

Document the Decision

Turning an API reliability lab into developer education

DOCUMENTATION & DEVELOPER EXPERIENCE

AMY V

14 OCTOBER 2026

An endpoint reference answers what a request accepts and returns. A companion tutorial can teach what a developer should do when the workflow is partial, interrupted, or uncertain. This article proposes a practical teaching and review process around an existing Olostep reference project.

EXPLAIN

Name the decision

REPRODUCE

Run the evidence

EVALUATE

Watch the reader

A useful tutorial leaves the reader able to make the next decision without guessing.

Independent portfolio writing prepared for Hamza and the Olostep team.
Not commissioned by, affiliated with, or endorsed by Olostep.

Choose a journey small enough to finish.

A technical article can be correct line by line and still leave the reader unable to finish a task. The missing ingredient is often a decision: which workflow to choose, what an unexpected result means, or when it is safe to continue. For developer education, I would organize the material around those moments rather than reproduce an endpoint catalogue.

The Olostep Integration Reliability Lab starts with known public URLs and follows them through an asynchronous batch. This removes source discovery from the first lesson. Finding the right pages and processing a known list are different problems; combining them would make a failure harder for a new reader to diagnose. Olostep already documents batching and related capabilities. This is a proposed companion learning path, not a claim that its documentation is absent or defective. [1]

A learning objective a reader can demonstrate

After the tutorial, a developer should be able to account for every expected item, explain a partial result, and resume a known operation without automatically recreating it. That objective is more useful than "understand the API": it describes observable behavior and tells the author what the exercise must contain.

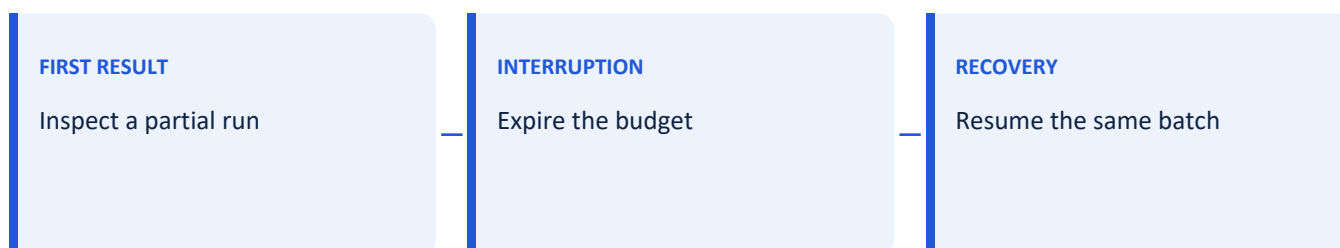
Reader question	Teaching artifact	Evidence to inspect
Did every source return?	An identity-reconciliation explanation.	Expected, observed, missing, and unexpected IDs.
Can I try again?	A creation-versus-read decision table.	Recorded attempt policy and retained batch ID.
Why is content missing?	A partial-result walkthrough.	Failed or unresolved item and next action.
Was this a live result?	A visible evidence label.	Fixture mode or bounded live report.

This table also defines an editorial test. If a page provides an impressive diagram but cannot answer one of these questions, the diagram is not doing enough work. If a code snippet only prints success, it needs either a narrower promise or a richer explanation of the result.

Teach one complete journey, then add breadth where reader evidence justifies it.

Teach the failure before it becomes a ticket.

I would publish the learning path in three layers. First, a short orientation describes the task and evidence boundaries. Second, an executable tutorial leads from a partial fixture to checkpoint recovery. Third, a reference section explains individual errors and links to the provider contract. A reader can choose the depth needed without losing the underlying model.



A support answer inside the tutorial

A vague message such as "something went wrong" forces the developer to reconstruct the system. A useful explanation names the observation, its implication, and the next action. The following is proposed documentation copy for the existing timeout fixture; it is not a report of a live Olostep incident. [2]

The polling budget expired before this client could finish verifying the batch. Keep the saved checkpoint and resume reads against that batch. Do not submit another batch solely because this read timed out. The current report is incomplete, not proof that the remote operation failed.

