

Integration Failures You Can Actually Work

When a message between utility systems fails, the team should be able to see it, fix the cause, and send it again.

PRODUCT

Grid Data Integration Suite

AUDIENCE

Integration and IT teams, CIS and billing leads, operations managers

The failures nobody sees

A utility runs on messages passing between systems. Meter reads travel from head-end systems into the meter data platform. Billing determinants move to the CIS, whether that is SAP IS-U or Oracle CC&B. Service orders go to work management. Settlement data goes to the market operator. When these hand-offs work, nobody thinks about them. When one fails, often nobody notices either, at least not right away.

Integration failures tend to be quiet. A move-in event is rejected because a premise record changed in the CIS overnight. A submission to the market operator times out and is never sent again. A head-end system starts formatting a field differently, and a transformation drops it without complaint. The symptom shows up days later as a customer call, an exception in the bill run, or a settlement dispute. By then the original message is buried in a log file, if it was kept at all.

Why this is hard to fix

Utility integration is a web of systems with different owners, formats and release schedules. Some interfaces are web services, some are file drops, some are message queues. Many were built as point-to-point scripts by people who have since moved on.

When something breaks, the evidence is split across teams. The integration team sees an error in a log. The billing team sees a missing charge. Neither can easily see the actual message that failed, the step where it failed, and why.

Finding the cause is only half the job. Once the mapping or the reference data is corrected, someone has to resend the specific messages that failed, safely, without re-running a whole batch or creating duplicates downstream. In many shops that means a developer, a database query and a nervous afternoon.

Change is risky too. Editing a live interface often means a restart and a maintenance window, so small fixes wait. And the payloads themselves carry names, addresses and account numbers. A logging approach that captures everything for troubleshooting can quietly become a privacy problem.

Principles of a good approach

Keep every failure, whole. A failed message should be captured with its payload and enough context to see where and why it failed. Without that, triage is guesswork.

Make failures workable. A dead-letter queue is only useful if people can work it like a queue of tasks. Inspect the failed payload, fix the cause, then replay the message. The fix and the replay should happen in the same place the failure is reported.

Let transient problems handle themselves. Many failures are blips: a timeout, a brief outage, a busy endpoint. Automatic retries should absorb those. When a target system is down for longer, a circuit breaker should stop calling it rather than turning one outage into a flood of failed messages.

Version every change, and change without restarts. Every change to an integration flow should be saved as a new version that cannot be altered, so everyone knows exactly what is running. Flows should be started, stopped and paused at runtime, without restarting anything else.

See the whole landscape. Teams need a map of every application, its endpoints, and the flows between them. It is the first thing you want during an incident, and it is hard to keep current by hand.

Audit everything, and protect personal data. Every change to a flow and every execution of it should be recorded. Personal information should be masked in audit logs and in dead-letter entries, so troubleshooting does not expose customer data to everyone who can open a log.

Let AI draft, and let people decide. AI can help write transformation code: describe the input, the output and the logic in plain English, and get a working draft. That draft is a starting point. An engineer reviews it, edits it, and saves it as a new version like any other change. Nothing reaches production because a model wrote it.

What it looks like in practice

ILLUSTRATIVE EXAMPLE (HYPOTHETICAL)

A utility sends move-in events from its CIS to its meter data platform. One morning, a batch of those events starts failing. The CIS has begun sending a new rate code that the transformation does not recognize. Retries do not help, because the problem is not transient, so the messages land in the dead-letter queue with their payloads and the error.

The integration analyst opens the queue. Every failure points to the same step and the same error. She opens one payload, with the customer's personal fields masked, and sees the new rate code. She describes the updated mapping in plain English, and the AI assistant drafts the change to the transformation. A senior engineer reviews the draft, adjusts one condition, and saves it as a new version of the flow.

The deployment is switched to the new version while it runs. There is no restart and no maintenance window. The analyst replays the failed messages from the queue, and they go through. The audit log records who changed the flow, which version is now running, and every execution, including the replays. On the architecture map, the link between the CIS and the meter data platform is healthy again.

Later that afternoon, the market operator's endpoint goes down for a while. Retries run first; then the circuit breaker holds further calls until the endpoint recovers, and nothing piles up in a way that needs manual cleanup.

Nobody wrote a script, searched a log file, or waited for the next release. The failure was a task, and the team finished it before it became a billing exception.

Questions to ask any integration vendor

- When a message fails, can we see the exact payload, the step that failed, and the error?
- Can we replay a single message or a selected group after fixing the cause? What stops a replayed message from being processed twice?
- Are retries and circuit breakers built in, or do we write them for each interface?
- Is every flow version kept unchanged? Can we pause, start or switch versions without a restart?
- Is there a live map of every application and flow?
- Is every change and every execution audited? Is personal data masked in logs and error queues?
- If AI writes transformation code, who reviews it, and is it versioned like any other change?

Building integrations your team can run

The Grid Data Integration Suite is the enterprise front door for the Grid Data family. Teams design flows visually, connect the CIS, head-end systems, work management and market operators, and run them with retries, circuit breakers, a workable dead-letter queue, versioned flows, an architecture map and a full audit trail built in. It feeds platforms that support more than 15 million meters and over 1 billion interval reads per day in production.

Learn more about the [Grid Data Integration Suite](#), or [talk to our team](#) about the interfaces that keep your people up at night.

ABOUT PERINIMBLE

PeriNimble builds mission-critical grid data software for energy and utility companies. Founded in 2022 and headquartered in Bloomington, Minnesota, we support more than 15 million meters and over 1 billion interval reads per day in production.

Built for systems that can't afford failure. · perinimble.com · perinimble.com/contact