EXAMPLE 06

Same captured inputs. Reproduced behaviour. Visible differences.

An investigation explains an observed decision. Replay asks a different question: can the application reproduce that behaviour from the captured events and starting conditions? This demonstration records a live synthetic host session, replays its journal, then runs the same inputs through a changed rule.

16 / 16 host observations match · 9 / 9 order audit records match · 2 intentional rule differences

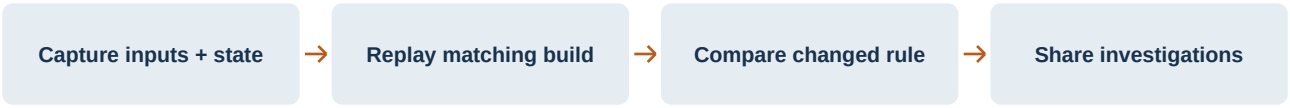


Illustration of the demonstrated workflow.

The problem

A sequence of orders changes its limits, stock and dependency availability over time. Checking isolated inputs is not enough: the order of updates and the starting conditions matter. A reviewer needs to distinguish changes caused by the code from changes caused by the environment.

What changed

The host saves each command before dispatch and captures its initial state. The original input producer is then deleted. Fresh JVMs replay only that saved journal and state. All 16 host observations and all nine order audit records match. Changing the limit rule from “at or below” to “below” rejects exactly the two equality orders; the other observations retain their expected behaviour.

MEASURED RESULTS

Observed result

The host saves each command before dispatch and captures its initial state. The original input producer is then deleted. Fresh JVMs replay only that saved journal and state. All 16 host observations and all nine order audit records match. Changing the limit rule from “at or below” to “below” rejects exactly the two equality orders; the other observations retain their expected behaviour.

Case or component	Recorded result
Captured inputs	16 commands: nine orders and seven host-state updates
Starting conditions	Fresh processor; explicit limit, stock, risk availability and clock
Same-build host observations	16 of 16 match
Resolved order inputs	9 of 9 match
Selected order audit records	9 of 9 match, including recorded node observations and injected clocks
Original execution	8 decision publications and one synthetic dependency exception
Changed rule	2 equality decisions change from ACCEPTED to REJECTED_LIMIT
Negative controls	Different state/order change behaviour; malformed state, sequence, overflow and clock mismatch are refused before execution

A synthetic engineering demonstration, with actual runs and saved investigations. Not a customer incident or a production-readiness claim.

Inspect the changed-rule investigation

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

5-stage walkthrough

1. Record the session

Save actual incoming commands and the explicit starting-state contract.

2. Reproduce it

Start a fresh application process using the saved inputs.

3. Compare the records

Check exact host observations and selected audit records.