That paragraph works because each sentence resolves a different uncertainty. It identifies a local deadline, limits the inference about the provider, preserves an existing operation, and explains the report status. It gives the reader a safe next move without burying the cause in implementation details.

Match the format to the question

The article explains why counts are insufficient. The runnable example shows the manifest and returned IDs. The walkthrough makes a partial result visible. A troubleshooting entry provides a concise recovery instruction. These are different presentations of one verified behavior, not four unrelated content ideas.

I would keep the walkthrough in fixture mode so reviewers can replay a failure without credentials or credits. The article would link separately to the recorded live check. A polished demonstration should never require the reader to infer whether its behavior was simulated.

Give every important claim a home.

Technical accuracy is easier to maintain when each material claim points to a source or an executable assertion. For this project, I separate provider documentation, observed live responses, fixture behavior, and client policy. Unknowns remain a fifth category rather than being forced into one of the others. [3, 4]

Evidence category	Example	Permitted claim
Documented	Batch items and retrieval IDs in the public reference.	The reference describes this interface.
Observed live	Two URLs retrieved; returned order differed.	This bounded run behaved this way.
Fixture-tested	Cursor loop or transient read failure.	The client handled this synthetic condition.
Client policy	One creation attempt and bounded read retries.	This implementation chooses this behavior.
Unknown	Null usage and untested production delivery.	No measured cost or production guarantee follows.

These labels are editorial tools as much as interface labels. They prevent a synthetic latency number from becoming a performance claim, or a successful local deduplication test from becoming a promise about remote side effects. They also make review faster: an engineer can challenge the source of a statement without first debating its wording.

A concrete review loop with engineers

For a new guide, I would submit the intended reader outcome, the runnable example, and a short claim-to-evidence table together. An engineer reviews contract accuracy and failure boundaries. The writer checks terminology, sequence, and whether each error has an actionable next step. A final clean run confirms that the published commands still produce the described result.

The reference includes 42 offline tests, but a passing suite does not automatically validate every paragraph. Editorial review still needs to catch outdated names, incorrect test counts, links to earlier revisions, and claims that go beyond the fixture. Record the reviewed revision with the writing sample so a later code change does not silently invalidate the explanation. Contract changes and client-policy changes should trigger separate review questions.

For future maintenance, I would propose a small documentation check for commands, expected report fields, and link targets. That automation is a recommendation, not a new CI capability claimed by this writing sample.

Measure whether the reader can recover.

Page views can show reach, but they do not establish comprehension. The next useful experiment would be a small formative study with developers unfamiliar with the project. Give them the same offline scenario and ask them to explain the result and choose the next action. Record uncertainty as well as completion. No such study has been performed for this sample.

Task	Observable measure	Interpretation boundary
Explain a partial result	Correctly identify failed and unresolved items.	Shows understanding in this exercise only.
Resume after timeout	Use the existing checkpoint without new creation.	Does not establish production recovery skill.
Interpret null usage	State that cost is unknown.	Does not validate billing or account balance.
Locate supporting evidence	Find the live report and distinguish fixtures.	Shows discoverability, not general trust.

A first study could involve a few participants and short think-aloud sessions. Keep the task and environment stable; record where people hesitate, whether they need a hint, and which instruction resolves the hesitation. Report raw observations rather than treating a small sample as statistically representative. If a revised guide is compared later, account for learning effects from repeating the same task.

A contribution that can be reviewed today

The immediate package is already concrete: working source, a reproducible offline tutorial, a bounded live-compatibility record, and a short walkthrough. The proposed study would evaluate the teaching, not rescue an unfinished implementation. [2, 4, 5]

For Olostep, this is the contribution I would want to make: connect the API contract to the decisions developers face when operating an integration. Start with a specific journey, make uncertainty legible, give readers an executable recovery path, and improve the explanation using evidence from their experience. That approach supports technical writing and developer education without claiming support savings or adoption before measuring either.

SOURCES & REVIEW PATH

- 1. [Olostep batch workflow - provider documentation.](#)
- 2. [Local quickstart, fixture modes, and recovery commands.](#)
- 3. [Reference reliability model and evidence vocabulary.](#)
- 4. [Bounded live-compatibility record and untested areas.](#)
- 5. [Portfolio overview and recorded fixture walkthrough.](#)