record data is synthetic; no customer artifact of any kind appears in this work.

2. CCS fundamentals, in brief

(The full treatment — the toolchain, the five SCB APIs, in-process compilation from Java, JVM/C++ benchmarks — is §2–§3 of the published HL7 report. The minimum for this report:)

CCS is a grammar-driven data platform, protected by US Patent 8,464,232. You write a grammar in SGDL; the b3u.dev toolchain generates the format's parser/compiler frontend (CLI + shared library with a small C ABI) and the grammar's keyword tables in five languages. Whatever the frontend parses becomes an SCB, readable by five native SCB APIs (C++, Python, JavaScript, Rust, Java) that are held together by a standing byte-identity oracle. Performance anchors measured on this substrate: the binary form loads 20–30× faster than its text form in C++ and lands the JVM load at about one Jackson parse.

3. Why the data layer — and deliberately not translation

Legacy modernization is three layers wearing one name, and we chose ours on an honest capability analysis:

layer	what it is	grammar-shaped?
program logic	COBOL/PL-I source translated to Java, C#, Go	NO – dialects, embedded EXEC, parsing is ~15% of the job
data	copybooks, record layouts	YES – small, stable, read by everything
discovery	JCL, job graphs, lineage	YES – bounded

The top row is where the crowd is — IBM watsonx Code Assistant for Z, AWS Blu Age, the established conversion vendors, and a funded wave of AI translation startups. A general COBOL front end is years of dialect work that still loses on features, and we are not building one. The other two rows are compiler problems: a copybook is a schema in a small, stable declarative language, and turning schemas into cross-language readers is exactly the machinery CCS already is.

One more asymmetry matters in 2026: translation is commoditising fast under LLMs, but nobody can prove the output is correct. The scarce good in this market is **verification** — byte-level oracles that a translated or migrated system still reads the estate's data exactly. Everything in §4 is built in that spirit: every claim has an oracle, and the oracles are shown.

Unlike X12 (whose licensing chapter the second report devotes itself to), this material is unencumbered end to end: COBOL's record description language is open knowledge, and the evidence corpus is Apache-2.0 code from public repositories. So this report can do what the X12 one could not — print the grammar, the corpus census, and the whole pipeline.

4. The demonstration

4.1 The corpus verdict first — real copybooks, no hacks

Before any demo was built, the grammar faced a falsifier: does it parse *real* copybooks, with zero per-file special-casing? The corpus: 66 public copybooks from two origins, both Apache-2.0 —

AWS CardDemo (30 files of real application source) and the test fixtures of ABSA's Cobrix (36 files) — covering COMP-3 (13 files), REDEFINES (28), and OCCURS DEPENDING ON (8).

The verdict: **27 of 27 real-application record descriptions parse; 47 of 52 distinct contents overall; zero per-file hacks.** Every failure was classified from evidence, and none is a grammar gap — the residue is input that is not a COBOL record description in the first place (vendor-specific fixture content). The corpus harness runs as the demo gate's first leg to this day, and the buckets are pinned: a moved count fails the gate.

4.2 The grammar (abridged)

The core of the record-description grammar — the full version adds OCCURS (with DEPENDING ON), VALUE/VALUES, SIGN, SYNC, JUSTIFIED, BLANK, 88-levels, and the usage spellings:

```
(copybook
  (copybook ::= entries )
  (entries  ::= { entry } )

  -- 05 CUST-BALANCE PIC "S9(9)V99" COMP-3 .
  (entry    ::= level dataName clauses )
  (level    ::= integerToken )
  (dataName ::= identifier | mettaSymbol )
  (clauses  ::= { clause } '.' )

  (clause ::= redefinesClause | picClause | usageClause | ... )

  (redefinesClause ::= 'REDEFINES' dataName )
  (picClause       ::= 'PIC' picBody )
  (picBody         ::= [ 'IS' ] picString )
  (picString       ::= stringToken )
  (usageClause     ::= 'USAGE' usageBody )
)
```

A lexical stage precedes the grammar, and the report states it plainly because it shapes the fidelity contract: COBOL copybooks are fixed-format (columns matter, * in the indicator column is a comment), COBOL words contain hyphens, and the PICTURE character-string is one lexical unit whose alphabet collides with ordinary punctuation. A small documented normalizer handles exactly those four facts and nothing else, producing a one-entry-per-line token stream the generated parser consumes:

```

01 DEMO-REC .
05 CUST-ID PIC "X(6)" .
05 AMOUNT PIC "S9(5)V99" COMP-3 .
05 COUNT-Z PIC "S9(3)" .
05 FLAGS REDEFINES COUNT-Z PIC "X(3)" .

```

Consequently the byte-stable canonical form is this normalized form (and the SCB's decompilation), not the raw fixed-format source — unlike the HL7/X12 demos, canonical == raw input bytes is *not* claimed here, and §6.2 records it as a limit of the fidelity story.

