EXAMPLE 01

From a business rule to running software

A team needs to reject orders above an agreed limit and explain each decision. This worked example follows one request through an actual AI authoring conversation, generated coordination and checked runtime results.

1 prompt · 4 application nodes authored · 7 decision cases passed

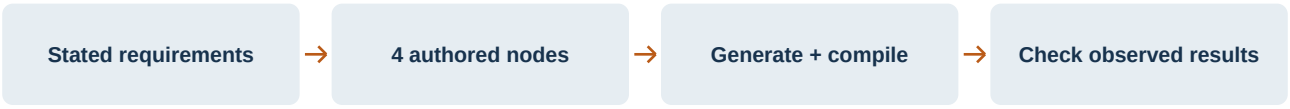


Illustration of the demonstrated workflow.

Observed result

In the captured run, 10 units at GBP 12 produce a GBP 120 order. With a GBP 100 limit it is rejected, including a second identical input. With a GBP 150 limit it is accepted. Equality is accepted. All seven decision cases match the stated rule; four invalid or overflowing inputs are refused.

Requirement	Observed check
Calculate quantity × unit price	10 × GBP 12 = GBP 120
Reject an order above its limit	GBP 120 / GBP 100: REJECTED
Accept at or below the limit	GBP 120 and GBP 150 limits: ACCEPTED
Repeat the same input consistently	Both identical inputs: REJECTED
Refuse invalid or overflowing input	Four invalid inputs refused

Recorded worked example with prepared SDK notes and local compiler setup.

Recorded session: gpt-6.1-sol configured with low reasoning effort; one completed CLI turn, three visible assistant messages and eight command actions.

Read the actual authoring conversation

Follow the five-step requirements check

Download the matching investigation bundle

To reopen the investigation in the Analyser, save the HTML reel and bundle in the same folder.

FOLLOW THE DEMONSTRATION

4-stage walkthrough

1. State the rule

Calculate quantity multiplied by price in integer pence. Accept a valid order at or below its limit, reject a higher value and refuse invalid input.

2. Author the application

With gpt-6.1-sol configured at low reasoning effort, the recorded session authored four application nodes plus an input type, generator and verifier.

