



Process Truth

Product and Usage Guide: Discover, Diagram, Eval, Handoff

An open-core Claude Code plugin for AI-deployment Forward Deployment Engineers

June 2026

Contents

1. Purpose: Two Days or Nine?

- A scene you have probably lived
- What Process Truth is, in one breath
- The two bookends every AI rollout fumbles
- What you get out of it
- What Process Truth deliberately is not
- The shape of the deal
- The one paragraph to remember

2. How It Works: The Model

- Capture first, analyze second
- The Discovery Egress Broker: one locked door
- The deterministic core, and the hash that pins it
- Open core: MIT core, proprietary Pro Pack, AGPL kept at arm's length
- Model policy: Opus 4.8 only, behind the seam

3. The Full Flow: Discover, Diagram, Eval, Handoff

- Discover
- Diagram
- Eval
- Handoff

4. Safety and Your Data

- Will this touch production?
- Where does my data go?
- Redaction, quarantine, and the honest limits

5. How to Use It

- Step 1: Check the prerequisites
- Step 2: Install the plugin
- Step 3: Add your license (Pro commercial edition only)
- Step 4: Choose a mode
- Step 5: Install the pm4py extra and run the quickstart
- Step 6: Run the full pipeline and read your first map

6. The Deliverable You Hand Over

- Why these steps were chosen, and what the discrepancies revealed
- The limitation statement, quoted in full

The residual-risk register, counter-signed by two keys
Corpus provenance, coverage, and the frozen baseline
Reproducibility carve-out and the runbook
The scorecard says it passed

7. Editions and Licensing

The free core (MIT)
The Pro (commercial) edition (USD 5,000 per license)
How the offline license works
The anti-piracy posture
Where the legal language lives

8. FAQ and Glossary

The questions people actually ask
Glossary

Appendix A. The Non-Negotiable Invariants

Appendix B. Build Status and Provenance

About this guide

1. Purpose: Two Days or Nine?

◆ A scene you have probably lived

Imagine you are an engineer who just got dropped into Acme Insurance. Your job is to help them put artificial intelligence to work on their claims process. On day one, a vice president tells you, with total confidence, that a claim takes two days from the moment it arrives to the moment it pays out.

That afternoon you sit with the people who actually do the work. They laugh. It takes nine days, they say, and most of that is waiting. A claim bounces between three systems, gets re-keyed by hand twice, and parks in someone's inbox over the weekend.

So which is it, two days or nine? Nobody in the building can give you a straight, evidence-backed answer. Leadership has the org chart's version. The frontline has the lived version. The truth is somewhere in the data, and nobody has gone and gotten it. This is the moment Process Truth was built for.

◆ What Process Truth is, in one breath

Process Truth is a plugin for Claude Code, used by a Forward Deployment Engineer. A Forward Deployment Engineer, or FDE, is the person a company like Anthropic sends to sit on-site with a customer and actually make a deployment work. Throughout these docs we will just call that person "the engineer."

Process Truth does one focused thing. It figures out how work really happens inside a company and turns that into clear, reviewable diagrams plus an honest measurement of how long things take today. It does this by reading the evidence, not by collecting opinions. That is the whole pitch. It is deliberately not trying to be more than that, and this chapter explains why holding that line is the entire point.

◆ The two bookends every AI rollout fumbles

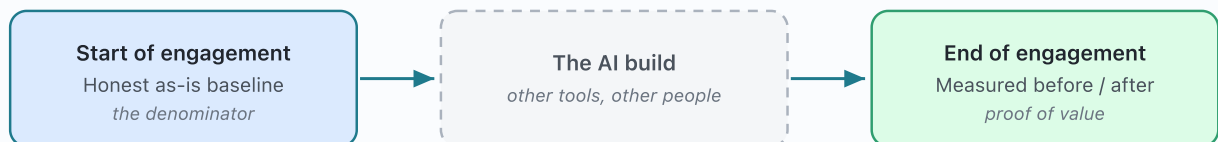
Every AI deployment is a story with a beginning and an end. The messy middle, building the actual AI, gets all the attention. The beginning and the end get skipped, and that is exactly where projects quietly fail.

The opening bookend is the honest baseline. Before you can claim an AI saved time, you need to know how long the work took without it. That number is the denominator for everything that follows. At Acme, that denominator is the answer to "two days or nine?"

Without it, any future claim of improvement is just a louder opinion. The spec calls this the empirical as-is baseline: the real step durations, the real volumes, and the rework loops where work gets redone. “Empirical” simply means measured from evidence rather than asserted from memory.

The closing bookend is the proof of value. At the end, leadership wants to know whether the money was well spent. To answer that honestly you need a fair before-and-after comparison, and the “before” has to be the same baseline you froze at the start, not a new number invented to flatter the result.

Process Truth calls itself the ground-truth and proof-of-value layer because it deliberately owns exactly these two bookends. “Ground truth” means the version of reality you can point to in the data and defend in a meeting, as opposed to the version people believe.



The two bookends. Process Truth owns the honest baseline at the start and the measured proof of value at the end. The dashed middle box, building the AI itself, is the part it deliberately does not do.

The shaded middle box is the part Process Truth does not do, and that matters as much as what it does.

◆ What you get out of it

Process Truth produces one central artifact and two diagrams drawn from it. The central artifact is the Process Spec, the single source of truth for how the process works, assembled from three lanes of evidence (interviews, system event logs, and screen recordings). “Single source of truth” means everything else, every diagram and every later decision, points back to this one document instead of drifting off into separate versions.

From that spec, Process Truth renders two pictures. The first is a Data Flow Diagram, or DFD, which shows the systems, where data is stored, and how it moves between them, including where it crosses a trust boundary or carries sensitive personal data. The second is a process-flow diagram, a lightweight version of the standard BPMN notation (BPMN being the common visual language for drawing business processes), which shows the steps, who does each one, the decision points, and the paths things take when something goes wrong. The

lanes in that diagram, the bands that show who owns each step, are called swimlanes. You also get the honest baseline numbers as a natural byproduct of all that discovery.

◆ What Process Truth deliberately is not

Here is the discipline that makes everything above hold together. Process Truth captures ground truth and proves value. It does not build the AI application itself. It does not build the retrieval pipelines that feed a model documents (often called RAG, for retrieval-augmented generation). It does not write or tune the prompts that go into the deployed system. It does not build the agent. Those are real and demanding crafts, and they belong to the engineer and the rest of the toolchain.

The spec is firm on this. Process Truth feeds the prototype and proves the value, and it does not build the agent. Any feature request that drags it toward becoming the whole AI development platform is out of scope by default and has to justify itself against that boundary.

This restraint is a feature. A tool that owns one job, the honest beginning and the honest end of an engagement, can be small enough to audit, safe enough to run inside a regulated company, and trustworthy enough that its baseline numbers mean something. A tool that tries to do everything is none of those things. Automation is bounded the same way: Process Truth can recommend what is worth automating and rank the candidates, but actually executing automation against live systems is a later phase, not part of version 1.

◆ The shape of the deal

Process Truth follows an open-core model. Open-core means a free, openly licensed core with a paid layer on top. The framework and a starter set of capabilities are free under the permissive MIT license. A proprietary Pro pack of deeper capabilities costs 5,000 US dollars per license.