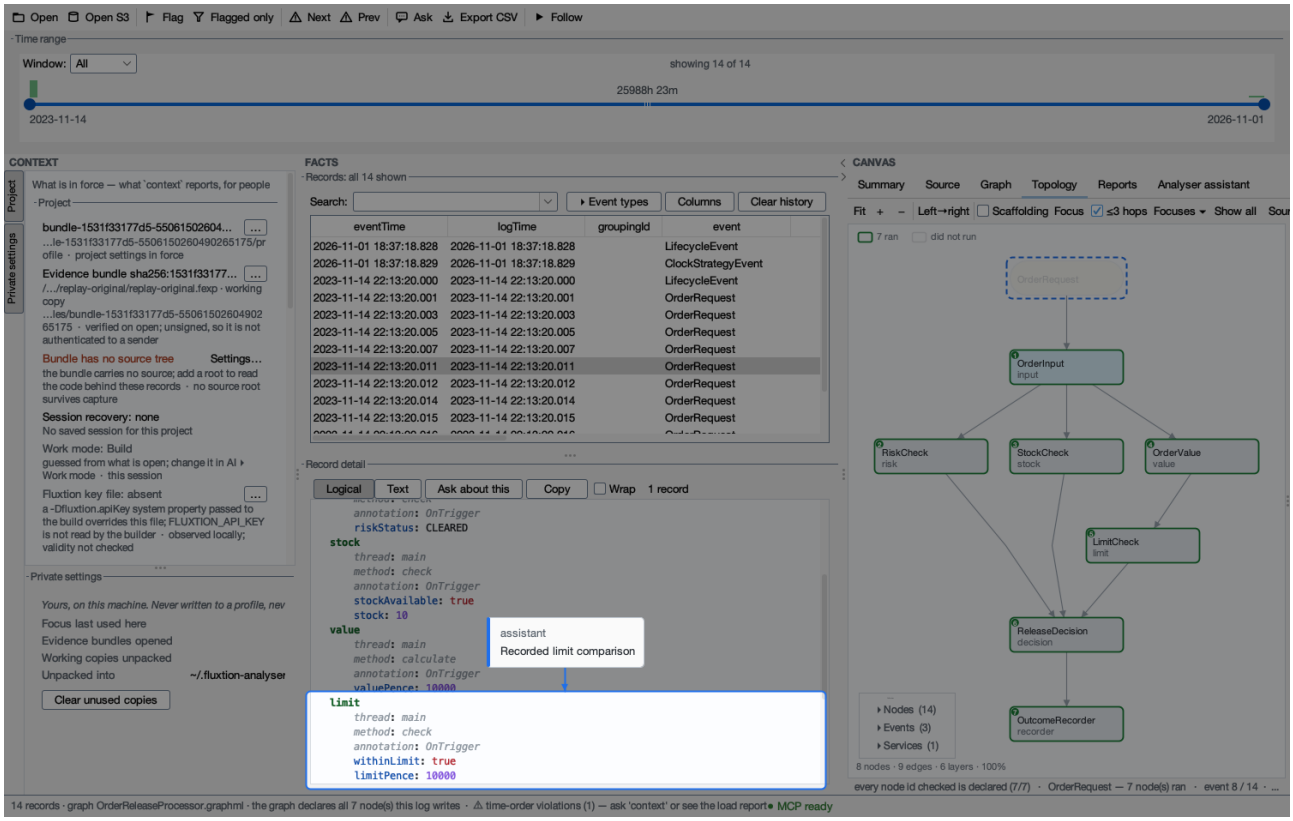
4. Evaluate a change

Apply a stricter limit rule and identify the two changed decisions.

5. Share the result

Hand over the runner, inputs, comparison and saved investigations.

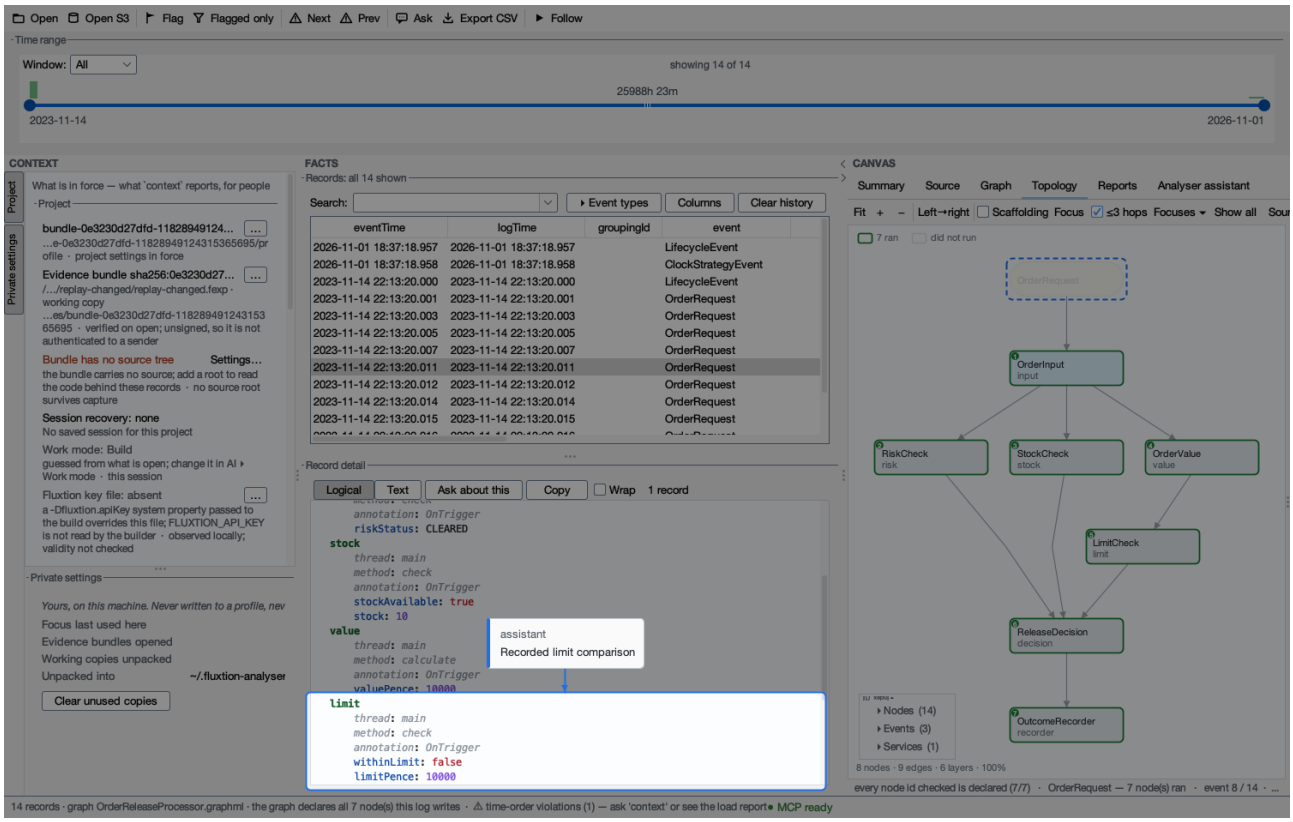
Original recorded values: valuePence 10000, limitPence 10000, withinLimit true, outcome ACCEPTED. Mixed clock sources make the timeline span unsuitable as an elapsed duration.