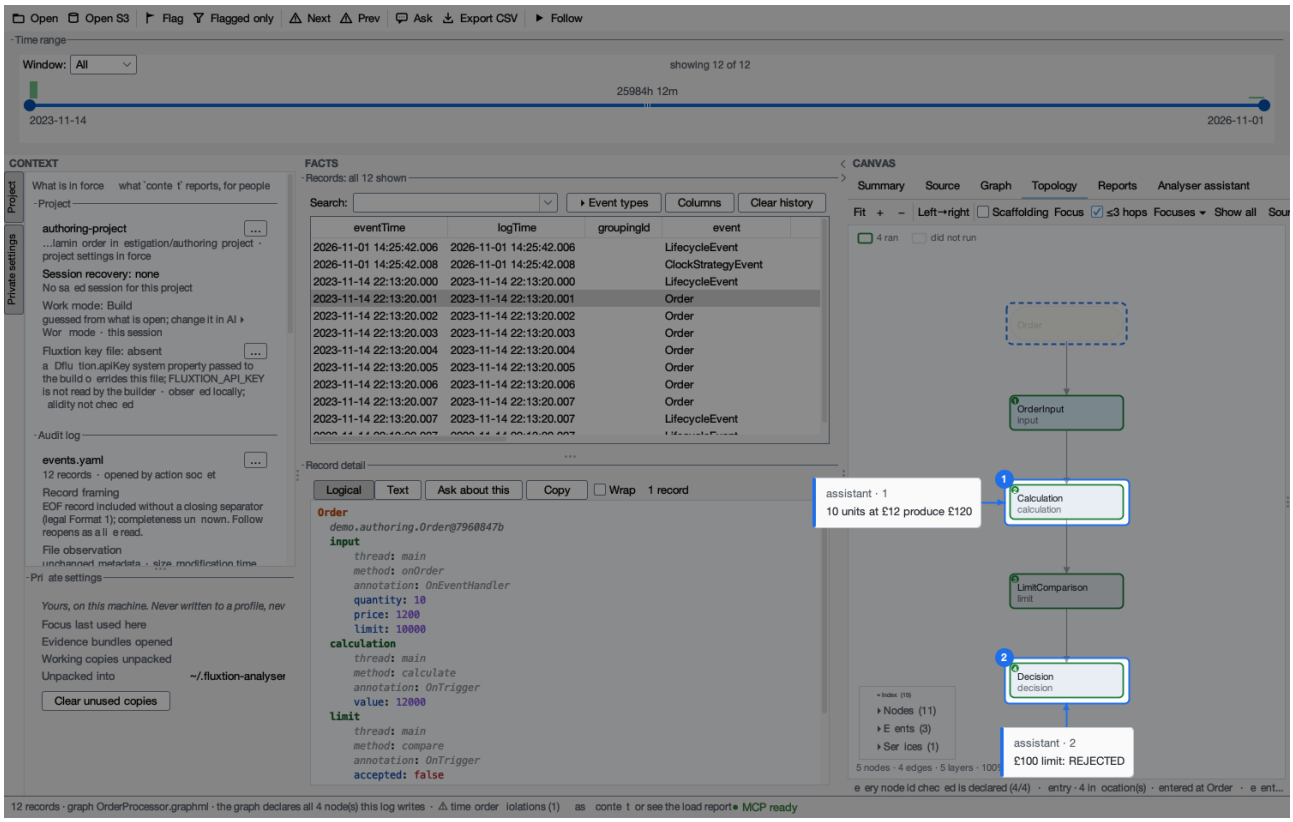
3. Generate and run

The assistant invoked the prepared local compiler helper. Fluxtion emitted Java and GraphML, then the runtime-only verification checked seven decisions and four invalid constructions.

4. Check the evidence separately

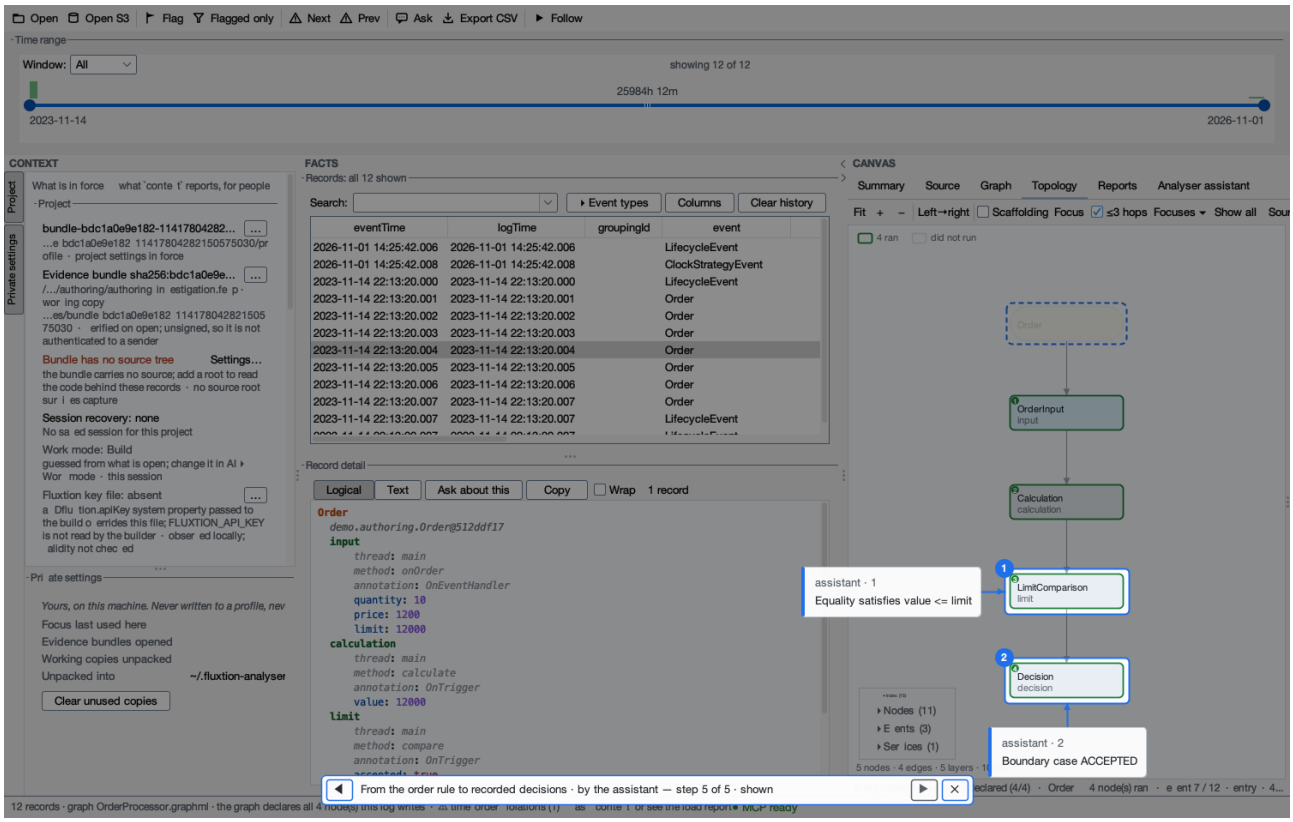
Reviewer-authored checks independently exercised nine valid cases with exact audit facts, five invalid constructions and a subsequent valid event. The original seven captured records were also checked against the requirements.