On safety, the spec leans on a deterministic core. "Deterministic" means the same input produces the same output, with no randomness in the parts that matter. The structural facts in the Process Spec come from plain, repeatable code, so the frozen baseline stays comparable forever. On the same kind of hardware and pinned toolchain it reproduces byte-for-byte, meaning a rerun yields the same exact file hash (a deterministic core hash). Across different hardware it reproduces value-equivalently, where the activities, ordering, and counts match even if the hash differs.

On data, Process Truth runs in one of two ways. Either it does deterministic-only work with no AI at all, the mode the spec calls `inference:none`, or it runs Claude Opus 4.8 on the

customer's own cloud account through AWS Bedrock, so the data stays in the customer's account and there is no phone-home.

◆ The one paragraph to remember

Acme could not tell you whether their claims process took two days or nine. Process Truth exists to answer that kind of question with evidence, to draw the real picture clearly, to set an honest starting line and an honest finish line for an AI project, and then to get out of the way. It owns the two bookends nobody else bothers to get right, and it deliberately leaves the middle to the people whose job that is.

2. How It Works: The Model

Before you run Process Truth on anything, it helps to hold the mental model in your head. The whole tool is built around a handful of decisions that show up everywhere once you know to look for them. This chapter walks through those decisions so the rest of the guide reads as obvious rather than surprising.

Process Truth is the FDE's ground-truth layer. An FDE (Forward Deployment Engineer) is the engineer who embeds on-site with a customer to deliver an AI project. Ground truth here means the real, as-is process plus an honest empirical baseline of how long the work takes and how often it goes wrong. Empirical baseline means a number measured from evidence, not asserted in a meeting. Process Truth establishes that ground truth and proves whether later work paid off. It does not build the AI agent or write the production prompts.

◆ Capture first, analyze second

Discovery is split into two phases that never overlap. The capture phase is a narrow, non-AI program that reads and records near the live system, under your supervision. The analyze phase, where all the AI work happens, runs in a sealed-off space with no route back to any customer system. The thing that looks at live systems has no AI in it, and the AI has no path to live systems.

This split is the reason the safety story is not “the model decided to behave.” The component near production cannot reason its way into a bad action because it has no reasoning in it. The component that reasons is on the other side of a wall.

◆ The Discovery Egress Broker: one locked door

Every action discovery takes goes through a single broker (the Discovery Egress Broker, or DEB). It is default-deny: if a tool is not supposed to be used, it is not handed to the agent in the first place. You cannot misuse a tool you were never given.

The DEB is also where the inference seam lives. In version 1 the only mode that ships is deterministic-only, written `inference:none`, where no AI runs at all and you get just the reproducible core and the diagrams it drives. Under that mode the seam raises

`InferenceUnavailableError` and performs no egress, so in v1 there is no live egress path at all. The DEB additionally enforces a provider-SDK backstop: it asserts that no model-provider client library (the list covers `anthropic`, `boto3`, `botocore`, `openai`, `cohere`, `litellm`, `google`, and `azure`) has been imported, both at DEB import and at the egress chokepoint.

◆ The deterministic core, and the hash that pins it

The Process Spec is the single artifact everything hangs off, and it has two halves. The core is built only from deterministic code (deterministic meaning the same input always produces the same output, with no judgment calls in between). It comes from pm4py and from plain, reproducible Python doing arithmetic on a frozen, fingerprinted copy of the event log. It holds the activities, their order, the variants, the counts, the bottlenecks, the rework loops, and a coverage manifest.

Because it is deterministic, the core is reproducible, which is what lets it serve as the baseline you defend to a customer months later. That reproducibility is pinned by a deterministic core hash: a fingerprint of the core that should not move unless the core itself changes. The frozen golden core hash is `5c6692d8...923ed`, and it has not changed across P1, P2, P3, P4, or the later security audit remediation. The runnable `examples/quickstart.py` produces a stable discovery hash across runs.

The annex is the other half. It holds everything an AI model wrote (the plain-language summaries, the vision-model labels, the fusion narrative) and is marked advisory, with each piece stamped with which model produced it and a human-sign-off flag that defaults to no. The annex and the per-license watermark are woven only into the advisory annex and the human-readable deliverable, so neither touches the core hash.

◆ Open core: MIT core, proprietary Pro Pack, AGPL kept at arm's length

Process Truth ships open-core: a permissively licensed core (MIT) plus a separately licensed commercial layer. The `pro/` tree is the proprietary part, wrapped with ChaCha20-Poly1305 and gated by entitlement. CI guards keep the two apart, including a gate that checks no Pro cleartext leaks into the MIT path.

The process-mining library, pm4py, is AGPL-3.0. It is never shipped in the MIT wheel. It is an opt-in `[pm4py]` extra and runs subprocess-isolated, with `mining/pm4py_subprocess.py` as the single canonical importer. The transitive dependency cvxopt (GPL-3.0-or-later) is disclosed in the legal docs. The point of the isolation is that the copyleft library lives behind a process boundary rather than inside the core.

◆ Model policy: Opus 4.8 only, behind the seam

When AI does run (in a later mode, not v1), Process Truth uses one model, Claude Opus 4.8, and only that model. Two other models, Fable 5 and Mythos 5, are deliberately excluded because they are under an export suspension that would route customer data back to the

model provider. Every inference call routes through the DEB seam, so the model policy is enforced at the chokepoint, not by convention.

The whole codebase has been through phase reviews and a cross-cutting security audit, with the full suite standing at 570 Python plus 29 TypeScript tests after the audit fixes. Hold these five ideas (the two-phase split, the one locked door, the hashed core, the open-core boundary, and the single model behind the seam) and the rest of Process Truth follows from them.

3. The Full Flow: Discover, Diagram, Eval, Handoff

This chapter walks the whole tool, once, end to end. It follows one synthetic procure-to-pay log through the four stages of the v1 pipeline, Discover, Diagram, Eval, and Handoff, so you can see exactly what comes out the far side before you ever point Process Truth at real work. A quick reminder of what the tool is: Process Truth is what an FDE (a forward-deployed engineer, the person a vendor sends in to make a deployment actually land) runs to capture the real current state of how work happens. It establishes the ground truth, the evidence-backed account of what the data shows, that later automation projects get built on top of. It is open-core, meaning the engine is open source and the commercial edition adds the licensed pieces around it.

You can reproduce everything below yourself. The discovery half needs the optional `pm4py` extra:

```
uv pip install "pm4py>=2.7,<2.8"
```

Then run the full pipeline:

```
.venv/bin/python examples/full_pipeline.py    # Discover -> Diagram -> Eval -> Handoff
```

`full_pipeline.py` runs offline, reads only the synthetic log baked into the script, and writes its artifacts to `examples/sample_output/`. Run it twice and the files are byte-identical. Nothing here touches a production system and no data leaves your machine, because the inference mode is `inference:none` (a setting that means no AI model runs at all, so the run is purely deterministic code).