4.3 The schema as a compiled artifact

Compiling that five-line copybook produces an SCB of 1,832 bytes — the parsed schema. From it, every runtime derives the same layout:

```

1 DEMO-REC GROUP offset=0 len=13
5 CUST-ID PIC X(6) DISPLAY offset=0 len=6
5 AMOUNT PIC S9(5)V99 COMP-3 offset=6 len=4 scale=2
5 COUNT-Z PIC S9(3) DISPLAY offset=10 len=3
5 FLAGS REDEFINES COUNT-Z PIC X(3) DISPLAY offset=10 len=3

```

Note what the parser already enforced structurally, and what the layout adds: COMP-3 length arithmetic (7 digits → 4 bytes), the REDEFINES view sharing offset 10 with the field it redefines, and the implied decimal scale.

4.4 The decode witness — built the careful way

Numeric fidelity is where mainframe data work lives or dies: one wrong sign convention silently corrupts financial data. So the decode demonstration is deliberately a **witness, not a library** — one record exercising the encodings that actually hurt, decoded against an oracle computed by hand and cross-checked mechanically.

The 13-byte record:

offset	bytes	field	encoding	value
0-5	C1C2 F1F2F3F4	CUST-ID	EBCDIC cp037 text	"AB1234"
6-9	12 34 56 7D	AMOUNT	COMP-3 packed, sign nibble D	-12345.67
10-12	F0 F4 C2	COUNT-Z	zoned decimal,	+42

10-12	(same bytes)	FLAGS	C-zone overpunch REDEFINES, read as cp037 text	"04B"
-------	--------------	-------	--	-------

The last row is the point: the same three bytes are simultaneously +42 (zoned, the overpunched 0xC2 carrying digit 2 with a positive sign) and the text "04B" (0xC2 is EBCDIC 'B') — REDEFINES made byte-real. All five runtimes produce:

```
CUST - ID=AB1234 | AMOUNT=- 12345 . 67 | COUNT - Z=42 | FLAGS=04B
```

The oracle discipline: the expected bytes were derived by hand from the documented encodings, and the fixture generator re-derives them independently through a second route, refusing to write the fixtures if the two derivations disagree. The gate then adds the red that makes the oracle meaningful: flipping one byte inside the packed digits changes the decode — measured, so a passing comparison is evidence, not ceremony.

4.5 The five runtimes — and what was NOT written

The canonical layer cost **zero new code**: each substrate's existing generic SCB decompiler reproduces the compiler's own decompilation of the schema byte-identically — Java, Python, JavaScript, and Rust, with the C++ compiler's output as the reference. That is the platform's standing five-way oracle simply absorbing a new grammar, and it is the strongest maintenance statement in this report.

What was written per runtime is the value layer: the schema walk (~150 lines each, resolving every grammar symbol by name from the generated keyword table), the layout arithmetic, and the decode — five implementations of one written spec, every one agreeing with the hand oracle on the first gate run.

The Java pipeline additionally compiles the copybook **in-process**: the generated frontend ships as a shared library exporting only a small C ABI, Java calls it through the FFM API, and the returned SCB is byte-identical to what the CLI wrote to disk — proven in the battery, along with the obfuscation cycle (XTEA; ECIES with AES-GCM; two independent ECIES implementations cross-decoding both ways; a wrong key refused at the authentication tag) from Java, Python, and JavaScript.

4.6 The gate

One command runs everything — 24 checks, all green: the corpus buckets pinned, the pipeline reproducing the committed SCB byte-identically, the shared library's export surface verified with nm, the Java battery, the four runtime parities, and three red witnesses (an unterminated entry is caught; a corrupted SCB does not silently walk to an identical schema; a flipped record byte changes the decode).

5. The mainframe data bridge on CCS

The deployment pattern this enables:

```
the estate's copybooks (yours, unchanged)
  | normalize + compile ONCE (CLI or in-process .so)
  ▼
schema SCBs – versioned, diffable, obfuscatable artifacts
  |
  ├── JVM      Spark/ETL jobs read the schema natively
  ├── Python   analytics/data-science, same artifact
  ├── JS       ops consoles and viewers, same artifact
  ├── Rust/C++ feed processors and gateways, same artifact
  |
  └── each consumer decodes records AGAINST THE SAME SCHEMA
      – the layout arithmetic can no longer drift per language
```

What changes for an estate mid-migration: the copybook stops being documentation that N teams re-implement and becomes a compiled, versioned artifact with a mechanical cross-runtime agreement check. When a migration team asks "does the new system read this record exactly as the old one did?", the answer becomes a byte-level comparison against a shared schema, not a meeting.