Open the full-resolution screenshot

VERIFY THE CHANGE

Changed-build recorded values: valuePence 10000, limitPence 10000, withinLimit false, outcome REJECTED_LIMIT. The application runner compares the recorded observations.



Open the full-resolution screenshot

These are genuine Analyser captures of separate application runs. The nine order observations match on same-build replay. Mixed initialization and fixture clocks make the displayed date span unsuitable for elapsed-time analysis.

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

What is an evidence bundle?

Why this matters

A reviewer can reproduce this captured scenario, see that state and event order matter, and identify the exact effects of a changed rule. In a pilot, the same discipline can support regression review and incident investigation when the necessary inputs and external conditions have been captured.

Apply it to your process

Bring one representative process, its expected outcomes and failure cases. Agree the checks and evidence before evaluating a candidate change; finish with an adopt, extend or stop decision.

Discuss a five-stage pilot

TECHNICAL APPENDIX

Reproduce this result

Unzip the package and run from its captured-replay directory:

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

The runner verifies the public runtime checksum, compiles the supplied generated Java and executes the checks locally. No generation credit, model request or network service is used. Use a new output directory.

Source and reproduction package for this example

Generated orchestration is included for offline execution. No hosted compiler is called by reproduction. Changing topology requires regeneration.

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

summary.json

audit-comparison.json

initial-state.properties

commands.json

captured-inputs.csv

observations.csv

Changed build investigation bundle

Same-build replay investigation bundle

Original captured run investigation bundle

All exported PDFs, screenshots and verification receipts

Scope and interpretation

Synthetic captured-input session, not replay of a production incident. Starting state means a fresh processor plus explicit host values, not arbitrary heap checkpointing.

Risk availability is a controlled synthetic dependency. Unrecorded service responses, randomness, uncontrolled clocks, shared concurrent state or irreversible effects require separate capture/control.

Nine order audit records are compared. Raw complete logs are not byte-identical; initialization and clock-install records contain wall-clock/object-identity differences.

The Analyser screenshots contain mixed initialization and fixture clocks. Their displayed multi-year span is not an elapsed-time measurement; follow record order.

The application runner executes replay. Opening a bundle or stepping through an HTML reel only reopens the saved investigation.

Member hashes detect changed contents; they do not authenticate the sender or attest an actual production deployment.

Example code is released under Apache 2.0. 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