◆ Discover

The engine takes a small event log, three procure-to-pay cases of four steps each, and derives the deterministic core: the activities, the directly-follows flow edges with their honest observed counts, the process variants, and a coverage manifest. None of this is inference. It is `pm4py` mining (reconstructing the real sequence of steps from the recorded events) plus deterministic Python over the log you provided. Before any of that runs there is an integrity pre-flight on the input, and capture-time redaction strips sensitive data on your own machine before anything moves downstream.

The whole core is summarized by one fingerprint, the deterministic core hash:

```
deterministic_core_hash: cad7aca5b16a2214b1e41f482f98e7cbcd59ebcb5c2c499bf67a4b4f4b8a
```

That hash is stable across runs and across hardware. It is the load-bearing thing the rest of the pipeline leans on, and it is what the eval baseline signs in the third stage.

Coverage is reported honestly rather than smoothed over. The sample log observed three of an estimated four annual cases, so the verdict is `partial` and the report says so out loud instead of implying it saw the whole year. Entity resolution lines up the actors and systems across the lanes, and connectivity is read straight from the transports the log actually recorded.

◆ Diagram

The diagram layer projects that deterministic core into a render model and emits three views of the same flow. The swimlane diagram (a swimlane being a map that groups each step under the actor responsible for it) places the four steps under their lanes: approver, requester, procurement, and finance. The data-flow diagram, or DFD (a view that shows systems and the transports moving data between them rather than human steps), shows the two systems observed in this run, CRM and ERP, with the `api` and `file` transports between them. The same flow is also exported as BPMN 2.0 XML (BPMN being the Business Process Model and Notation standard that other process tools consume).

Every label and identifier is sanitized before it is written. A step named after a diagram keyword, for example `end`, is rendered as a safe identifier rather than parsed as syntax, so a process name cannot corrupt the output.

◆ Eval

The eval stage is the bookend that lets you certify the map, and it has two parts.

First is the signed baseline (`signed_baseline.json`). It is computed over the deterministic map only, never over any model output, which is why it is called mode-blind: it certifies the reproducible core regardless of which inference mode a customer runs later. It binds the core hash and is signed with Ed25519:

```
{
  "baseline": {
    "case_count": 3,
    "coverage_verdict": "partial",
    "deterministic_core_hash":
      "cad7aca5b16a2214b1e41f482f98e7cbcd59ebcb5c2c499bf67a4b4f4b8adbe7",
    "variant_count": 2
  },
  "signature_hex":
    "f3e783410dc1f0e6bd12a7df003c0e71262adb5d62e569f68b6cdad71b3614e5a..."
}
```

`verify_baseline` returns `True` against the signing key.

Second is the scorecard (`eval_scorecard.json`). The golden cases assert deterministic facts about this run, the mined activities, the variant paths, and the flow edges, and they are graded by deterministic graders only. Under `inference:none` there is no model, so the LLM judge is non-applicable and contributes nothing to the passing score. The scorecard passes the CI gate, meaning the gate exit code is `0` :

```
{
  "accepted": true,
  "passed": 4,
  "total": 4,
  "pass_rate": 1.0,
  "threshold": 1.0,
  "per_grader": { "exact-match": { "passed": 3, "total": 3 }, "schema": {
    "passed": 1, "total": 1 } }
}
```

Four golden cases, four accepted, against a threshold of `1.0` . The scorecard is stamped with its dataset version, `p2p-capstone-v1` .

◆ Handoff

The handoff bundle pairs the map with its liability instruments and renders the customer deliverable (`handoff_deliverable.md`). This is the thing you actually hand over. It states why the steps were chosen, what the discrepancies revealed, and, plainly, what the artifact does and does not certify:

It certifies that Process Truth did not modify production. It is NOT a statement of how the process actually works, NOT a certification that controls are effective, and NOT an audit opinion.

The residual-risk register (`residual_risk_register.json`) is parameterized by the run mode and counter-signed by two keys, the FDE and the customer.

`verify_countersigned_register` returns `True` . Because this run was `inference:none` , the egress-disclosure row is correctly marked not applicable, while redaction-best-effort, anti-piracy, and audit-suppression stay applicable and are disclosed in plain language.

The deliverable also carries the frozen baseline hash, a corpus-provenance stamp with the coverage gap stated, a reproducibility carve-out (only the deterministic tier is reproducible, the advisory annex may drift on re-run), and a deterministic-core runbook for re-deriving the baseline offline with zero credentials.

From this one offline run you walk away with a deterministic process map and its signed, mode-blind baseline, three diagram views, a golden-case scorecard that passed its gate, and a counter-signed handoff deliverable that says honestly what it certifies and what it does not. None of it required network access, an LLM, or any change to a production system.

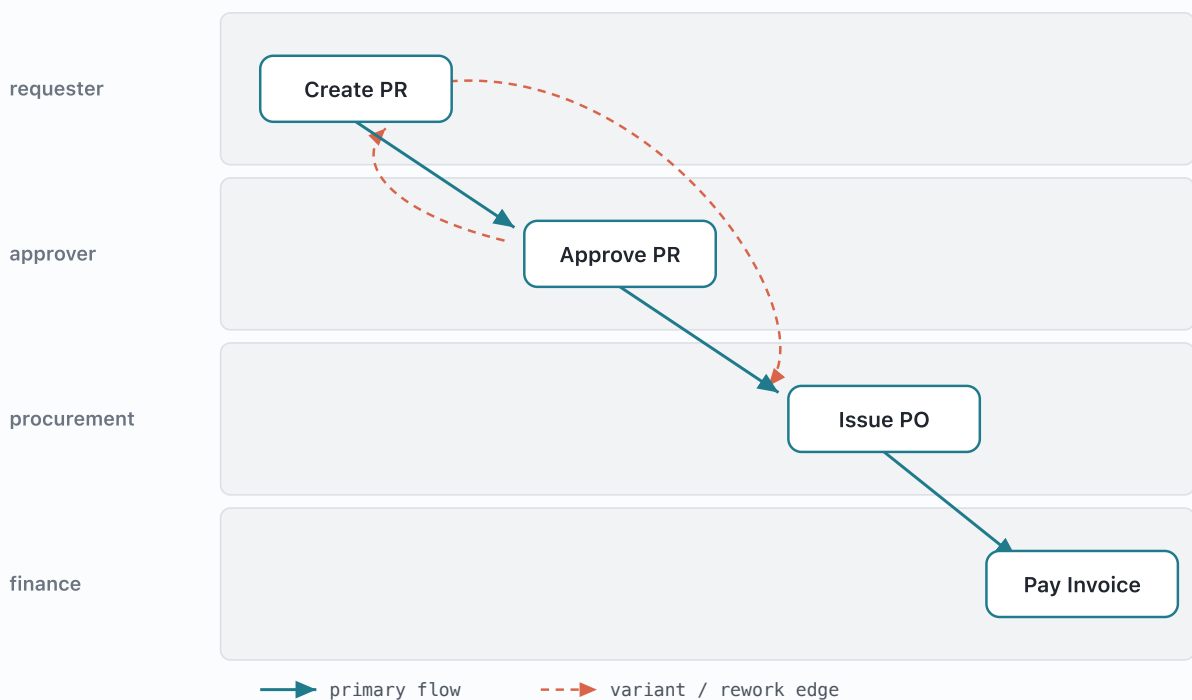


Figure 1. The swimlane process-flow view, generated from the deterministic core (`examples/sample_output/process_flow.mmd`). Each step sits in its actor lane; solid arrows are the primary flow, dashed arrows are the variant and rework edges the mining surfaced.