What this is not. It is not a COBOL translator and does not compete with one — it is the data substrate underneath a migration, whoever performs it. Nor is it a general decode library today: §6.2 is the scope statement.

6. The good, the bad, and the ugly

6.1 The good

The schema becomes a compiled artifact with an oracle. One copybook, one parse, five runtimes in byte-level agreement — enforced by a gate, not a convention. The canonical layer required zero new code per runtime: the platform's existing decompilers absorbed the new grammar unchanged, which is the concrete form of "adding a format is cheap here."

The grammar faced reality before it faced a slide deck. 27/27 real-application record descriptions from public corpora, zero per-file hacks, failures classified in the open. The corpus harness is a standing gate, not a one-time claim.

The careful-decode culture is the product posture. Where the HL7 and X12 reports proved message fidelity, this one proves the thing mainframe estates actually fear — sign conventions and packed digits — with double-derived oracles and deliberate reds. Careful beats hasty in exactly the layer where hasty corrupts money.

Fully open evidence. Unlike X12's licensed specification world, everything here — the language rules, the corpus, the grammar, the pipeline — could be printed in this report.

6.2 The bad (current limits — real, and fixable)

1. **The decode layer is a witness, not a library.** It covers exactly what the demo record exercises: PIC X/9/S/V with (n) repeats, COMP-3, zoned DISPLAY with overpunch, flat REDEFINES, code page 037. Not yet: OCCURS (fixed or DEPENDING ON), SYNC alignment, COMP-1/2/4/5 and BINARY, editing pictures, other EBCDIC code pages, nested-group offset arithmetic. That is the enumerated next phase, de-risked but not delivered.
2. **The grammar scope has named gaps.** COPY/REPLACING, 66-level RENAMES, and a handful of clause forms are recorded exclusions — each is a documented decision, not a surprise, but a real estate will hit some of them.
3. **Canonical form is the normalized stream, not the raw source.** Fixed-format columns and comments are consumed by the documented lexical stage; byte-identity holds from the normalized form onward. If your compliance story requires raw-source fidelity, that is a different (and harder) contract.
4. **Record data itself is not yet an SCB.** Today the schema is the compiled artifact and records are decoded against it; the record-as-SCB path (parse the data file once, too) is roadmap.
5. **One code page, demo-scope.** cp037 only, via a shared table; the code-page matrix (037/500/1047, euro variants) is an open engineering question we have deliberately not

guessed at.

6.3 The ugly (the hard truths — for us and for you)

1. **Numeric fidelity is the whole game, and it is unforgiving.** A decode layer that is 99% right is a data-corruption engine with good marketing. That is why this report ships a small witness with layered oracles instead of a broad library with asserted correctness — and why the broad library will grow gate by gate, encoding by encoding, with the same discipline.
 2. **Your copybooks are your property, and they describe regulated data.** No customer artifact enters our systems or our evidence corpus — engagements run on your side or on synthetic material, and the public corpus is how we prove the machinery without touching anyone's estate. This slows demos down. It is the correct trade.
 3. **The famous problem is not the one we solve.** If you need COBOL program logic translated, that market is crowded with large vendors and we are deliberately not in it. We make the data layer solid underneath whichever translation or rehosting path you choose — and underneath the verification that it worked.
 4. **CCS is proprietary and patented.** One vendor, commercial terms, not an OSS parser you can fork. Three reports now make the same honest case: what you buy is a maintained single source of truth with a cross-runtime oracle, and you should weigh that trade with eyes open.
-

7. Working with b3u.dev

The engagement template is deliberately small:

1. **Scope one record family** — the copybooks behind one dataset or interface of your estate. We compile them into schema artifacts and reference readers in your stack's languages.
2. **You provide the oracle:** byte-level expectations from your existing system, on synthetic or de-identified records. Acceptance is your expectations passing plus the five-way byte-identity oracle.
3. **You get the component set:** the compiled schema artifacts, the generated frontend (CLI + C-ABI shared library), the per-grammar keyword tables in five languages, and the reference readers — the same deliverables §4 printed.

Licensing is subscription-based through b3u.dev, with production and OEM/embedding terms under a separate written agreement. The demo in this report runs end to end in one command on

delivered artifacts — ask for a walkthrough. If your estate speaks HL7 or X12 at its edges, read the sibling reports: three formats, one platform, one discipline.

Contact: support@b3u.dev

© 2026 BET THREE LLC. CCS technology is protected by US Patent 8,464,232. Code listings and transcripts are excerpts of the working demonstration, lightly edited for publication (internal build references removed). The evidence corpus consists of publicly available Apache-2.0 material: AWS CardDemo application copybooks and ABSA Cobrix test fixtures, unmodified, credited to their authors. All record data shown is synthetic; no customer artifacts, no PHI, and no real financial data were used anywhere in this work. IBM, watsonx, AWS, and other product names are the property of their respective owners.