The newly authored application connects four application nodes to the recorded order checks.



CHECK THE STATED REQUIREMENTS

An order at its limit is accepted in the reopened investigation.



The saved investigation follows the calculated amount, above-limit rejection, repeated input, a higher limit and equality acceptance. These are the original seven valid trial events. The separate native verifiers establish invalid-input refusal before dispatch and the additional reviewer checks.

An evidence bundle is a portable investigation created by the Analyser from selected records, the graph and a saved walkthrough. A person and their AI assistant guide the investigation. The bundle can be reopened; the reel and PDF are for reading and sharing.

What is an evidence bundle?

Why this matters

The request, visible conversation, application graph and test outcomes remain available for review. A reviewer can compare the intended rule with what the generated application actually did, including repeated input and boundary cases.

Apply it to your process

A pilot could replace this small rule with one representative approval or operational control. Your team supplies examples, exceptions and acceptance criteria; together we assess the implementation and its evidence before deciding whether to adopt, extend or stop.

Discuss a five-stage pilot

TECHNICAL APPENDIX

Reproduce this result

Unzip this example's standalone source package and run from its authoring-session directory with Python 3, JDK 21 and the separately supplied reviewed runtime:

```
python3 run.py \
  --runtime /path/to/fluxtion-runtime-1.1.3.jar \
  --out results
```

The runner verifies the runtime and original trial manifest, compiles the actual generated Java and authored nodes, then executes the model's verifier and separate reviewer checks. The original decision CSV must match. No model or generation request is made.

Source and reproduction package for this example

This is the actual fresh processor from the recorded trial. Four application nodes and their definition were authored without a previous order implementation; Fluxtion generated the coordination. The curated Java was rebuilt after its owner-authorized demo licence notices were applied.

Runtime prerequisite: com.telamin.fluxtion:fluxtion-runtime:1.1.3. Download the JAR without an account; the runner checks its SHA-256. The source ZIP includes RUNTIME-PREREQUISITES.txt with the complete download command.

Download runtime JAR | Published POM: declares GNU AGPL version 3, separate from Apache 2.0 demo code.

SHA-256: 2bd45eb62409403422d5a676a7ad177f1984aa80dc399c20ceb6b467eab4b775

Detailed evidence

decisions.csv

summary.txt

session-receipt.json

events.yaml

original-evidence-review.json

manifest.json

Open this application's evidence bundle

Capture method and reopen instructions

Actual Analyser tool requests and responses

All exported PDFs, screenshots and verification receipts

Scope and interpretation

One prepared authoring demonstration, not a benchmark or general success-rate estimate. Generic SDK notes, a compiler helper and cached dependencies were supplied; no previous order implementation was provided.

One completed CLI turn contains multiple visible messages and command actions. The number of underlying model requests is not exposed. The receipt's elapsed time excludes prepared setup and is not a total-effort measure or timing guarantee.

The model-authored verifier checks seven valid cases and four invalid constructions. The separate reviewer checks nine valid cases, five invalid constructions and recovery. These establish the tested requirements, not universal correctness or human acceptance.

The original audit contains lifecycle wall time before the application clock is configured. Compare record values and event order rather than treating these mixed timestamps as a production chronology.

The subsequent Analyser capture used its isolated REST action socket with hosted OpenAI inference. Its bundle contains the original trial audit and graph, not the additional reviewer events or source tree. It was verified and reopened on the same machine. Ordinary member hashes do not authenticate origin or deployed-binary identity.

Demo code is Apache 2.0 under owner authorization dated 1 November 2026. Runtime, third-party and general customer-output terms remain separate. Source and artifact hashes compare bytes; they do not authenticate a sender or certify correctness.

Deployment and data flows | Applicable product rights