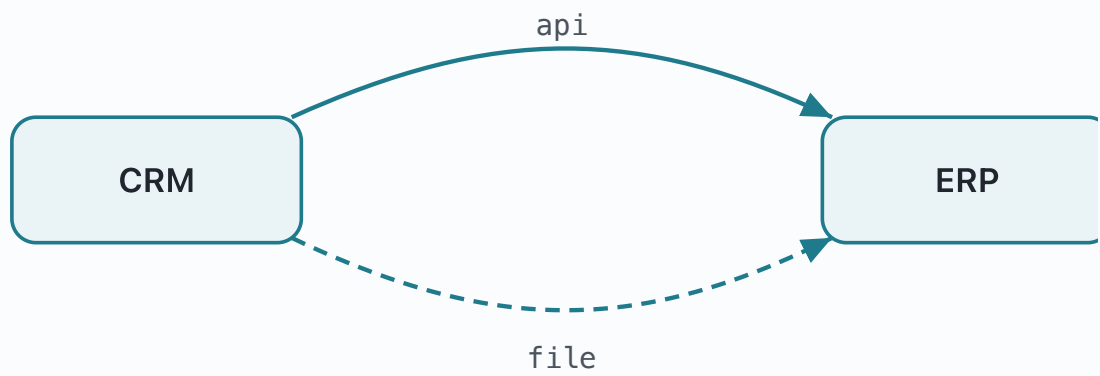


Figure 2. The data-flow diagram (`examples/sample_output/dfd.mmd`): the system-to-system links observed in the log, here CRM writing to ERP over two transports (api and file).

4. Safety and Your Data

Before any team brings in Process Truth, two questions come up, and they are the right ones. Will this touch our production systems? And where does our data go? This chapter answers both, in plain words, with the honest edges left visible.

A quick orientation if you are new here. Process Truth is a tool an engineer (often called an FDE, a forward-deployed engineer who works on site with the customer) uses to map how work actually happens inside a company, then draws that map as diagrams. It does its mapping by reading three things: interviews, the logs your systems already write, and recordings of people's screens.

◆ Will this touch production?

No. Process Truth looks, it never touches. The whole safety promise fits in four plain words.

Read-only. It can read data. It cannot write, change, delete, or create anything on your systems. The credentials it connects with have no write permission at all, granted by your own database and identity systems, not by Process Truth's good intentions.

Deterministic. This is a technical word for a simple idea: same input, same output, every time. Feed it the same exported log twice and you get the identical map twice. (There is one honest caveat about which parts are deterministic, covered below.)

Never touches the live system. Process Truth does not study your running production system. It studies a frozen copy, a point-in-time snapshot or an offline export that your team hands it.

No added load. Because it works on a frozen copy, it does not compete with your real users for database connections, CPU, or query budget. Reading a snapshot on the side does not slow down the people filing claims.

It also does no scanning. It ingests only artifacts your team has already captured and handed across. There is no by-construction path that quietly reaches into production.

The reason you can believe this is that touching is not a matter of trust. Every request passes through one gate, the Discovery Egress Broker (DEB), where "egress" means traffic leaving toward your systems. That gate is default-deny: closed unless a request is on a short, signed, read-only allowlist, and anything else is dropped. The write tools are never loaded into the engineer's session in the first place, so there is nothing there to write with. The broker also forces every database read to be read-only at the database itself, and your own database roles would refuse a write from that account even if every other layer failed.

◆ Where does my data go?

There are three modes, and you choose which one before any work begins. Underneath all three is one rule that does the heavy lifting: sensitive data is scrubbed on the operator's laptop, before anything leaves it.

The default and most private mode is deterministic-only, named `inference:none`. Here no AI runs at all and there is no inference traffic to worry about. Process Truth mines your event logs and draws diagrams using ordinary software (the open-source library `pm4py` and supporting Python). Nothing is sent to a model, because there is no model. The trade-off is real: you get the deterministic map (the activities, their order, the variants, the bottlenecks, the rework loops), but not the AI-assisted summaries, labels read off screen recordings, or narrative.

If you later enable AI, the primary mode is `customer-bedrock`, where the model runs inside your own AWS account, in your region, under your access controls, reaching the model over a private network link (PrivateLink) that never crosses the public internet. The model provider does not receive your data, AWS does not store or train on it, and the author of Process Truth never receives it. There is no phone-home. When AI is enabled, traffic still routes only through the single egress broker to your cloud.

A third path, `first-party-provider-share`, would send data to the model provider directly. It is avoided for any sensitive corpus and is off unless you explicitly turn it on, for non-sensitive material only.

◆ Redaction, quarantine, and the honest limits

The genuinely sensitive material is not just text. It is the pixels in screen recordings: a name, a date of birth, a diagnosis, a partial card number in a form field. Process Truth handles this with capture-time redaction. Personal information, health information, payment-card data, and credentials are detected and removed from frames and text on the operator's own machine, before anything is written to disk and before any AI sees it.

The design fails safe. If the redaction detector errors out on a frame, that frame is dropped or quarantined rather than passed through, and the quarantine is encrypted at rest. The default is to lose data, not to leak it. High recall is the goal, and a measured minimum is recorded for each engagement.

Be clear-eyed: redaction is best-effort, not a guarantee. No automated detector catches everything, and we say so plainly rather than implying perfection. It is backed by data minimization, encryption at rest, and an audit log. Cloud guardrails (AWS Bedrock Guardrails) are a backup layer only, never the real control, because they work on text and do not look inside the OCR'd frames of a screen recording.

The honest framing on the model: Process Truth uses Claude Opus 4.8 only, pinned to a specific dated version. And the honest framing overall is customer-owned cloud, not air-gapped. Your data does not reach us, and in the deterministic-only and Bedrock modes it does not reach the provider either, but the operator's laptop holds the corpus in memory while it works, and the plugin still fetches updates and revocation lists over the network. We will always tell you the difference.

5. How to Use It

This chapter is the hands-on part. It takes you from a fresh machine to a real diagram drawn from real data, in order, with the exact commands. If you are an FDE (forward deployment engineer, the person who runs Process Truth on a customer engagement), this is the chapter you keep open while you work.

The whole pipeline has four stages in this version: Discover, Diagram, Eval, and Handoff. Steps 1 through 4 below get you set up. Step 5 runs a discovery-only quickstart. Step 6 runs the full pipeline and explains the map you get back.

◆ Step 1: Check the prerequisites

You need a working Claude Code setup and a few basics.

- Claude Code, installed and working. Process Truth is a plugin that lives inside it.
- Python, because the part that does the heavy mining is built on a Python library called `pm4py`.
- A place to keep the work isolated. Process Truth sets up an encrypted, per-engagement workspace for each customer so one client's data never mixes with another's. You just need disk space and the ability to let it create that workspace.
- A way to run inference, or a decision not to. You will choose this in Step 4. You do not need to decide right now.

