

A read-only bridge into COBOL business logic

Portfolio case-study draft | Original contribution summary + proposed synthetic illustration

Original work supported by the sources

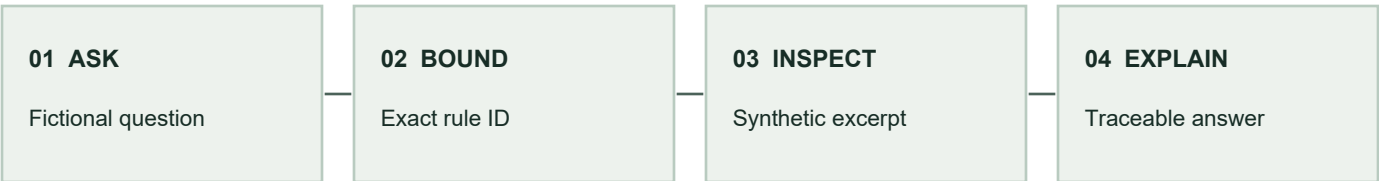
At Global Shop Solutions, I built a security-filtered, read-only Model Context Protocol (MCP) interface to COBOL business logic for QA, Support, and engineering. I also provided training and documentation. This is supported by my corrected October 7 résumé and the existing portfolio's high-level account.

The engineering problem was making established business behavior easier for colleagues to investigate while limiting what an assistant could access. My eight years of manufacturing ERP work, strongest in COBOL and VB.NET, provided the context for that work.

The sources establish the interface's purpose, read-only scope, sensitive-data filtering and intended internal teams. They do not establish its transport, tool schema, retrieval method, deployment, authentication model, or whether it inspected source, executed logic, or combined both. Those details are intentionally not presented as facts here.

No employer source code, customer records, screenshots of internal systems, or proprietary architecture are included.

Proposed illustration: one fictional rule, one bounded lookup



Proposed flow only. Not the original GSS architecture.

Everything in this section is newly authored synthetic material for this portfolio draft. This is one possible design illustration, not a reconstruction of the GSS implementation.

A QA colleague asks: “In this fictional example, what makes a work order require manual review?”

1. **Ask.** An assistant requests the known synthetic rule ID DEMO-REVIEW-RULE.
2. **Bound.** A proposed tool boundary accepts an exact allowlisted ID. Unknown IDs return a defined error; it accepts no paths, SQL, or executable commands.
3. **Inspect.** The toy adapter returns only a built-in synthetic COBOL excerpt and a synthetic source label. It neither executes COBOL nor reads a real codebase.
4. **Explain.** The reader can tie the explanation to the excerpt: REVIEW-REQUIRED is set to Y when HOLD-FLAG is Y or ITEM-COUNT exceeds REVIEW-LIMIT; otherwise it is N.

The visual version is in `mcp-cobol-bridge.html`; the standalone editable graphic is `proposed-flow.svg`.

Synthetic COBOL excerpt

```
*> SYNTHETIC PORTFOLIO EXAMPLE. NOT EMPLOYER CODE.
IF HOLD-FLAG = "Y" OR ITEM-COUNT > REVIEW-LIMIT
    MOVE "Y" TO REVIEW-REQUIRED
ELSE
    MOVE "N" TO REVIEW-REQUIRED
END-IF
```

The labels and rule are fictional. They do not represent GSS product behavior or a customer workflow.

Inspect the illustration locally

From this folder:

```
python illustrate_bridge.py
```

The script prints an allowed lookup and an unknown-ID rejection, then checks both outcomes. It uses only the Python standard library and hardcoded synthetic content. It makes no network requests and writes no data.

This is a deterministic tool-boundary illustration. It does **not** implement the MCP wire protocol, an AI assistant, authentication, production filtering, a COBOL runtime, or a complete security design. A built-in allowlist here demonstrates a small boundary; it does not validate the original system's security.

Results: what can and cannot be claimed

Evidence type	Supported statement
Original engineering work	Built a read-only, security-filtered MCP interface to COBOL business logic; provided training and documentation.
Local illustrative checks	The new toy script checks a known-ID response and an unknown-ID rejection. These checks apply only to the synthetic example.
Original engineering measurements	No comparable latency, accuracy, coverage, defect, or time-saving evidence supplied.
Business outcomes	No supported revenue, customer impact, support-volume reduction, or measured adoption claim.

The case study can describe the engineering contribution without inventing a business outcome. Internal audience names do not establish the size or frequency of adoption.

Questions needed for a faithful technical account

1. Did the original MCP tools retrieve source, call compiled COBOL logic, use documentation, or combine these approaches?
2. Which tool names, input/output schemas, transport and implementation language can be described publicly?
3. What information was filtered, at which boundary, and how was read-only behavior enforced? Can a nonproprietary example be shared?
4. How were identities, access scope and authorization handled? What was the deployment context?
5. How did the interface handle missing, stale or ambiguous results and tool failures?
6. Which design, implementation, tests and training work did Travis personally own, and which parts belonged to teammates?
7. What dated engineering evidence is available: regression cases, evaluation criteria, before/after task timings or verified defect examples? What do those measurements actually establish?
8. Is any measured usage or business outcome both available and approved for public disclosure? If not, keep those claims absent.

Provenance

Original-work summary: corrected October 7, 2026 résumé and existing portfolio account. All code, diagrams and data in this draft were newly authored synthetic illustrations on October 20, 2026.