◆ Step 2: Install the plugin

Process Truth follows the open-core model. Open-core means the foundation is free and open source, and a premium add-on is paid. The framework, the core discovery pipeline, the diagram tooling, and a starter set of capabilities are free under the MIT license (a permissive open-source license you can use freely). A separate proprietary Pro pack adds a deeper library of discovery capabilities, fusion playbooks, and the eval-harness pack. It costs USD 5,000 per license and comes with a fixed window of updates.

Install the free core the way you install any Claude Code plugin in your environment. The core gives you a complete discovery and diagram pipeline, and you can run a full engagement on it alone. This guide does not use the Pro edition.

◆ Step 3: Add your license (Pro commercial edition only)

If you are running on the free core, skip this step. You have everything you need.

If your team bought the Pro (commercial) edition, you activate it with a license. The Pro content is encrypted ("crypto-wrapped") and carries a hidden per-license watermark, so the content stays locked until your license unlocks it and a leaked copy can be traced back to its source. The license binds to your specific device through your machine's secure hardware, and it can be revoked when you leave the engagement. Activation details live in `08-
editions-and-licensing.md`.

◆ Step 4: Choose a mode

This is the most important decision before you run anything, because it determines whether any data ever leaves the operator's machine. There are three modes.

Mode	What it does	When to use it
<code>inference:none</code>	Deterministic mining only. No AI model is used at all.	The strictest environments, where the customer forbids any third-party AI processing.
<code>customer-bedrock</code>	Claude runs inside the customer's own AWS cloud account.	The normal cloud-permitted choice. Data stays in the customer's cloud.
<code>first-party-provider-share</code>	Claude runs in Anthropic's own cloud, and the customer's data reaches Anthropic.	Avoided for sensitive data. Only for explicitly non-sensitive use.

Two terms. Deterministic means same input, same output, every time, with no randomness and no judgment call. The deterministic part of Process Truth is reproducible: feed it the same event log and you get the same map. That reproducible map is the official baseline you later use to prove whether an automation helped. The AI-written parts live in a separate, clearly labeled section called the advisory annex, which is helpful but not authoritative and not reproducible word-for-word.

When a mode does use AI, Process Truth uses Claude Opus 4.8, and only that model.

`inference:none` is the easiest place to start: it needs no cloud setup, produces the solid deterministic map, and lets you see the core of the tool working.

◆ Step 5: Install the pm4py extra and run the quickstart

The discovery half of the pipeline needs the optional pm4py extra. Install it once:

```
uv pip install "pm4py>=2.7,<2.8"
```

Now run the quickstart, which is discovery only and prints each step as it goes:

```
.venv/bin/python examples/quickstart.py      # Discover, printed step by step
```

This reads a synthetic event log baked into the script, so nothing about a real customer is at stake. An event log is a table where each row is one thing that happened, with at minimum a case id (which claim or order the event belongs to), an activity (the step that happened), and a timestamp. A fourth column, the resource or actor (who performed the step), is worth fighting for, because without it Process Truth cannot check controls like whether the same person both submitted and approved a payment.

◆ Step 6: Run the full pipeline and read your first map

The second example runs all four stages and writes its output to disk:

```
.venv/bin/python examples/full_pipeline.py    # Discover -> Diagram -> Eval ->
Handoff
```

`full_pipeline.py` runs offline, reads only the synthetic log in the script (three procure-to-pay cases, four steps each), and writes seven artifacts to `examples/sample_output/`. Run it twice and the files are byte-identical. The inference mode is `inference:none`, so no data leaves your machine.

The seven artifacts it writes are:

File	Stage	What it is
<code>process_flow.mmd</code>	Diagram	The swimlane process map (Mermaid).
<code>dfd.mmd</code>	Diagram	The data flow diagram (DFD), showing system-to-system flows.
<code>process.bpmn</code>	Diagram	The same flow as BPMN 2.0 XML.
<code>signed_baseline.json</code>	Eval	The mode-blind, Ed25519-signed baseline that binds the core hash.
<code>eval_scorecard.json</code>	Eval	The golden-case scorecard and its accept/reject verdict.
<code>handoff_deliverable.md</code>	Handoff	The customer report. This is the thing you hand over.
<code>residual_risk_register.json</code>	Handoff	The two-key counter-signed liability register.

The discovery step (run by `quickstart.py`) also writes an eighth file alongside these, `core_deterministic.json`, the hashed, authoritative process core.

The deterministic core (the deterministic core hash is the single fingerprint of the whole map) is reported as:

```
deterministic_core_hash: cad7aca5b16a2214b1e41f482f98e7cbcd59ebcb5c2c499bf67a4b4f4b8a
```

That hash is stable across runs and across hardware, and it is what the Eval baseline signs. A swimlane (a lane on the diagram that groups each step under the actor or system that performed it) keeps the map readable. The swimlane map in `process_flow.mmd` groups each step under its actor lane (approver, requester, procurement, finance), and the DFD in `dfd.mmd` shows the systems (CRM, ERP) and the transports observed between them.

Read the map the right way. It is what the data showed over the window you were given. It is not an audit. The deliverable states this limitation in plain language:

It certifies that Process Truth did not modify production. It is NOT a statement of how the process actually works, NOT a certification that controls are effective, and NOT an audit opinion.

Coverage is reported honestly too. The sample log observed three of an estimated four annual cases, so the coverage verdict is **partial** and the report says so out loud rather than implying it saw everything. That honesty is the feature: it lets you put the map in front of both the working team and leadership and have a real conversation about the gap between what they each believe.

6. The Deliverable You Hand Over

This chapter walks the real deliverable a customer receives, section by section, using the actual generated sample in the repository. The file is

`examples/sample_output/handoff_deliverable.md`, produced by the FDE (Forward Deployment Engineer, the specialist who comes on-site to do the discovery work and then leaves). It is supported by two machine artifacts in the same folder:

`residual_risk_register.json` and `eval_scorecard.json`. Everything below is what those files actually say.

The sample was generated under `inference:none`, meaning no AI inference happens. Only the deterministic core ran. ("Deterministic" means same input, same output, every time.) The named internal owner recorded in the file is the customer process owner, so the tool does not rot once the visiting engineer leaves.

◆ Why these steps were chosen, and what the discrepancies revealed

The artifact opens by explaining the analysis it ran. The process under study is procure-to-pay, mined deterministically from the provided event log. Activities and the flow edges between them derive only from observed directly-follows ordering (which activity was seen immediately after which), with no inference layered on top.

Then it tells you where the data is thin. Coverage is partial: 3 observed cases against a known annual volume of 4, so the window understates the full process. This is the kind of honesty that distinguishes ground truth (what the data actually shows) from a tidy story. The deliverable does not hide that one case in four never showed up.

◆ The limitation statement, quoted in full

Every Process Spec carries a limitation of findings in writing, so it can never be misread as something it is not. Here is the sample verbatim.

Limitation of findings. This Process Spec certifies only what the provided corpus showed over the window 2026-01-01T00:00:00+00:00 to 2026-01-31T23:59:59+00:00, under these assumptions: converted 0 non-UTC timestamps to UTC; dropped 0 exact-duplicate events (retries/webhook replays); deterministically ordered 2 events sharing a (case, timestamp); flagged 0 machine-batch timestamp window(s) (≥ 50 events); flagged 0 reversal(s) as compensating actions; observed 3/4 known cases (ratio 0.75). It certifies that Process Truth did not modify production. It is NOT a statement of how the process actually works, NOT a certification that controls are effective, and NOT an audit opinion.

Read it once and the boundary is clear. It certifies what the corpus showed over a stated window under stated assumptions, and it certifies that Process Truth did not touch production. It does not certify how the process truly works, that controls are effective, or anything resembling an audit opinion.

◆ The residual-risk register, counter-signed by two keys

The register names a pre-handoff owner (FDE) and a post-handoff owner (Customer process owner), and it lists the required signatories as FDE and Customer. The JSON file carries both `fde_signature_hex` and `customer_signature_hex`, so the register is two-key counter-signed, not just asserted by the vendor. The customer holds a key, and their signature is on the record.

The register has four rows. Three are marked applicable, one is not:

- `redaction-best-effort` (applicable): capture-time redaction is best-effort and is not a guarantee of perfect recall, bounded by the configured recall floor and mitigated by minimization, at-rest encryption, and the audit ledger.
- `egress-disclosure` (not applicable): no provider-share egress in this mode, because no inference lane is active.
- `anti-piracy` (applicable): v1 anti-piracy is deterrence and attribution only.
- `audit-suppression` (applicable): the v1 default audit detects suppression rather than prevents it.

The `egress-disclosure` row reading `applicable=False` is correct, not an omission. Under `inference:none` no AI lane runs, so there is no data leaving for a provider to share. If you later turn AI on, that row becomes live again.

◆ Corpus provenance, coverage, and the frozen baseline

The artifact stamps where the data came from. Run id `run-capstone-0001`, operator `fde-demo`, coverage verdict partial. The coverage gap is spelled out: aggregate completeness 0.75 (3 of 4 known cases), with 2 intake cases that never appeared in any lane (abandoned or off-system). A swimlane here is one row per person or system in the process picture, so “never appeared in any lane” means those cases left no trace anywhere the tool could see.

Then comes the empirical baseline, the measured starting point your team holds the future against. The frozen baseline records a deterministic core hash (a short cryptographic fingerprint of the deterministic output) of

`cad7aca5b16a2214b1e41f482f98e7cbcd59ebcb5c2c499bf67a4b4f4b8adbe7`, a case count of 3, a variant count of 2, total rework of 0, and total bottleneck wait of 0.0 seconds. These are the numbers that do not drift.

◆ Reproducibility carve-out and the runbook

The artifact is precise about what reproduces. Only the deterministic tier is reproducible. It is value-equivalent across hardware, meaning identical canonical activities, ordering, variants, and counts, and re-derivable forever from the frozen input. The advisory annex (the AI-written commentary clipped alongside the core) is produced by a language model and may differ on re-run due to model drift, and the artifact states plainly that such a difference is expected and is not a regression.

The deterministic-core runbook tells your engineers how to re-derive the baseline offline with zero credentials: load the frozen `CoreDeterministic` and capture window, then call `processtruth.runnable.run.deterministic_run(core, window, seam)` with an `inference:none` seam. To activate LLM lanes on a fresh corpus, you must pass the mode-lock interlock with a fresh customer counter-signature first, so turning AI on is always a deliberate, recorded act.

◆ The scorecard says it passed

The matching `eval_scorecard.json` confirms the run was accepted. It shows `accepted:true` on dataset version `p2p-capstone-v1`, with 4 of 4 checks passed, a pass rate of 1.0 against a threshold of 1.0. By grader, exact-match passed 3 of 3 and schema passed 1 of 1. The gate failed zero checks. The deliverable you hand over is one that cleared its own bar before it left your hands.

7. Editions and Licensing

Process Truth, the Claude Code plugin built for the Forward Deployment Engineer (FDE, the engineer dropped into a company to make an AI deployment succeed), comes in two halves. The first half is free and open source. The second is a paid add-on at USD 5,000 per license. This chapter explains what each half includes, how the license is enforced, and where the honest limits sit. The job both halves serve is the same: capture how work really happens, turn it into reviewable diagrams plus a measured starting point, and prove value. Neither half builds the AI agent or runs the chatbot.

◆ The free core (MIT)

The free edition is the framework plus a starter set of tools. It lives in the `core/` part of the project and is released under the MIT license, one of the most permissive open-source licenses, so you can use, modify, and build on it with very few obligations.

What you get for free:

- The framework that runs the Discover then Diagram pipeline.
- The interfaces every part plugs into, so you can extend the tool without rewriting it.
- The diagram schemas (the rules that turn a Process Spec into a Data Flow Diagram, DFD, and a process-flow diagram). The Process Spec is the single source of truth everything else is generated from.
- The Discovery Egress Broker (DEB), the single controlled gate all data traffic passes through. It keeps discovery read-only and is open source so a customer's security team can inspect it.
- The deterministic pipeline plumbing. Deterministic means the same inputs always produce the same output, so the core is reproducible.
- A starter dimension set. A dimension is one specific thing the tool knows how to look for in the data, such as a kind of bottleneck or handoff.
- A best-effort stub for tacit-rule extraction. Tacit rules are the unwritten judgment calls people make. The free pass at uncovering them is marked advisory.

This is enough to run a real discovery, produce a real Process Spec, and generate real diagrams. It is not a crippled demo.

◆ The Pro (commercial) edition (USD 5,000 per license)

The Pro (commercial) edition is a separate add-on sold under a proprietary EULA (End User License Agreement). It lives in a separate `pro/` repository that is never mixed into the free MIT code. The USD 5,000 buys a deep, curated dimension library refined from real engagements, Fusion playbooks (Fusion reconciles the three discovery lanes, what people say, what system logs show, and what screen recordings reveal, and flags where they disagree), the eval-harness pack that builds a scored test set from your captured data, and curated tacit-rule packs that go beyond the free stub. One license includes a fixed update window, after which renewing keeps updates flowing. This split is what people mean by open-core: a useful free foundation with paid depth on top.

◆ How the offline license works

Buying runs through Polar.sh, with the website, checkout, and license issuance running on Vercel serverless functions, so there is no always-on server. When payment clears, Polar issues a key that looks like `PROCESSTRUTH_xxxxxxxx` and shows it in your Polar Customer Portal. You also receive an offline, signed version of the license for air-gapped sites with no internet. On a connected machine, Process Truth checks the key with Polar and counts it against your seats. On an offline machine, it verifies the signed token locally with no call out. Either way, the key binds to that specific machine. Polar acts as the Merchant of Record (the official seller that collects and pays sales tax, VAT, and GST), but it does not encrypt the Pro content. Polar answers how many seats were paid for, and Process Truth answers whether the content is allowed to open on this machine.

◆ The anti-piracy posture

The Pro content is encrypted, not shipped as plain files. To use it, your machine unwraps it using a two-level key scheme. Each license carries an entitlement (proof you paid), and from that entitlement a per-device key is derived inside secure hardware (a TPM or Secure Enclave, the tamper-resistant chip in modern laptops). The plain unwrap key never exists outside that hardware, and content is decrypted only into memory, never written back to disk as readable files. Because device keys are issued by the vendor, an engineer cannot quietly move a license to a new machine, and offboarding revokes device keys.

There is also a per-license watermark, a hidden, traceable marker. It is woven only into the advisory, human-readable parts of the output and kept out of the deterministic core. The deterministic-core files stay un-watermarked and byte-identical for every licensee on purpose, because watermarking the core would break the promise that the same inputs reproduce the same output. Revocation uses an offline revocation list (CRL) checked when you update, plus a grace window and a signed expiry date as a fallback for machines that are

rarely online. A build-time check refuses to ship if paid content lands in the free MIT code, and an output filter blocks attempts to make the model repeat its proprietary prompts verbatim.

The honest limit: anti-piracy on software that runs on a customer's own machine cannot be made unbreakable. Once code runs on hardware someone else controls, a determined attacker can eventually reach it. The encryption, watermark, and EULA are deterrence and attribution. They raise the cost and help prove where a leak came from, but they do not make piracy impossible. This is priced and contracted as an accepted residual risk. One choice supports it: Process Truth ships no local model weights. Inference runs on the customer's own cloud or does not happen at all, which keeps the encryption scheme coherent.

◆ Where the legal language lives

This chapter is the plain-language map. The binding text lives in the root `LICENSE` file (MIT, for the free core) and the `legal/` directory, which holds the proprietary EULA for the Pro edition, the AGPL-3.0 handling for the pm4py mining engine (and its cvxopt GPL-3.0-or-later transitive dependency), and the liability instruments. Before you distribute Process Truth to anyone, the pm4py AGPL-3.0 dependency must be reviewed so the open-core split stays clean. The MIT warranty disclaimer covers the free core only, not the Pro edition or the deliverable you hand a customer.

8. FAQ and Glossary

This is the page you read after someone in a meeting asks “wait, what is this thing actually going to do to our systems?” and you need a straight answer in plain words. A few terms first, defined once so the rest reads smoothly. Process Truth is a plugin for Claude Code (Anthropic’s engineering tool) that an engineer runs to discover how work really happens inside a company and turn it into reviewable diagrams plus a baseline of measured facts. An FDE is a Forward Deployment Engineer, the person (often from an AI vendor or a consultancy) embedded on-site to make a deployment land. Discovery is the act of finding out how the work really happens, done passively, by looking, not by touching.

◆ The questions people actually ask

Will this slow down our production systems?

No, and that is built into how the tool is shaped. Discovery is passive and read-only: the credentials it gets carry no permission to write, change, or administer anything. The work is split into two phases that cannot blur together. The phase that goes near your live systems is a narrow program that only reads. The thinking phase (the AI part) runs in a sandbox with no route back to your systems. The preferred way to run discovery is against a frozen snapshot or an offline export, not the live system. A live read is a deliberate exception your team has to counter-sign, under hard limits on query time and pull size. The honest caveat: “zero impact” is the design target enforced by removing capability, not a promise that software can never fail. See 04-we-wont-break-your-systems.md.

Does our data leave our network?

It depends on which mode you choose, and you choose. There are three. `inference:none` uses no AI at all, only the deterministic engine that crunches event logs into a process map, so nothing is sent for AI processing. This is the fit for the strictest environments. `customer-bedrock` runs the AI inside your own AWS account, on your AWS Bedrock service, in your region, so the data goes to your cloud, not ours. This is the recommended mode for AI features, and the company that makes Process Truth never receives your data. `first-party-provider-share` would reach Anthropic’s cloud directly, avoided for anything sensitive. The honesty this deserves: `customer-bedrock` is “your own cloud,” not air-gapped. Those are different things. See 05-where-your-data-goes.md.

How is our private data actually protected?

The real protection happens early and locally. When Process Truth captures a screen recording or pulls text, it runs redaction on the FDE’s own machine first, automatically finding

and blacking out personal data, health data, payment card numbers, and credentials before anything is written to disk or seen by AI. The cloud-side safety filters (AWS Bedrock Guardrails) do not do this job: they work on text and do not look inside screen-recording images, which are the bulk of the sensitive material. So capture-time redaction is the actual control; the cloud filters are a backstop. Honest caveat: redaction is best-effort, not a guarantee. See 05-where-your-data-goes.md.

Is this an audit? Can we use the diagram as a compliance record?

No. The diagram and the underlying Process Spec (the single written description of the process) are, in plain terms, “this is what we saw in the data,” over the window you gave us, under the assumptions we wrote down. They also certify that Process Truth did not modify your production systems. The spec does not claim the process works that way in some absolute sense, does not claim your controls are effective, and is not an audit opinion. Every spec carries a limitation-of-findings statement and a residual-risk register that both you and the FDE sign. See 08-editions-and-licensing.md.

Can our own team run this after the FDE leaves?

Yes. The deterministic core (the part that turns event logs into a process map and renders the diagrams) runs with zero credentials, offline, forever. It is config-driven, with no secret paths hardwired to the original setup. You also get a handoff artifact: a written record of why each step was chosen, what the discrepancies revealed, the full run manifest, and a runbook for re-running the core, naming a budgeted internal owner. Two honest notes: switching on AI mode against fresh sensitive data means re-signing the risk register, and the AI-written annex may read slightly differently if re-run later, because the model evolves. The deterministic part is the stable, re-derivable bookend. See 07-handing-it-over.md.

Why not the newest, most powerful model?

Because stable and export-clean beats bleeding-edge here. Process Truth uses Claude Opus 4.8 only, pinned to a specific dated version rather than a rolling “latest” label. The flashier-sounding Fable 5 and Mythos 5 are excluded: as of mid-June 2026 they are under a US export-control suspension (so they cannot lawfully serve non-US customers), and the only access path routes your data through a first-party cloud that retains it. Opus 4.8 is a separate, generally available model, not part of that suspended family. The startup check verifies the pinned model is available, export-clean, and set to no-data-share-for-training, and fails closed rather than falling back. See 05-where-your-data-goes.md.

Can Process Truth produce the same answer twice?

For the part that matters most, yes. Output splits into two pieces. The deterministic core is produced by plain repeatable code (the process-mining library pm4py, plus deterministic image and text processing): same frozen input, same map. The advisory annex is everything

the AI wrote (suggested labels, summaries, hypotheses), clearly marked as advisory and gated behind a human before it drives automation. One precise detail: the core reproduces byte-for-byte identically on the same hardware and toolchain, and value-equivalently across different hardware (same activities, ordering, variants, and counts even when the file hash differs because floating-point math is not bit-identical across processors). See 04-we-wont-break-your-systems.md.

Will this build our AI agent for us?

No. Process Truth is the ground-truth and proof-of-value layer. It captures the real as-is process and an honest baseline of how long things actually take, so a deployment can be scoped against reality. It does not build the RAG pipeline, tune production prompts, run model bake-offs, harden your runtime, or generate the deployed agent. It feeds the prototype and proves the value at the end. See 01-the-why.md.

Does this include the automation that fixes the process?

Not in version 1. The pipeline is Discover, then Diagram, then Automate. Version 1 ships Discover and Diagram. The Automate phase is specified at the interface level but not built yet. When it arrives it comes with a dry run, a reviewable diff, human approval, and a rollback path, with larger-blast-radius changes requiring your counter-signature. For now, v1 produces a ranked shortlist of what is worth automating, not the automation itself. See 02-how-it-works.md.

What is the difference between what people told you and what you found?

This is the heart of what Process Truth is for. Discovery runs three lanes at once. The elicitation lane is interviews, SOPs, and wikis: what people say. The mining lane is pm4py reading exported system event logs: what the systems logged. The task-mining lane is AI watching screen recordings: what was observed on screen. A step called fusion reconciles the three and surfaces, with evidence, where the stated process diverges from the observed or logged one. The most valuable category it flags is done-but-not-said: the step everyone does but nobody wrote down. That discrepancy engine is the wedge. See 03-discovery.md.

What does it cost, and what is free?

It is open-core. The framework, interfaces, diagram schemas, deterministic plumbing, and a starter set of dimensions are free under the MIT license. A proprietary Pro (commercial) edition (the deep curated library, fusion playbooks, eval-harness pack, tacit-rule packs) costs USD 5,000 per license with a fixed update window. The Pro edition is protected with crypto-wrapping and per-license watermarking, which are deterrence and attribution, not an unbreakable lock. One legal note: pm4py is licensed AGPL-3.0 (copyleft, stronger than GPL-3.0, with a network-use trigger), and it pulls in cvxopt (GPL-3.0-or-later) on older Python

versions. Raise this with legal early if you package or redistribute. See 08-editions-and-licensing.md.

What happens if redaction or egress fails?

It stops. Process Truth fails closed across the board. If the redaction detector errors on a frame, that frame is dropped or quarantined. If AI mode is on but the consent gate is not signed, no data is sent. If the access-logging system cannot be reached, no AI call happens, because the access trail must be written first. The safe failure is the one that does nothing. See 04-we-wont-break-your-systems.md.

◆ Glossary

Advisory annex: The AI-written part of the Process Spec (suggested labels, summaries, hypotheses). Marked advisory, not authoritative, and may not drive automation until a human signs off.

BPMN / BPMN-lite: Business Process Model and Notation, a standard visual language for process diagrams (boxes for tasks, diamonds for decisions, lanes for who does what). “Lite” is a simplified version. Process Truth renders in this style and can export full BPMN 2.0 XML.

DEB (Discovery Egress Broker): The single enforced chokepoint every outbound action passes through during discovery. Default-deny, it logs every system-touching call and is the one seam where the AI mode is chosen.

Deterministic core: The authoritative, reproducible part of the Process Spec, produced only by plain repeatable code (pm4py plus deterministic image and text processing), with no AI judgment. It drives the diagrams and the frozen baseline.

Deterministic core hash: The core reproduces byte-for-byte identically (identical file hash) on the same hardware and toolchain, and value-equivalently across different hardware.

DFD (Data Flow Diagram): One of the two diagrams. It shows systems, data stores, the flows between them, trust boundaries, and where personal or residency-sensitive data crosses an edge.

Empirical as-is baseline: The real “before” picture (step durations, volumes, rework loops) measured during discovery. Frozen on the deterministic map only, so it stays a stable comparison point. The denominator for proving value later.

FDE (Forward Deployment Engineer): The person who runs Process Truth, embedded on-site to make a deployment land. Treated as a “semi-trusted insider” for security purposes because they hold a lot of access.

Ground truth: The real, evidence-backed picture of how the process works, as opposed to opinion or the official story. Process Truth supplies this layer.

inference:none : The egress mode where no AI runs at all; only the deterministic engine. The honest answer for the strictest information-barrier policies (for example, MNPI). The core works fully; the annex is absent.

Mode-blind baseline: The customer-facing baseline is frozen on the deterministic core only, so the same baseline holds regardless of which AI mode was run.

Open-core: A business model where the core is free and open-source while certain advanced add-ons are paid and proprietary. The framework is MIT-licensed; the Pro edition is a USD 5,000 paid license.

pm4py: The open-source Python process-mining library Process Truth uses to turn event logs into a process model. Runs under deterministic Python control, not an AI agent. Licensed AGPL-3.0 (copyleft, stronger than GPL-3.0); pulls in cvxopt (GPL-3.0-or-later) on older Python versions.

Pro (commercial) edition: The paid, proprietary add-on (USD 5,000 per license): the deep curated dimension library, fusion playbooks, eval-harness pack, and tacit-rule packs. Protected by crypto-wrapping and watermarking (deterrence, not an unbreakable lock).

Swimlane: The lane in a process diagram showing who does what (a BPMN convention for assigning tasks to actors).

Two-phase barrier: The hard separation between CAPTURE (the narrow, non-AI program allowed near your systems) and ANALYZE (all AI work, in a sandbox with no route back to your systems).

Appendix A. The Non-Negotiable Invariants

These properties hold across every stage of Process Truth and are enforced by tests and continuous-integration guards. They are the promises the tool keeps no matter what corpus it is pointed at.

- **Passive, read-only, zero production impact.** Discovery never scans your network or your assets. It only ingests artifacts you have already captured (event logs, documents, screen recordings).
- **Deterministic hashed core.** The deterministic core hash covers a fixed subset of the analysis. No wall-clock time, randomness, or iteration order ever enters the hashed or rendered output, so the same input yields the same result on any machine.
- **Fail-closed.** Redaction, consent, the license counter-signature, the kill switch, offboarding, and the egress broker all deny on error. Nothing falls open.
- **Single egress chokepoint.** The Discovery Egress Broker is default-deny. The inference seam ships frozen to `inference:none`, and no model-provider code is loaded outside that seam.
- **Open-core.** The core is MIT licensed. The proprietary Pro Pack and the copyleft `pm4py` dependency are kept on the correct side of the line (`pm4py` is isolated in a subprocess and is never shipped in the MIT package).
- **Model policy.** Only Opus 4.8 runs behind the seam. Other model families are excluded by policy.

Appendix B. Build Status and Provenance

This guide describes software that is built and tested, not a proposal.

- Phases P0 through P4 are all built, reviewed, remediated, and accepted.
- A whole-codebase cross-cutting security audit (eight dimensions, adversarially verified) found zero critical and zero high findings, and four medium findings that are now closed.
- The test suite is green: 571 Python tests and 29 TypeScript tests.
- The frozen golden core hash `5c6692d8...923ed` has not changed across any phase or the audit, which is the determinism guarantee in practice.
- The worked sample referenced throughout this guide lives in `examples/sample_output/`. It is generated by `examples/full_pipeline.py` and is byte-identical across runs.
- The authoritative, detailed status (and every place the build diverged from the original design) is recorded in `docs/design/AS-BUILT-STATUS.md`.

◆ About this guide

This document synthesizes the product storyline in `docs/process-truth/` (chapters 01 through 10) into a single reference. It was generated in June 2026 and is reproducible from source with `docs/guide/build.sh` (Markdown plus a shared stylesheet, rendered to HTML and PDF). When a detail here disagrees with the code or with `docs/design/AS-BUILT-STATUS.md`, those win.